ENOVIA DesignSync Command Reference All Volume 1

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 Design Development
- URL Sync Object Model
- TCL Interface
- Third-Party Integrations
- Administration
- ProjectSync Data Manipulation
- Glossary

Table of Contents

ENOVIA Synchronicity Command Reference 1
Using this Guide with Different Methodologies 1
Module Based Commands 1
Legacy Module Based Commands 1
File Based Commands 2
Organization of the Command Reference 2
Syntax Description 3
Accessing Command Descriptions from Client Shells 4
Fundamental Topics 5
Overview of hcm Commands 5
hcm Commands 5
Overview of Module Commands 6
module Command 6
Understanding Fetch Preference 8
fetch preference 8
Understanding Server-Side Commands 10
server-side 10
Using Interrupt (Control-c) 12
interrupt 12
Using Revision Control Keywords 13
keywords 13
Using Selectors 16

selectors 16
Using Wildcards 28
wildcard 28
Client Applications 31
DesSync 31
DesSync Command 31
dss 33
dss Command 33
dssc 35
dssc Command 35
stcl 38
stcl Command 38
stclc 42
stclc Command 42
Client Shell Control 47
alias 47
alias Command 47
exit 48
exit Command 49
log 50
log Command 50
more 54
more Command 54

prompt 56
prompt Command 56
rstcl 57
rstcl Command 57
record 62
record Command 62
Module-Based Design 65
Workspace Setup 65
Enterprise Design Development Area 65
Module Views 84
Exclude from Workspace 96
populate 106
setfilter 197
setselector 202
setroot 211
setvault 214
setview 219
Primary Revision Control 221
add 221
cancel 230
ci 241
mkmod 283
populate 292

showmods 383
showstatus 396
tag 417
Advanced Revision Control 444
duplicatews 444
exportmod 450
freezemod 453
import 454
importmod 458
lock 461
migratetag 463
mkbranch 467
mkedge 475
mkfolder 479
mvmember 481
mvmod 489
purge 491
reconnectmod 503
remove 506
rmedge 516
rmfile 518
rmfolder 521
rmmod 525

rmversion 535
rollback 541
select 545
setfilter 548
setowner 553
switchlocker 555
unfreezemod 559
unlock 560
unremove 569
unselect 572
upgrade 575
upload 592
Navigational 604
cd 604
pwd 605
Module Hierarchy Management 606
Module Swapping 606
Whereused Information 628
addbackref 654
addhref 659
edithrefs 676
reconnectmod 688
rmhref 691

showhrefs 696
Informational 718
Whereused Information 718
annotate 744
compare 748
compare-foreach 771
contents 773
contents-foreach 788
datasheet 790
diff 792
help 804
locate 807
ls 811
ls-foreach 859
showhrefs 861
showmcache 875
showmods 881
showstatus 891
showproduct 912
showlocks 914
syncinfo 919
version 928
vhistory 929

vhistory-foreach ... 949
vhistory-foreach-obj ... 951
webhelp ... 953
Workflows ... 956
SITaR ... 956
Legacy Module-Based Design ... 1023
hcm ... 1023
hcm Commands ... 1023
hcm addhref ... 1024
hcm addhref Command ... 1024
hcm Example ... 1032
hcm Example ... 1032
hcm get ... 1041
hcm get Command ... 1041
hcm mkalias ... 1062
hcm mkalias Command ... 1062
hcm mkconf ... 1065
hcm mkconf Command ... 1065
hcm mkmod ... 1070
hcm mkmod Command ... 1070
hcm put ... 1073
hcm put Command ... 1073
Description ... 1079

hcm release 1089
hcm release Command 1089
hcm rmalias 1099
hcm rmalias Command 1099
hcm rmconf 1101
hcm rmconf Command 1101
hcm rmhref 1104
hcm rmhref Command 1104
hcm rmmod 1109
hcm rmmod Command 1109
hcm showconfs 1113
hcm showconfs Command 1113
hcm showhrefs 1120
hcm showhrefs Command 1120
hcm showmcache 1129
hcm showmcache Command 1129
hcm showmods 1133
hcm showmods Command 1133
hcm showstatus 1136
hcm showstatus Command 1136
hcm upgrade 1151
hcm upgrade Command 1151
Index 1157

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
- Legacy Module based Commands
- File based Commands

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.

Fundamental Topics

Overview of hcm Commands

hcm Commands

NAME

```
hcm                 - Commands to manipulate legacy module data
                      (legacy)
```

DESCRIPTION

```
These commands allow you to create and manage your legacy modules.

To display a list of available 'hcm' commands, do the following:

 dss> hcm <Tab>

Notes:
 o  The 'hcm' commands are available from all DesignSync client shells.
 o  The legacy module, 'hcm' commands, must be typed with the hcm
    prefix. The non-legacy module commands are not typed with an hcm
    prefix.  For more information on non-legacy module command, see
    module.

Accessing Help
--------------
You can access information about the 'hcm' commands from the command
line.
- To display complete information for an 'hcm' command, do any of the
  following:

  dss> help "<commandName>"

  dss> <commandName> -help

  dss> <commandName> -?

  The following examples return help information for the hcm Example
  command. The double quotes are required, and there must be exactly
  one space between 'hcm' and 'Example':

  dss> help "hcm Example"

  dss> hcm Example -help

  dss> hcm Example -?

- To display brief (synopsis) information for an 'hcm' command, do
```

```
either of the following:

dss> help -brief "<commandName>"

dss> <commandName> -usage

The following examples return help information for the hcm addhref
command. The double quotes are required, and there must be exactly
one space between 'hcm' and 'addhref':

dss> help -brief "hcm addhref"

dss> hcm addhref -usage
```

SYNOPSIS

```
hcm <hcm_command> [<hcm_command_options>]

Usage:  hcm (addhref|addlogin|doc|get|mkalias|mkconf|mkmod|put|release|
            rmalias|rmconf|rmhref|rmlogin|rmmod|showconfs|showhrefs|
            showlogins|showmcache|showmods|showstatus|upgrade|version)
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
dss, dssc, help, stcl, stclc
```

EXAMPLES

```
See specific 'hcm' commands.
```

Overview of Module Commands

module Command

NAME

```
module              - Commands to manipulate module data
```

DESCRIPTION

```
These commands allow you to create and manage your modules and module
members. They do not need to be typed with a command prefix, such as
hcm. If hcm legacy mode is enabled, you cannot type module commands
with the "hcm" prefix; legacy module commands, by contrast, are
always prefixed by hcm.  For more information on legacy module
commands, see hcm.

Note: The module commands are available from all DesignSync client
shells.

Accessing Help
--------------

You can access information about the module commands from the command
line.
- To display complete information for a module command, do any of the
  following:

  dss> help "<commandName>"

  dss> <commandName> -help

  dss> <commandName> -?

  The following examples return help information for the add
  command.

  dss> help add

  dss> add -help

  dss> add -?

- To display brief (synopsis) information for a module command, do
  either of the following:

  dss> help -brief "<commandName>"

  dss> <commandName> -usage

  The following examples return help information for the addhref
  command.

  dss> help -brief addhref

  dss> addhref -usage
```

SYNOPSIS

```
<module_command> [<module_command_options>]

Usage: [add|addbackref|addhref|addlogin|edithrefs|get|lock|migratetag|
        mkedge|mkmod|mvmember|remove|rollback|rmedge|rmhref|
        rmlogin|rmmod|showconfs|showhrefs|showlogins|showmcache|
        showmods|showstatus|whereused|unremove|upgrade|version]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
dss, dssc, help, stcl, stclc
```

EXAMPLES

```
See specific module commands.
```

Understanding Fetch Preference

fetch preference

NAME

```
fetch preference    - How to specify a default fetch state
```

DESCRIPTION

```
DesignSync supports five object states: locked copy (original),
unlocked copy (replica), link to the cache, link to the mirror,
and reference. You have a locked copy when you want to edit an
```

```
object to create a new version. You can also have a locked
reference to lock an object that you intend to regenerate
without fetching the last version of the object. The state
you want for unlocked objects -- objects you have fetched
into your work area -- typically depends on your team's
project methodology. For example, if your team employs a
file cache to share files, you would generally have links in
your work area to objects in the cache.

You can specify the object state each time you invoke a
command that affects the state (ci, co, populate, cancel) --
for example, always specifying -share. To simplify specifying
your team's recommended fetch state, your project leader can
define a default fetch state for these commands. Your project
leader uses the SyncAdmin tool to define a default fetch state.

DesignSync determines the fetch state for a given operation
as follows:
  1. Operations use the state option (-lock, -get/-keep, -share,
     -mirror, -reference) specified on the command line.
  2. If no state option is specified, operations use the default
     fetch state as defined by your project leader.
  3. If no default fetch state has been defined, operations use
     the command's default behavior (-get for 'co' and 'populate',
     -keep for ci and 'cancel').
The 'co' and 'populate' commands update the states of modified
objects only.  To update the states of all objects, use the
-unifystate option.

Note:
Use of caches and mirrors is limited to UNIX
machines. DesignSync displays an error message when a
command tries to use an invalid default fetch preference.
```

SEE ALSO

```
ci, co, populate, cancel
```

EXAMPLES

```
This example demonstrates the behavior of the 'co' command
first without, then with a default fetch state defined.

- No default fetch state defined
  dss> co top.v         # Fetches a copy (default behavior for 'co')
  dss> co -get top.v    # Fetches a copy
  dss> co -share top.v  # Creates link to top.v in the cache

- Default fetch state of 'mirror'
  dss> co top.v         # Creates link to top.v in the mirror
  dss> co -get top.v    # Fetches a copy (explicit option overrides default)
  dss> co -share top.v  # Creates link to top.v in the cache
```

Understanding Server-Side Commands

server-side

NAME

```
server-side          - Understanding and using server-side commands
```

DESCRIPTION

```
Server-side only commands can be run only on the SyncServer. You
run server-side scripts either from your browser, or using the
rstcl command from a DesignSync client:

o From your browser, specify a URL as follows:
  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 server-side script.

The SyncServer looks for stcl scripts in the following locations (in the
order listed):

  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. 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.

Important: 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.

You must verify access controls explicitly in server-side scripts.
Access controls are generally ignored in server-side scripts;
the script itself must call the 'access verify' command for access
controls it wishes to honor.  The following server-side script
verifies the built-in AdministrateServer access control:

      if [access verify AdministrateServer $SYNC_User] {
          access reset
          puts "AccessControl files reread."
      } else {
          puts "Permission denied."
      }

If your server-side script operates on RevisionControl notes, you need
to protect the integrity of your data by blocking access to the server
while the script runs:

1. Edit your custom AccessControl file to deny all actions that operate
   on RevisionControl notes, for example:

   access deny Checkin everyone
   access deny Tag everyone

2. Load the modified AccessControl file by using the ProjectSync Access
   Reset menu item.

3. Run your server-side script.

4. Edit your custom AccessControl file and remove the 'access deny'
   commands.

5. Load the modified AccessControl file by using the ProjectSync Access
   Reset menu item.

See the 'access' commands for more information about custom
AccessControl files.

The following are examples (but not the exhaustive list) of
server-side only commands:
  url users
  url notes
  note setprops
  note query
  access verify
  access reset

For more information on server-side development, see the ENOVIA
Synchronicity stcl Programmer's Guide.
```

SEE ALSO

```
rstcl
```

EXAMPLES

```
The following is an example of passing parameters into a
server side Tcl script.

Assuming the script uses the TclScript panel, the arguments are
passed using the same syntax that browsers use to pass parameters
to CGI scripts. This makes it simple to invoke TclScript panels
from HTML forms.

All of the arguments are packaged as members of an array variable
named SYNC_Parm. The index to the array is the name of the
argument, and the value is the argument value.

For example, to pass the color and shape of an object to a script
called 'DrawShapes.tcl', you would have something like this:

The URL to invoke the script (this example is meant to be a single
line but is displayed across two for enhanced readability):

http://holzt:2647/scripts/isynch.dll?panel=TclScript&file=DrawShapes.tcl
   &color=red&shape=triangle

Within DrawShapes.tcl, you might print out the parameters:

  puts "Color = $SYNC_Parm(color) <br>"
  puts "Shape = $SYNC_Parm(shape)"

In this example, submitting the URL should produce the output:

  Color = red
  Shape = triangle

You could also execute this script using the rstcl command from a
DesignSync client (dssc in this example):

  dss> rstcl -server sync://holzt:2647 -script DrawShapes.tcl \
       -urlparams color=red&shape=triangle
```

Using Interrupt (Control-c)

interrupt

NAME

```
interrupt           - How to interrupt commands
```

DESCRIPTION

```
To interrupt DesignSync commands, press Control-c. Not all DesignSync
commands can be interrupted; you can only interrupt commands that
perform multiple operations and do not complete immediately. These
commands include cancel, ci, co, import, ls, populate, retire,
setvault, setmirror, tag, and unlock.

When a command is run on multiple files or directories, the command
operates on multiple files or directories at once, as opposed to
processing one file at time, one directory at a time. When a command
is interrupted, before the command stops it will complete its
processing of the current files or directories being operated on.
```

SYNOPSIS

```
Control-c
```

EXAMPLES

```
The following is an example of interrupting a recursive checkin:
  dss> ci -recursive -nocomment -force Sportster
  Beginning Check in operation...

  Checking in: Sportster/code/samp.asm
  Checking in: Sportster/code/samp.lst
  Checking in: Sportster/code/samp.mem
  Checking in: Sportster/code/samp.s19

  Interrupt detected!

  Checking in: Sportster/code/sample1.asm

  Command Interrupted!

  dss>
```

Using Revision Control Keywords

keywords

NAME

```
keywords            - How to use revision control keywords
```

DESCRIPTION

- Module Note (Module-based)

```
The revision control engine used by DesignSync is RCE.
DesignSync supports RCE revision control keywords (or
keys) in your design files. By including keywords in your files,
you can access revision information (such as revision number,
author, and comment log), which is stored in the RCE archive
that underlies your vault.

You use revision control keywords by inserting them in your files,
typically within comments to keep programs that operate on your
files from interpreting the keywords. Keywords are delimited by
preceding and trailing dollar signs ($). There cannot be a space
between a keyword and its dollar-sign delimiters. Keywords are
expanded (keyword substitution) based on the version of the file
that you are checking out. Because the keyword expansion usually
changes the length of the file, keywords should only be used with
files whose format is position independent, such as text files.
You can control keyword expansion using the -keys option to the
ci, co, and populate commands.

Note: Revision control keyword expansion is not supported for:
      o files belonging to collections
      o the initial checkin of module data from "mkmod -checkin"

The following are the revision control keywords. The keywords are
case sensitive.

$Aliases$  List of tag names assigned to the revision
$Author$   Who checked the revision in
$Date$     The date and time the revision was checked in
           For modules and module member objects, the time is
           displayed in GMT. For DesignSync objects, the time is
           displayed in the server's local time.
$Header$   Concatenation of Source, Revision, Date, Author, and Locker
$Id$       Concatenation of RCSfile, Revision, Date, Author, and Locker
$KeysEnd$  Not expanded, and stops further expansion of keys
$Locker$   Who has locked this revision. If the revision is not
           locked, this value is empty (null).
$Log$      The full name of the archive file ($Source$) followed by
           the comment log (see Notes)
$RCSfile$  The name of the archive file, without the path
$Revision$ The revision number. For modules, the version provided
           is that of the module member.
$Source$   The full name of the archive file, including the path

Notes:
- The $Log$ keyword, when expanded, permanently adds log
  information to your file -- later collapsing the keyword
  leaves the log information in your file. Existing
  log messages are not replaced. Instead, the new log information
  is inserted each time the keyword is expanded. $Log$ is useful
  for accumulating a complete change log in a source file, but can
  result in differences or conflicts when doing merges or
  file comparisons. $Log$ is the only keyword with this behavior.
```

```
- When a keyword expansion requires spans multiple lines,
  the comment delimiter at the beginning of the line is repeated on
  subsequent lines. Therefore, make sure that the resulting syntax
  is valid. For example, in a C file, do not specify:

    /* $Log$
    */

  The resulting expansion has invalid comment syntax:

    /* $Log: /home/syncmgr/syncdata/adams/2647/Projects/Test/test.c.rca $
    /*
    /*  Revision: 1.6 Wed Jun 20 09:12:07 2001 Adams
    /*  *** empty comment string *** */

  Whereas if you specify:

    /*
    * $Log$
    */

  The resulting expansion is valid:

    /*
    * $Log: /home/syncmgr/syncdata/adams/2647/Projects/Test/test.c.rca $
    *
    *  Revision: 1.6 Wed Jun 20 09:12:07 2001 Adams
    *  *** empty comment string ***
    */
```

Module Note (Module-based)

```
For module members, the $RCSfile$ keyword expands to show the full
natural path of the module member.

When module members are locked, the locker keyword is not updated and
remains empty (null).  To view the locker information in that case,
use the ls command.
```

SEE ALSO

```
co, ci, populate
```

EXAMPLES

```
The following is an example of keywords in a file before and after
expansion. Note that the keywords appear within comment
delimiters, in this case  /* and */, such as a C file.
```

```
 /*
 * $Aliases$
 * $Author$
 * $Date$
 * $Header$
 * $Id$
 * $Locker$
 * $Log$
 *
 *
 * $RCSfile$
 * $Revision$
 * $Source$
 * $KeysEnd$
 * $Id$
 */


 Following substitution:

 /*
 * $Aliases:  Key-Example  $
 * $Author:  ja $
 * $Date:  Wed Jun 20 11:47:20 2001  $
 * $Header:  /home/syncmgr/syncdata/adams/2647/Projects/Test/test.c.rca  1.1
Wed Jun  20   11:47:20 2001 ja Stable  $
 * $Id:  test.c.rca 1.1 Wed Jun 20   11:47:20 2001 ja Stable  $
 * $Locker: $
 * $Log:  /home/syncmgr/syncdata/adams/2647/Projects/Test/test.c.rca $
 *
 *   Revision: 1.1 Wed Jun  20   11:47:20 2001 ja
 *   Initial revision
 *
 * $RCSfile:  test.c.rca $
 * $Revision:  1.1 $
 * $Source:  /home/syncmgr/syncdata/adams/2647/Projects/Test/test.c.rca $
 * $KeysEnd$
 * $Id$
 */
```

Using Selectors

selectors

NAME

```
 selectors           - How to use selectors and selector lists
```

DESCRIPTION

- What Are Selectors?

- Static Selectors Versus Dynamic Selectors
- How Does DesignSync Resolve Branch and Version Selectors?
- What are Selector Lists and Persistant Selector Lists (Module-based)
- What Are Selector Lists? (File-based)
- What Are Persistent Selector Lists? (File-based)
- Selector Formats
- File-Based Specific Formats (File-based)
- Date Formats


```
DesignSync uses selectors and selector lists to determine the
branch or version of an object to use for revision-control
operations. Understanding and properly using selectors and
selector lists is critical in multi-branch environments, but is
also important when using a single branch.

Note:
DesignSync supports two distinct features: 'select lists'
and 'selector lists'. 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.
```

What Are Selectors?

```
A selector is an expression that identifies a branch and version of
a managed object. For example, the version selector 'gold', the branch
selector 'Rel2:Latest', the version number '1.4', and the reserved
keyword 'Latest' are all selectors.
```

Static Selectors Versus Dynamic Selectors

```
Static selectors denote a set of objects whose contents are fixed.
These fixed objects might constitute a group of objects being
prepared for release. Static selectors include version selectors such
as 'gold' and branch selectors with fixed versions, such as
'Rel2:gold'. The objects denoted by a static selector do not change
with subsequent checkins.

Dynamic selectors denote a set of objects whose contents are
not fixed.  A branch selector such as 'Rel2:Latest' is a dynamic
selector because the objects denoted by the selector change;
a new 'Latest' version is created on the Rel2 branch with each
subsequent checkin.
```

How Does DesignSync Resolve Branch and Version Selectors?

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.

If the selector identifies a branch, DesignSync resolves the selector to both that branch and the Latest version on that branch. If branch 1.2.4 has branch tag 'Rel2', DesignSync resolves 'Rel2:Latest' as both branch 1.2.4 and the Latest version on that branch (say, 1.2.4.5). This behavior is important because some commands (such as 'co') operate on a version, some (such as 'ci') operate on a branch, and others (such as 'tag') operate on either a version or branch. If the tag cannot be resolved as a branch, DesignSync searches for a version of the same name, determines which branch the version is on, and resolves to the Latest version on that branch. For example, suppose an object, 'netlist.txt', has a version tagged 'beta' on its 1.2.4 branch. If the selector is 'beta', DesignSync first searches for a 'beta' branch. Finding no beta branch, DesignSync searches for a 'beta' version. DesignSync finds the 'beta' version, determines its branch, 1.2.4, and resolves to the Latest version on the 1.2.4 branch.

A selector can also specify both a branch and a version, for example, 'Rel2:gold'. This selector resolves if there is a branch 'Rel2' and if a version tagged 'gold' exists on the 'Rel2' branch.

A selector might not match any branch or version of a given object. For example, a file may not have a branch or version tagged 'Gold'. Because selectors can fail, it is common to specify selector lists.

Note: To resolve a selector, DesignSync does not search above the first folder that has a vault association. 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.

What are Selector Lists and Persistant Selector Lists (Module-based)

Selector lists automatically combine to create a blended workspace containing elements from the specified selectors. The selector list, a comma-separated list of selectors. No whitespace between items is allowed.
Examples of selector lists are:

```
gold,silver,bronze,Trunk:Latest
auto(Test),Main:Latest
Dev2.1:Latest,Rel2.1:Latest,Trunk
gold, 1.2, 1.2.1:Latest, Trunk
```

The final item in the selector list, is the main selector. When used, the command using the selector list creates a module manifest from the main selector, and layers that with the contents of the other selectors, in reverse order, processing the first selector in the command last. 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 include in the workspace.

A workspace created from the selector list can be checked in as a module member tagged workspace (aka: "snapshot"), or checked into the main selector, if the main selector is dynamic. If the main selector is a static selector, changes to the workspace can only be checked into as a snapshot.

When a workspace is populated with a selector list during the initial populate, or when the selector list is specified with the -version tag, the selector list is set as the persistent selector for the workspace and will be used for all subsequent populate operations.

Note: When using a selector list with -recursive, the operation recurses through a directory hierarchy according to the command it is used with, however it does not recurse through a static href, or a hierarchical reference to a legacy module, external module, or file based vault. The selector list is silently ignored when applied recursively to these sub-module types and the populate only considers the main selector.

What Are Selector Lists? (File-based)

You can combine selectors into a selector list, a comma-separated list of selectors. No whitespace between items is allowed.

```
Examples of selector lists are:
  gold,silver,bronze,Trunk:Latest
  auto(Test),Main:Latest
  Dev2.1:Latest,Rel2.1:Latest,Trunk
  gold, 1.2, 1.2.1:Latest, Trunk

Selector lists are used by commands that fetch objects ('co',
'populate', and 'import'). They provide a search order for
identifying and retrieving versions. DesignSync operates on each
element of a selector list in the same way it operates on an
individual selector. If the first selector in the list does not
resolve to a version, then DesignSync looks at the next selector in
the list. The first matching version is used. The command fails if
none of the selectors in the list resolves to a version. In the
case of tags, DesignSync first looks for a branch with the
specified tag, and if found, resolves to the Latest version on that
branch. Otherwise, DesignSync looks for a version with that tag.

For example, if you want to populate with "the most stable
configuration", you might define your selector list as
'Green,Yellow,Red,Trunk', where your team's development methodology
is to use version tags of 'Green', 'Yellow', and 'Red' to indicate
decreasing levels of stability. With this selector list, DesignSync
retrieves the first of the following versions it locates:
   1. The version tagged 'Green'.
   2. The version tagged 'Yellow'.
   3. The version tagged 'Red'.
   4. The Latest version on the branch tagged Trunk. Trunk
      (shorthand for Trunk:Latest) is the default tag name for
      branch 1.

If your team is using a branching methodology and you want to populate
with "the most stable configuration", you might define your selector
list instead as 'Green:Latest,Yellow:Latest,Red:Latest,Trunk'.
With this selector list, DesignSync retrieves the first of the following
versions it locates:
   1. The Latest version on the branch tagged 'Green'.
   2. The Latest version on the same branch as the version tagged
      'Green'.
   3. The Latest version on the branch tagged 'Yellow'.
   4. The Latest version on the same branch as the version tagged
      'Yellow'.
   5. The Latest version on the branch tagged 'Red'.
   6. The Latest version on the same branch as the version tagged
      'Red'.
   7. The Latest version on the branch tagged Trunk.

Selector lists are a powerful mechanism, but can also add
complexity. To avoid accidental checkins to unintended branches
when you have complicated selector lists, the 'ci' and 'co -lock'
commands consider only the first selector in a selector list. If
the first selector resolves to a branch, the operation
continues, otherwise the operation fails.

Note:  Using CONFIG statements in sync_project.txt files, you
       can map a configuration to a single selector or a
       selector list. See the DesignSync help topic "Using
```

```
       Vault References for Design Reuse" for information.
```

What Are Persistent Selector Lists? (File-based)

```
Some commands (ci, co, populate, import) support persistent
selector lists. Persistent selector lists specify what branch or
version a command operates on in the absence of an explicit
-version or -branch option. Commands that do not obey the persistent
selector list typically operate on the current version or current
branch of the object in your work area.

You explicitly set the persistent selector list, typically on an
entire work area, using the 'setselector' command. You can also set
the selector at the same time you set the vault with the 'setvault'
command. DesignSync also sets the persistent selector list in the
following cases:
  o When populating a configuration-mapped folder. This behavior is
    a performance optimization. See the "populate" help topic for
    details.
  o When populating or checking out with the -overlay option and an
    object exists on the overlay branch but not on the branch being
    overlaid (the work area branch). DesignSync may augment the
    object's persistent selector list with the Auto() selector so
    that the object is automatically branched when checked in. See
    the -overlay option description for details.

A persistent selector list is stored in an object's local
metadata, or it is inherited from its parent folder. If a
persistent selector list has not been defined, the default is
'Trunk'.

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). The persistent selector list is
not updated when you use the -version or -branch option.

The 'P' data key for the 'ls' command and the 'url selector'
command report an object's persistent selector list.

Note:  Using CONFIG statements in sync_project.txt files, you
       can map a configuration to a single selector or a
       selector list. See the DesignSync help topic "Using
       Vault References for Design Reuse" for information.
```

Selector Formats

```
A selector can have one of several formats:

o <number>

  A branch or version number. Branch and version numbers are also
  known as "dot numerics". Using branch or version numbers as
```

selectors is typically less convenient than using tags or date-based selectors.

A version <number> selector is a static selector; the objects denoted by the version <number> selector are fixed. A branch <number> selector is a dynamic selector; the objects denoted by the branch <number> selector change upon subsequent checkins.

Examples: 1.1, 1.3.2.3 -- version <number> selectors
1, 1.3.4, 1.1.1 -- branch <number> selectors

o <versiontag>

A version selector. If you specify a version selector, DesignSync resolves the selector to both the object's version number and branch number. For more details, see "How Does DesignSync Resolve Branch and Version Selectors?"

A <versiontag> selector is a static selector; the objects denoted by the <versiontag> selector are fixed.

A given tag name might be applied to a branch or to a version (but never both at the same time for the same object). Branch selectors use the syntax '<branchtag>:<versiontag>', for example, 'Rel2:Latest' to differentiate them from version selectors.

If you specify a version selector during the check-in of a new object, the object is created, by default, on the Trunk branch. If you instead intend to check the object into a different branch, be sure to specify a branch selector rather than a version selector.

Examples of version selectors: gold
alpha

o <branchtag>:Latest

A branch selector that specifies the most recent version on the branch. A given tag name might be applied to a branch or to a version (but never both at the same time for the same object). To specify a branch selector, append ':<versiontag>', in this case, ':Latest' to the branch tag name, for example, 'Rel2.1:Latest'. You can leave off the 'Latest' keyword as shorthand. For example, 'Rel2.1:' is equivalent to 'Rel2.1:Latest'.

A <branchtag>:Latest (or <branchtag>:) selector is a dynamic selector; the objects denoted by this selector change upon subsequent checkins.

If <branchtag> cannot be resolved as a branch tag, DesignSync searches for a version tag of that name and resolves to the Latest version on that version's branch. For more details, see "How Does DesignSync Resolve Branch and Version Selectors?"

The tag 'Trunk' (shorthand for 'Trunk:Latest') has special

```
   significance; it is the default tag name for branch 1.

   Examples of branch selectors: Trunk          -- branch 1 default
                                                   (shorthand for
                                                   Trunk:Latest)
                                 Rel2.1:Latest  -- branch selector of
                                                   most recent Rel2.1
                                                   version
                                 Rel2.1:        -- shorthand for
                                                   Rel2.1:Latest

   Notes about using Latest and Date():

   - <branchtag>:Date(<a_date_in_the_future>)is equivalent to
     <branchtag>:Latest. Meaning that as long as the specified date is
     in the future, the <branchtag>:Date selector will resolve to the
     <branchtag>:Latest.  Once the date has been reached, however;
     the last checked in version for the date becomes the version
     indicated for that selector.
   - When used with the 'setselector' command or as part of a
     selector list argument (more than one selector) to
     -version, Latest and Date() must be qualified with a branch:
          <branchtag>:Latest, <branchtag>:Date(<date>)
   - When used as the only selector to a -version option,
     DesignSync augments the selector with the persistent selector
     list. For example, if the persistent selector list is
     'Gold:,Trunk' and you specify 'co -version Latest', the
     selector list used for the operation is
     'Gold:Latest,Trunk:Latest'; the persistent selector list
     remains 'Gold:,Trunk' after the operation.

     Exception: When you check in an object whose branch you have
                locked (having done a 'co' or 'populate' with the
                -lock option), the date selector augments the
                current branch, not the persistent selector
                list. You typically want to remain working on the
                locked branch even if the persistent selector list
                has changed.

   These restrictions are required to avoid ambiguity about which
   branch a Latest or Date() selector applies to.

 o <branchtag>:<versiontag>

   A specific version on a specific branch. The <branchtag> and
   <versiontag> values are themselves selectors.

   Unlike the dynamic <branchtag>:Latest selector, a
   <branchtag>:<versiontag> selector is a static selector; the
   objects denoted by the <branchtag>:<versiontag> selector
   are fixed.

   Examples: Rel2:alpha
             Trunk:Date(yesterday)

   Notes:
    - To specify a specific branch and version, the selector must
```

```
      contain both the branch and version. ':<versiontag>' is illegal.
      '<branchtag>:' resolves to the Latest version on the specified
      branch.
    - A selector such as 'Trunk:gold' is valid and indicates a version
      tagged 'gold' only if it is on a branch called 'Trunk'; otherwise,
      the selector fails.
    - A selector of the form 'Gold:Latest' looks for a branch
      tagged Gold, and if found, fetches the Latest version on that
      branch. If a 'Gold' branch is not found, DesignSync looks for a
      version tagged 'Gold', and if found, retrieves the Latest
      version on the branch that the 'Gold' version is on. Selectors
      of the form 'Gold:Date(<date>)' behave similarly.
    - For backward compatibility, DesignSync supports selectors of
      the form '<version_number>:Latest' and
      <version_number>:Date(<date>). DesignSync uses the branch of
      the specified version, and then applies the Latest or Date()
      selector. For example, 1.2:Latest resolves to 1:Latest, and
      1.3.2.1:Date(yesterday) resolves to 1.3.2:Date(yesterday).

o <branchtag>:Date(<date>)

  The most recent version on the specified branch that was
  created on or before the specified date. The 'Date' keyword
  is case insensitive. Note that if you specify a date in the future,
  the selector is equivalent to using the the 'Latest' version
  selector until the date is reached.

  A <branchtag>:Date(<date>) selector is a static selector;
  the objects denoted by this selector date are fixed provided the
  specified date has been reached. If the date is in the future, the
  selector is dynamic until the date is reached.

  For example, <branchtag>:Date(yesterday) will always resolve to
  yesterday's last checked in version.  But
  <branchtag:Date(<futureDate>) will match <branchtag>:Latest until
  the date specified has been reached, at which point the selector
  will resolve to the last checked in version before the date.

  You specify the Date selector as follows:
     <branchtag>:Date(<date>)
  where <branchtag> is a branch tag. If <branchtag> cannot be
  resolved as a branch tag, DesignSync searches for a version tag
  of that name and resolves to the most recent version created on
  or before the specified date on that version's branch.

  Examples: Trunk:Date(yesterday)  -- Resolves to the last version
                                      checked in yesterday on the
                                      Trunk branch
            gold:date(4/11/00)     -- If no branch is named 'gold',
                                       but there is a version selector
                                       'gold', DesignSync resolves
                                      this selector to the last version
                                      checked in on or before 4/11/00
                                      on the branch containing the
                                      'gold' version
            Rel2:Date(today)       -- Resolves to the last version
                                      checked in today on the Rel2
```

branch

See the "Date Formats" section below for details on how to specify dates. See <branchtag>:Latest for more information about specifying date selectors.

o VaultDate(<date>)

The most recent version on any branch that was created on or before the specified date. The 'VaultDate' keyword is case insensitive. Like the Date selector, the VaultDate specification can accept a branch tag, in the format:

<branchtag>:VaultDate(<date>)

where <branchtag>: is optional.

A VaultDate(<date>) selector is a dynamic selector; the objects denoted by this selector are dependent on the date.

Examples: VaultDate(yesterday)
VaultDate(4/11/00)
Rel40:VaultDate(today)

See the "Date Formats" section for details on how to specify dates.

File-Based Specific Formats (File-based)

o Auto(<tag>)

Used for auto-branching, which creates branches on an as-needed basis as opposed to branching an entire project. This methodology is useful for "what if" scenarios. In general, Auto(<tag>) is equivalent to just <tag>. The 'Auto' is significant only for operations that can create branches (ci, co -lock, populate -lock). An Auto(<tag>) selector is a dynamic selector.

The Auto keyword is case insensitive.

Examples: Auto(Dev)
auto(Rel2.1_p1)

Notes:
- The value supplied to the auto-branch selector must be a branch or version name, not a branch selector. 'Auto(Golden:)' and 'Auto(Golden:Latest)' are illegal selectors.
- The current version is always the branch-point version when Auto() creates a new branch.
- If present, Auto(<tag>) must be the first selector in a selector list.
- The auto(<branch>) option is only applicable for legacy modules and DesignSync objects. It is not a valid selector for modules.

Date Formats

This section describes how you can specify dates when using Date(<date>) and VaultDate(<date>) selectors. DesignSync uses a public-domain date parser that supports a wide range of date and time specifications. The parser is the same one used by the Gnu family of tools. Visit a Gnu website for a complete specification. This section documents the more common formats.

Note: If the date specification contains spaces, you must surround the entire selector list with double quotes.
For example: 'Gold:Date(last Tuesday),Trunk'

Year Format:
You can specify the year using 2 or 4 digits. DesignSync interprets 2-digit year specifications between 00 and 69, inclusive, as 2000 to 2069, and specifications between 70 and 99, inclusive, as 1970 to 1999.

If you omit the year, the default is the current year.

Month Format:
You can specify the month as a number (1 through 12), or using the following names and abbreviations:

January	Jan	Jan.		
February	Feb	Feb.		
March	Mar	Mar.		
April	Apr	Apr.		
May	May	May.		
June	Jun	Jun.		
July	Jul	Jul.		
August	Aug	Aug.		
September	Sep	Sep.	Sept	Sept.
October	Oct	Oct.		
November	Nov	Nov.		
December	Dec	Dec.		

Note: September is the only month for which a 4-letter abbreviation is valid.

If you omit the month, the default is the current month.

Day Format:
You can specify days of the week in full or with abbreviations:

Sunday	Sun	Sun.		
Monday	Mon	Mon.		
Tuesday	Tue	Tue.	Tues	Tues.
Wednesday	Wed	Wed.	Wednes	Wednes.
Thursday	Thu	Thu.	Thurs	Thurs.
Friday	Fri	Fri.		
Saturday	Sat	Sat.		

```
You can add words such as 'last' or 'next' before a day of the week to
specify a date other than the nearest day of the same name. For
example:

  Thursday        Specifies the most recent past Thursday, or
                  today, if today is Thursday.

  next Thursday   Specifies one week after the most recent Thursday
                  (includes the current day if today is Thursday).

  last Thursday   Specifies one week before the most recent Thursday
                  (includes the current day if today is Thursday).

If you omit the day, the default is the current day.

Note: A comma after a day of the week item is ignored.

Time Format:
You specify the time of the day as hour:minute:second, where hour
is a number between 0 and 23, minute is a number between 0 and 59,
and second is a number between 0 and 59.

Any portion not specified defaults to '0', so a date specification
of 03/04/00 defaults to a time of 00:00:00, which is the start of
the day (end of the previous day).
```

SEE ALSO

```
select, setselector, setvault, unselect, url selector
```

EXAMPLES

```
All of the following examples specify the same calendar date:

  Note: The preferred order in the U.S. may be ambiguous compared to
  other countries usage of DD-MM-YY if the number of either the month
  or the day is less than 10. For example, a date such as 9/12/00
  means September 12, 2000 in the U.S. but December 9, 2000 in many
  other countries.

  2000-09-24          # ISO 8601.
  00-09-24            # 00 indicates year 2000.
  00-9-24             # Leading zeros are not required. For
                        example, '9' is equivalent to  '09'.
  09/24/00            # U.S. preferred order. See previous note.
  24-sep-00           # Three-letter month abbreviations
                        are allowed.
  24sep00             # Hyphen and slashes are not required
                        delimiters.
  23 sep 00           # Spaces are permitted, but the entire
                        selector list must be placed within double
```

```
                        quotes.

  Note: The following three times represent local time, because no
  time zone is specified.

  20:02:0              # 2 minutes after 20 (8 PM)
  20:02                # 2 minutes  after 20 (8 PM), zero seconds implied
  8:02pm               # 2 minutes after 8 PM

  Note: The following time applies to Eastern U.S. Standard
  Time because the -0500 means 5 hours behind UTC (Coordinated
  Universal Time, also known as Greenwich Mean Time) time.
  The -0500 is a 'time zone item'.

  20:02-0500           # 2 minutes after 8 PM Eastern U.S. Time

  Note: You can combine a time and a date as follows:

  01/24/00 20:02       # 2 minutes after 8 PM on January 24th 2000.
                         Contains spaces, so the entire selector
                         list must be placed within double quotes.
```

Using Wildcards

wildcard

NAME

```
wildcard            - Using wildcard characters in filenames
```

DESCRIPTION

```
Wildcards are useful when you want to specify a large
number of files without having to specify the name of
each file. The following wildcards can be used in the
filename portion of pathnames and URLs:

 ?        Matches any single character
 [list]   Matches any single character present in list
 *        Matches any pattern

Note: In stcl/stclc mode, you need to use a backslash (\) to escape
the left bracket ([).

Depending on where wildcards are specified, they are either
immediately resolved (static list) or are resolved at the time
of an operation (dynamic list).

Exclude lists (-exclude option) are dynamic; wildcards are stored
as the actual wildcard characters. When an operation is to be
performed on files or directories, each file or directory is
```

```
checked against the exclude list. If any object matches one of
the names in the exclude list, that object is not processed. In
addition to specifying -exclude on a per-operation basis, you can
also define exclude lists, which apply to all revision-control
operations that accept the -exclude option, from the DesignSync
graphical interface (Tools->Options->General->Exclude Lists).

Select lists (-selected option) are static; the select list
resolves all wildcards to the exact names of the files or
directories at the time the select list is created.
This static select list can then be used by any
revision-control operation that accepts the -selected option. Note
that when wildcards are used in a list of objects to be processed by
the unselect command, the wildcards are matched against objects in
the select list, not against objects in the current directory.

Note: If you use a wildcard with a -recursive option of a DesignSync
command, DesignSync first matches that wildcard against the contents
of the starting directory, and then processes any matching objects
and recurses through any matching directories. DesignSync applies
the wildcard only at the starting point, and not throughout the
directory hierarchy.
```

EXAMPLES

```
This example lists any object in the directory mydir that begins
with the 'a', '3', or '#' character, and ends with
a '.' followed by any three characters.

  dss> ls /mydir/[a3#]*.???

In stcl, you need to escape the left bracket:
  stcl> ls /mydir/\[a3#]*.???

Example of wildcards in exclude lists:

CVS          Ignore any files or directories named "CVS".
foo.obj      Ignore any files or directories named "foo.obj".
src/foo.obj  Ignore any files named "foo.obj" in a directory
             named "src".
src/foo*     Ignore any files or directories beginning
             with "foo" in a directory named "src".
```

Client Applications

DesSync

DesSync Command

NAME

```
DesSync             - Invokes the DesignSync graphical interface
```

DESCRIPTION

```
This command invokes the DesignSync graphical user interface
(GUI). You execute DesSync from your operating system shell, not
from a Synchronicity client shell (dss/dssc/stcl/stclc).  On
Windows platforms, you can also invoke DesSync from the Windows
Start menu, typically:
  Start->Programs->Dassault Systems <VersionNumber>->DesignSync

You can also execute all DesignSync command-line commands from
the GUI.

DesignSync Data Manager User's Guide describes the GUI. From the GUI,
select Help->Help Topics, or click the book icon in the Tool Bar.
```

SYNOPSIS

```
DesSync [-nosplash] [path]
```

OPTIONS

- -nosplash
- -path

-nosplash

```
-nosplash     Prevents the DesignSync splash screen from
              displaying at startup.

              You can also disable the splash screen site-wide or
              by user:

              Site Wide
```

```
---------
Change the last line of SYNC_DIR/bin/DesSync to the include
the flag:

exec .runjava -jar $SYNC_DIR/classes/dsj.jar -nosplash $*:q

User
----
Create a script with the following contents, and put it in the
path ahead of SYNC_DIR/bin:

#!/bin/csh -f
exec ${SYNC_DIR}/bin/DesSync -nosplash $*:q

Note: If you use spaces in your path arguments to the command,
you must places them in quotes.
```

-path

```
path          The path to the directory/folder that you want
              DesignSync to expand at startup. You can specify an
              absolute or relative path. You can also set the
              DesignSync initial folder through a SyncAdmin option
              (GUI Options->Initial Folder).
```

RETURN VALUE

```
none
```

SEE ALSO

```
dss, dssc, stcl, stclc, SyncAdmin
```

EXAMPLES

- Example of Starting the DesignSync GUI
- Example of Starting the DesignSync GUI without the Splash Screen
- Example of Starting the DesignSync GUI Opened to the Current Directory
- Example of Starting the DesignSync Opened to Specified Path

Example of Starting the DesignSync GUI

```
This example invokes the DesignSync GUI:
  % DesSync
```

Example of Starting the DesignSync GUI without the Splash Screen

```
This example invokes the GUI in background mode without displaying
the splash screen:
  % DesSync -nosplash &
```

Example of Starting the DesignSync GUI Opened to the Current Directory

```
This example invokes the GUI and specifies the current directory
as the initial folder:
  % DesSync .
```

Example of Starting the DesignSync Opened to Specified Path

```
This example invokes the GUI and specifies a specific project
directory as the initial folder:
  % DesSync /home/projects/chip/alu
```

dss

dss Command

NAME

```
dss                 - Invokes the DesignSync shell (dss)
```

DESCRIPTION

```
This command invokes the DesignSync shell (dss), a DesignSync
command-line interface. dss is one of the DesignSync client
applications (along with dssc, stcl, stclc, and the DesSync
graphical interface).

You invoke dss from your operating-system (OS) shell in one of
the following ways:

o Specify no arguments to the dss command to enter the dss
  environment, indicated by the 'dss>' prompt. You remain in
  the dss shell until you issue the 'exit' command, which
  returns you to your OS shell.

o Specify a DesignSync command to execute. DesignSync executes the
  command, then returns you to your OS shell.
```

```
On Windows platforms, you can also invoke dss from the Windows
Start menu, typically:
  Start->Programs->Dassault Systems <Version>
       ->DesignSync Shell (dss)

The dss and stcl clients communicate with a Synchronicity server
(SyncServer) through syncd, the Synchronicity daemon. If you do
not already have a syncd process running, dss attempts to start
one.

The syncd process can manage multiple dss/stcl requests per user,
allowing one user to run parallel dss/stcl sessions. However,
syncd handles requests serially, which can cause operations from
one dss session to be blocked while operations from another session
execute. It is therefore recommended that you use dssc, the
concurrent version of dss. The dssc and stclc clients do not use
syncd; they communicate directly with a SyncServer. The dssc
and stclc clients also have the advantage of supporting more
robust command-line editing, including command and filename
completion. The dss client does support DesignSync command
abbreviations, like the dssc and stclc clients.

The only advantage of dss over dssc is that dss start-up time when
a syncd is already running is less than dssc start-up time. If you
frequently run DesignSync commands from your OS shell using the
form "dssc <command>" instead of staying in the dssc shell, use dss
instead of dssc.

If you want to use programming constructs (such as variables,
loops, and conditionals) in conjunction with DesignSync commands,
use stcl or stclc, the Synchronicity Tcl clients.

Note: Both dss and stcl inherit their environments, such as
      environment variable definitions, from syncd. Therefore,
      you must stop and restart syncd (see the syncdadmin command)
      for your dss and stcl sessions to pick up environment changes.
```

SYNOPSIS

```
dss [<command>]
```

OPTIONS

```
none
```

RETURN VALUE

```
Returns the success status of the last executed command: zero (0)
```

```
indicates success, non-zero indicates failure. You can specify a
return value as an argument to the exit command, which is typically
the last command executed.
```

SEE ALSO

```
dssc, stcl, stclc, DesSync, syncdadmin, exit
```

EXAMPLES

- Example of Invoking DSS (DesignSync Shell)
- Example of Running a DSS Command From OS Shell

Example of Invoking DSS (DesignSync Shell)

```
This example invokes the DesignSync shell, from which you can
enter any number of dss commands. Use 'exit' to return to your
OS shell.
  % dss
  Logging to /home/tgoss/dss_01192000_161324.log
  V3.0

  dss> spwd
  file:///home/tgoss/Projects/Sportster/code
  dss> exit
  %
```

Example of Running a DSS Command From OS Shell

```
This example executes the 'co' command while remaining in your
OS shell.

  % dss co -nocomment -lock samp.mem
  Logging to /home/tgoss/dss_01192000_160958.log
  V3.0

  Beginning Check out operation...

  Checking out: samp.mem  : Success - Checked Out  version: 1.1 -> 1.2

  Checkout command finished...
  %
```

dssc

dssc Command

NAME

```
dssc                - Invokes the concurrent version of dss
```

DESCRIPTION

```
This command invokes the concurrent version of the DesignSync shell
(dssc), a DesignSync command-line interface. dssc is one of the
DesignSync client applications (along with dss, stcl, stclc, and
the DesSync graphical interface).

You invoke dssc from your operating-system (OS) shell in one of
the following ways:

o Specify no arguments to the dssc command to enter the dssc
  environment, indicated by the 'dss>' prompt. You remain in the
  dssc shell until you issue the 'exit' command, which returns
  you to your OS shell.

o Specify a DesignSync command to execute. DesignSync executes the
  command, then returns you to your OS shell.

  Note: The UNIX shell does not pass a null string ("") to the
  DesignSync client when dssc is specified with a DesignSync command
  argument.  If your command requires a null string, use the stcl
  or stclc client with the -exp option.

On Windows platforms, you can also invoke dssc from the Windows Start
menu, typically:
  Start->Programs->Dassault Systems <version>
       -> DesignSync Concurrent Shell (dssc)

The dssc and stclc clients communicate directly with a
Synchronicity server (SyncServer). Unlike dss and stcl, no
Synchronicity daemon process (syncd) is used. Because syncd handles
requests serially, using dssc eliminates a potential bottleneck
when you have multiple shells communicating with a SyncServer. The
one advantage dss has over dssc is that dss start-up time when a
syncd is already running is less than dssc start-up time. If you
frequently run DesignSync commands from your OS shell using the
form "dssc <command>" instead of staying in the dssc shell, use dss
instead of dssc.

The dssc shell supports command-line editing:

Behavior               Control Keys     Special Keys
--------               ------------     ------------
Forward one character  Ctrl-f           Right arrow
Back one character     Ctrl-b           Left arrow
Beginning of line      Ctrl-a           Home (only supported
                                        for Windows platforms)
End of line            Ctrl-e           End (only supported
                                        for Windows platforms)
```

```
Kill rest of line       Ctrl-k
Kill line               Ctrl-u          Esc
                                        (Note: The Esc key
                                        instead invokes vi
                                        command mode if your
                                        <EDITOR> environment
                                        variable is set to vi
                                        or your ~/.inputrc file
                                        contains the line
                                        'set editing-mode vi'.
                                        Note also that the
                                        DesignSync GUI default
                                        editor setting does not
                                        affect the behavior
                                        of the Esc key.)
Delete character        Ctrl-d          Delete
Previous command from
command history         Ctrl-p          Up arrow
Next command from
command history         Ctrl-n          Down arrow
Exit stclc/dssc shell   Ctrl-d
```

The dssc shell also supports command, option, and filename completion, as well as DesignSync command abbreviations. See DesignSync Data Manager User's Guide for details.

If you want to use programming constructs (such as variables, loops, and conditionals) in conjunction with DesignSync commands, use stcl or stclc, the Synchronicity Tcl clients.

SYNOPSIS

```
dssc [<command>]
```

OPTIONS

none

RETURN VALUE

Returns the success status of the last executed command: zero (0) indicates success, non-zero indicates failure. You can specify a return value as an argument to the exit command, which is typically the last command executed.

SEE ALSO

```
dss, stcl, stclc, DesSync, exit
```

EXAMPLES

- Example of Invoking DSSC (DesignSync Concurrent Shell)
- Example of Running a DSSC Command From OS Shell

Example of Invoking DSSC (DesignSync Concurrent Shell)

```
This example invokes the DesignSync shell, from which you can
enter any number of dss commands. Use 'exit' to return to your
OS shell.
  % dssc
  Logging to /home/tgoss/dss_01192000_161324.log
  V3.0

  dss> spwd
  file:///home/tgoss/Projects/Sportster/code
  dss> exit
  %
```

Example of Running a DSSC Command From OS Shell

```
This example executes the 'co' command while remaining in your
OS shell.
  % dssc co -nocomment -lock samp.mem
  Logging to /home/tgoss/dss_01192000_160958.log
  V3.0

  Beginning Check out operation...

  Checking out: samp.mem    : Success - Checked Out  version: 1.1 -> 1.2

  Checkout command finished...
  %
```

stcl

stcl Command

NAME

```
stcl                - Invokes the Synchronicity Tcl interface
```

DESCRIPTION

```
This command invokes the Synchronicity Tcl (stcl) shell. stcl is
one of the DesignSync client applications (along with dss, dssc,
stclc, and the DesSync graphical interface). All DesignSync
commands and commercial Tcl commands are available in the stcl
environment.  Note: To determine the version of Tcl included in
your Synchronicity installation's stcl interpreter, use the
Tcl 'info tclversion' and 'info patchlevel' commands within
an stcl/stclc client shell.

You invoke stcl from your operating-system (OS) shell in one of
the following ways:

 o Specify no arguments to the stcl command to enter the stcl
   shell, indicated by the 'stcl>' prompt. You remain in the stcl
   shell until you issue the 'exit' command, which returns you to
   your OS shell.

 o Specify a script name to execute a Tcl script, which is
   equivalent to sourcing a Tcl script from within the stcl
   environment. For example:
        % stcl myscript.tcl
        ...script executes
        %
   is equivalent to
        % stcl
        stcl> source myscript.tcl
        ...script executes...
        stcl> exit
        %

o Specify the -exp option to directly execute one or more
  DesignSync or Tcl commands. Use a semicolon (;) to
  separate multiple commands, and surround the entire command
  string with single quotes.

On Windows platforms, you can also invoke stcl from the Windows
Start menu, typically:
  Start->Programs-> Dassault Systems <version>->
       -> DesignSync Tcl Shell (stcl)

Use the stcl shell when you need the scripting constructs of Tcl,
such as conditionals (if/then/else), loops (while, for, foreach),
and variable assignment (set). If you do not need Tcl constructs,
dss provides a simpler command environment. With stcl:
  - You cannot abbreviate DesignSync commands. For example,
    you cannot abbreviate 'populate' to 'pop' as you might in
    dss/dssc/stclc.
  - You must use double quotes around objects that contain a
    semicolon (;), such as vaults, branches, and versions. The
    semicolon is the Tcl (and therefore stcl) command separator.
  - You must specify the -exp option to execute a DesignSync
    command from the OS shell. Because stcl is primarily a
    scripting shell, an argument specified without -exp is assumed
    to be a script. With dss, the syntax for specifying a single
    command is simpler.
```

```
For more information on Tcl, including documentation for Tcl
commands, visit these Web sites:

    http://www.tcl.tk
    http://tcl.sourceforge.net

The stcl and dss clients communicate with a Synchronicity server
(SyncServer) through syncd, the Synchronicity daemon. If you do
not already have a syncd process running, stcl attempts to start
one.

The syncd process can manage multiple dss/stcl requests per user,
allowing one user to run parallel dss/stcl sessions. However, syncd
handles requests serially, which can cause operations from one
dss/stcl session to be blocked while operations from another
session execute. It is therefore recommended that you use stclc,
the concurrent version of stcl. The stclc and dssc clients do not
use syncd; they communicate directly with a SyncServer. The stclc
client also has the advantage of supporting more robust command-line
editing, including command and filename completion, as well as
command history search and DesignSync command abbreviations.

One advantage of stcl over stclc is that stcl start-up time when a
syncd is already running is less than stclc start-up time. If you
are frequently running DesignSync commands from your OS shell
using the form "stclc -exp '<command>'" instead of staying in the
stclc shell, use stcl instead of stclc.

Note: Both stcl and dss inherit their environments, such as
      environment variable definitions, from syncd. Therefore,
      you must stop and restart syncd (see the syncdadmin command)
      for your stcl and dss sessions to pick up environment changes.
```

SYNOPSIS

```
stcl [--] [-exp '<command>' | <scriptname>]
```

OPTIONS

- -exp
- --

-exp

```
-exp '<command>' Executes one or more commands. Separate multiple
                 commands with a semicolon, and surround the command
                 expression with single quotes. Using double quotes
                 works, but single quotes facilitate including
                 double quotes within the command expression.
```

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when arguments
                to the command begin with a hyphen (-).
```

RETURN VALUE

```
Returns the success status of the last executed command: zero (0)
indicates success, non-zero indicates failure (an exception was
thrown). You can specify a return value as an argument to the exit
command, which is typically the last command executed.
```

SEE ALSO

```
stclc, dss, dssc, DesSync, syncdadmin, rstcl, exit
```

EXAMPLES

- Example of Invoking the stcl shell
- Example of Using stcl to Run a Script
- Example of Invoking Commands in stcl without Using the Shell
- Example of Invoking Commands that Include a Quoted String

Example of Invoking the stcl shell

```
This example invokes the stcl shell:
  % stcl
  stcl>
```

Example of Using stcl to Run a Script

```
This example invokes a Tcl script that populates a work area:
  % stcl update.tcl
```

Example of Invoking Commands in stcl without Using the Shell

```
This example executes the 'ls' command followed by the 'spwd' command:
  % stcl -exp 'ls;spwd'
```

Example of Invoking Commands that Include a Quoted String

```
This example shows a command string that includes double quotes:
  % stcl -exp 'ci -keep -comment "Fixed defect 4354" file.c'
```

stclc

stclc Command

NAME

```
stclc               - Invokes the concurrent version of stcl
```

DESCRIPTION

```
This command invokes the concurrent version of the Synchronicity
Tcl (stclc) shell. stclc is one of the DesignSync client
applications (along with dss, dssc, stcl, and the DesSync graphical
interface). All DesignSync commands and commercial Tcl
commands are available in the stcl environment. Note: To determine
the version of Tcl included in your Synchronicity installation's
stcl interpreter, use the Tcl 'info tclversion' and
'info patchlevel' commands within an stcl/stclc client shell.

You invoke stclc from your operating-system (OS) shell in one of
the following ways:

 o Specify no arguments to the stclc command to enter the stclc
   shell, indicated by the 'stcl>' prompt. You remain in the stclc
   shell until you issue the 'exit' command, which returns you to
   your OS shell.

 o Specify a script name to execute a Tcl script, which is
   equivalent to sourcing a Tcl script from within the stclc
   environment. For example:
        % stclc myscript.tcl
        ...script executes
        %
   is equivalent to
        % stclc
        stcl> source myscript.tcl
        ...script executes...
        stcl> exit
        %

o Specify the -exp option to directly execute one or more
  DesignSync or Tcl commands. Use a semicolon (;) to
  separate multiple commands, and surround the entire command
  string with single quotes.
```

```
On Windows platforms, you can also invoke stclc from the Windows Start
menu, typically:
  Start->Programs->Dassault Systems <version>
  ->DesignSync Concurrent Tcl Shell (stclc)

Use the stclc shell when you need the scripting constructs of Tcl,
such as conditionals (if/then/else), loops (while, for, foreach),
and variable assignment (set). If you do not need Tcl constructs,
dssc provides a simpler command environment. With stclc:
  - You must use double quotes around objects that contain a
    semicolon (;), such as vaults, branches, and versions. The
    semicolon is the Tcl (and therefore stclc) command separator.
  - You must specify the -exp option to execute a DesignSync
    command from the OS shell. Because stclc is primarily a
    scripting shell, an argument specified without -exp is assumed
    to be a script. With dssc, the syntax for specifying a single
    command is simpler.

For more information on Tcl, including documentation for Tcl
commands, visit these Web sites:

    http://www.tcl.tk
    http://tcl.sourceforge.net

The stclc and dssc clients communicate directly with a
Synchronicity server (SyncServer). Unlike dss and stcl, no
Synchronicity daemon process (syncd) is used. Because syncd handles
requests serially, using stclc eliminates a potential bottleneck
when you have multiple shells communicating with a SyncServer. The
one advantage stcl has over stclc is that stcl start-up time when a
syncd is already running is less than stclc start-up time. If you
are frequently running stcl commands from your OS shell
using the form "stclc -exp '<command>'" instead of staying in the
stclc shell, use stcl instead of stclc.

The stclc shell supports command-line editing:

Behavior                Control Keys     Special Keys
--------                ------------     ------------
Forward one character   Ctrl-f           Right arrow
Back one character      Ctrl-b           Left arrow
Beginning of line       Ctrl-a           Home (only supported
                                         for Windows platforms)
End of line             Ctrl-e           End (only supported
                                         for Windows platforms)
Kill rest of line       Ctrl-k
Kill line               Ctrl-u           Esc
                                         (Note: The Esc key
                                         instead invokes vi
                                         command mode if your
                                         <EDITOR> environment
                                         variable is set to vi
                                         or your ~/.inputrc file
                                         contains the line
                                         'set editing-mode vi'.
                                         Note also that the
```

```
                                        DesignSync GUI default
                                        editor setting does not
                                        affect the behavior
                                        of the Esc key.)
Delete character        Ctrl-d          Delete
Previous command from
command history         Ctrl-p          Up arrow
Next command from
command history         Ctrl-n          Down arrow
Exit stclc/dssc shell   Ctrl-d

The stclc shell also supports command, option, and filename
completion, as well as command history search and DesignSync
command abbreviations.  See DesignSync Data Manager User's Guide for
details.

The stclc shell also supports Unix commands; if you enter a
Unix command which is not also an stcl function, the
interpreter automatically calls the Unix command. In this way,
you can execute Unix programs without using the Tcl 'exec'
command.  Note: If you do not want Unix commands to be
executed automatically, you can set the global variable,
auto_noexec, to disable this behavior.  For details, refer
to Tcl documentation of the Tcl 'unknown' command.
```

SYNOPSIS

```
stclc [--] [-exp '<command>' | <scriptname>]
```

OPTIONS

- -exp
- --

-exp

```
-exp '<command>' Executes one or more commands. Separate multiple
                 commands with a semicolon, and surround the command
                 expression with single quotes. Using double quotes
                 works, but single quotes facilitate including
                 double quotes within the command expression.
```

--

```
--               Indicates that the command should stop looking for
                 command options. Use this option when arguments
                 to the command begin with a hyphen (-).
```

RETURN VALUE

```
Returns the success status of the last executed command: zero (0)
indicates success, non-zero indicates failure (an exception was
thrown). You can also specify a return value as an argument to the
exit command, which is typically the last command executed.

A failure results when an exception is thrown.
```

SEE ALSO

```
stcl, dss, dssc, DesSync, rstcl, exit
```

EXAMPLES

- Example of Invoking the stclc shell
- Example of Invoking a Tcl Script
- Example of Invoking Commands with stclc
- Example of Invoking Commands including a Quoted String

Example of Invoking the stclc shell

```
This example invokes the stclc shell:
  % stclc
  stcl>
```

Example of Invoking a Tcl Script

```
This example invokes a Tcl script that populates a work area:
  % stclc update.tcl
```

Example of Invoking Commands with stclc

```
This example executes the 'ls' command followed by the 'spwd' command:
  % stclc -exp 'ls;spwd'
```

Example of Invoking Commands including a Quoted String

```
This example shows a command string that includes double quotes:
  % stclc -exp 'ci -keep -comment "Fixed defect 4354" file.c'
```

Client Shell Control

alias

alias Command

NAME

```
alias               - Creates, shows, or removes a command alias
```

DESCRIPTION

```
This command defines an alias for a command or series of commands.
The alias persists from one DesignSync session to the next unless
you use the -temporary option.

The symbols $1 through $n provide placeholders for argument
substitution. You can create an alias for a sequence of commands.
Use the symbol '&&' to separate commands on the command line.

You cannot use the alias command to redefine the behavior of
built-in DesignSync commands. For example, if you define an alias
called 'co', DesignSync ignores the alias definition when you use
the 'co' command.

Alias definitions remember the mode, dss/dssc or stcl/stclc, in
which they were defined. For example, if while in stcl/stclc mode you
create an alias containing Tcl constructs, the alias will always
execute in stcl/stclc mode even if you change to dss/dssc mode. Aliases
provide the mechanism for you to define new stcl commands and make
them available from dss/dssc.
```

SYNOPSIS

```
alias  [-args <#>] {-list | -delete <alias_name> | <alias_name>
{<cmd> [&& <cmd> ...]}} [-temporary] [--]
```

OPTIONS

- -args
- -delete
- -list
- -temporary
- --

-args

```
 -args <#>       Number of arguments accepted by the alias. A value of "*"
                 represents any number of arguments.
```

-delete

```
 -delete <name>  Delete the specified alias.
```

-list

```
 -list           Display a list of known aliases.
```

-temporary

```
 -temporary      Temporarily create or delete an alias. Without the
                 -temporary option, the alias definition or deletion
                 persists through future sessions.
```

--

```
 --              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
```

EXAMPLE

```
This example creates a command called 'lss', which lists a directory
using the "-report status" switch to report whether objects are up
to date.

   stclc> alias -args * -- lss ls -report status {$*}
```

exit

exit Command

NAME

```
exit                - Ends a DesignSync session, or an stcl script
```

DESCRIPTION

```
This command ends your DesignSync shell (dss/dssc/stcl/stclc)
session, or an stcl script. This command is the Tcl exit command
with one extension: the ability to bring down syncd using the
-daemon option.

Note that 'exit' within a script not only stops script execution, but
also exits your DesignSync shell session. To avoid this behavior,
stcl programmers should use 'return' and 'error' in their scripts
instead of 'exit'.

You can optionally specify an integer exit status to pass to the
parent process (OS shell). The default exit status is 0, which
indicates success.
```

SYNOPSIS

```
exit [-daemon [-force]] [<status>]
```

OPTIONS

- -daemon
- -force

-daemon

```
-daemon     Brings down syncd when exiting stcl or dss. This option
            has no effect when specified from the DesignSync
            graphical interface, stclc, or dssc, none of which
            communicates with syncd. The syncd process will not exit
            if there is another active dss/stcl session connected
            to syncd or if syncd is locked (see the 'syncdadmin
            lock' command) unless you specify -force.
```

-force

```
-force      Valid only when -daemon is specified, forces syncd to
            exit even if there is another active dss/stcl session
            connected to syncd or if syncd is locked (see
            the 'syncdadmin lock' command).
```

RETURN VALUE

```
The return value is not of the 'exit' command itself, but of
the shell or script that 'exit' is terminating. The default is '0'
(success), but you can specify an alternative value using the
status argument.
```

SEE ALSO

```
syncdadmin
```

EXAMPLES

- Example of Exiting a dssc Session
- Example of Exiting a stcl Session and Stoppping the syncdaemon

Example of Exiting a dssc Session

```
This example exits a dssc session. The default exit status is
0 (success).

  dss> exit
  % echo $status
  0
  %
```

Example of Exiting a stcl Session and Stoppping the syncdaemon

```
This example exits an stcl session, returns a failure exit status,
and brings down syncd.
  stcl> exit -daemon 1
  % echo $status
  1
  %
```

log

log Command

NAME

log - Logs DesignSync commands and results to a file

DESCRIPTION

This command controls the logging of DesignSync commands and command output. Use the run command to execute log files (no editing is needed) in order to repeat a series of commands.

The log file is put 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) unless you change the default using the -defaultdir option. Note that the run command and the log command use the same default directory to locate files.

The default log filename is dss_<date>_<time>.log, where <date> and <time> reflect when the log file was created. For example, a log file of dss_04052000_111258.log was created on April 5, 2000, at 11:12:58 AM. The time stamp ensures filename uniqueness so that log files from previous DesignSync sessions are retained. If you prefer to always use the same filename so that you do not collect log files from past sessions, you can do so with the log command. If a file with the same name as the specified log file already exists, then it is overwritten without warning. If you do not specify an extension for the log file, a '.log' extension is used by default.

The default file size for the log file is 10MB. If the log file grows beyond the maximum log file size, the contents of the log file is moved to dss_<date>_<time>.bak.log, the log file is zeroed out, and the client continues logging. This process repeats every time the session log limit is reached. DesignSync only maintains one backup file, which essentially has the same maximum size as the log file. For information on changing the default log size, see the DesignSync Data Manager Administrator's Guide.

To prevent large numbers of dss_<date>_<time>.log files from collecting in your log directory, DesignSync automatically deletes log files that are older than 2 days. If more than 20 log files remain, DesignSync deletes all log files older than 1 hour. If you want to retain a log file indefinitely, you must move or rename the file.

If you are an administrator, you can change the logging settings for all users on a LAN or for all users of a particular project. See "The Logging Tab" topic in SyncAdmin online help for more information.

SYNOPSIS

```
log [-defaultdir <dir>] [-off | -on] [-nooutput | -output] [-state]
    [--] [<logFileName>]
```

ARGUMENTS

- Log File Name

Log File Name

```
<logFileName>      Optionally enter a log file name.  If no name is
                   specified, DesignSync uses the default log name,
                   as detailed in the DESCRIPTION section.
```

OPTIONS

- -defaultdir
- -nooutput
- off
- -on
- -output
- -state
- --

-defaultdir

```
-defaultdir <dir>  Set the default log directory. This value is
                   saved between sessions. If you have not set a
                   default log directory, then your home directory
                   is used.
```

-nooutput

```
-nooutput          Record only the commands being executed. The
                   default is for logging of both commands and
                   command output (-output).
```

off

```
-off               Turn logging off. When you specify this option,
                   information is displayed indicating not only
                   that logging is disabled, but also the state
                   of logging if it were enabled (what is being
                   logged, the current log file, and the default
                   log directory).
```

-on

```
-on                Turn logging on. Turning logging off and
                   then back on appends to the current log file if
                   you do not specify a log filename.
```

-output

```
-output            Record command results as comments in addition
                   to the commands themselves. This behavior is the
                   default.
```

-state

```
-state             Display current logging state. Shows what
                   log file is currently in use, whether logging
                   is enabled, how much detail is being logged,
                   and the name of the default log file directory.
                   Specifying the log command without options has
                   the same behavior as specifying -state.
```

--

```
--                 Indicates that the command should stop looking for
                   command options. Use this option when you
                   specify a log file whose name begins with a
                   hyphen (-).
```

RETURN VALUE

```
none
```

SEE ALSO

```
run
```

EXAMPLES

- Example of Setting the Default Log File Directory
- Example Showing the Current Logging State

Example of Setting the Default Log File Directory

```
This example sets the default directory for the creation of log files
for this and future sessions, and specifies the log file for the
current session as 'pcimaster.log'. Specifying a default log
directory and a log file are mutually exclusive operations, so two
log commands must be used:
  dss> log -defaultdir c:\Logs
  dss> log pcimaster.log
```

Example Showing the Current Logging State

```
This example displays the current logging state:
  dss> log -state    (-state is optional)
  Logging: ON
   Output: Commands Only
  LogFile: c:\Logs\pcimaster.log
  Default Logging Dir: c:\Logs
```

more

more Command

NAME

```
more                - Controls the paging of command output
```

DESCRIPTION

```
This command controls the paging of command output. You can precede
any DesignSync command with "more" to prevent the command output
from scrolling off the window. Output pauses after the number of
lines determined as follows:
  1. As specified by the -lines option.
  2. If -lines is not specified, as determined by the size of your
     window.
  3. If "more" is unable to determine the size of your window (for
     example, you are using a telnet session), the default is 20 lines.

To see the next block of output, press the Return (Enter) key. To
stop the "more" command, use Ctrl-c; the command whose output you are
paging may or may not be interrupted depending on if and when the
command checks for interrupts.

Important: The command whose output you are paging is
           suspended when the output is paused. The operation
           continues when you press the Return key.
```

SYNOPSIS

```
more [-lines <n>] [--] <command>
```

OPTIONS

- -lines
- --

-lines

```
-lines <n>   Specifies the number of lines to display in the
             window before pausing. A value of "0" means scrolling
             is not controlled; all output is displayed without
             pauses.

             If you do not specify -lines, the "more" command
             defaults to the number of lines that can be displayed
             in your window. In cases where "more" cannot determine
             your window size, the default is 20 lines.

             Note: You must specify -lines before the command
                   you are executing. For example,
                   "more ls -lines 5" is invalid and must instead be
                   "more -lines 5 ls".
```

--

```
--             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
```

EXAMPLES

- Example Showing More With a Specified Amount of Lines
- Example Showing More with the Default Amount of Lines

Example Showing More With a Specified Amount of Lines

```
The following example pauses the "ci" operation after each
15 lines of output.
  dss> more -lines 15 ci -nocomment -recursive .
```

Example Showing More with the Default Amount of Lines

```
The following example lists all objects in the current folder,
pausing after each window of output. For example, if your window
size is 40 lines, then each block of output is 40 lines.
  dss> more ls
  ...
```

prompt

prompt Command

NAME

```
prompt              - Sets the dss/dssc command-line prompt
```

DESCRIPTION

```
This command sets the dss or dssc shell command-line prompt. The
prompt can be either the URL of the current working directory, or
the default prompt of 'dss>'.

Note: You can only specify the prompt for dss and dssc shells, not
for stcl or stclc shells.
```

SYNOPSIS

```
prompt [-default | -url]
```

OPTIONS

- -default
- -url

-default

```
-default    Sets the prompt to the string 'dss>'.
```

-url

```
-url        Sets the prompt to the current working directory URL. If
            you change working directories, the prompt
            automatically updates.
```

RETURN VALUE

```
none
```

SEE ALSO

```
dss, dssc, scd, spwd
```

EXAMPLES

```
This example demonstrates the prompt command:
  dss> prompt -url
  file:///home/goss/Projects/Sportster/code> scd ..
  file:///home/goss/Projects/Sportster> prompt -default
  dss>
```

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
- -server
- -script
- -urlparams

-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

- If -output is not specified, returns (and displays) the script output.
- 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
```

record

record Command

NAME

```
record              - Captures command output
```

DESCRIPTION

```
This command captures the output from a command and stores it in a
variable. stcl programmers can use this command as a debugging aid.

The two required arguments to the record command are the command to
execute and the variable in which to store the command output.
Note that 'record' differs from 'set' in that 'record' stores
the command output whereas 'set' would store the command's return
value (see Examples for an illustration).

The record command returns or throws whatever the command being executed
produces. Therefore, replacing any command with a corresponding
record command does not affect script execution.

The record command runs in transaction mode and caches data about a
file in memory. Subsequent operations may use the cached data, even
when file has been updated on the disk.
```

SYNOPSIS

```
record <command> <variable>
```

OPTIONS

```
none
```

RETURN VALUE

```
Same as the command being executed.
```

SEE ALSO

```
puts
```

EXAMPLES

- Example of Recording to a Variable
- Example Showing Typical Usage of Record
- Example Showing Using Command Arguments with Record

Example of Recording to a Variable

```
The following example stores the result of an ls command in the
variable 'listing':
  stcl> record ls listing
```

Example Showing Typical Usage of Record

```
This example shows how grouping operators are commonly used with
the record command:

  catch {record {ci -new -keep -rec -com $comment *.cds} text} ret
```

Example Showing Using Command Arguments with Record

```
This example demonstrates the different behavior of the 'record'
command:

  stcl> record {co -lock -nocom top.v} x
  {Objects succeeded (1)} {}
  stcl> puts $x

  Beginning Check out operation...

  Checking out: top.v : Success - Checked Out  version: 1.1 -> 1.2

Versus the 'set' command:

  stcl> set x [co -lock -nocom top.v]
```

```
Beginning Check out operation...

Checking out: top.v : Success - Checked Out  version: 1.1 -> 1.2
{Objects succeeded (1)} {}
stcl> puts $x
{Objects succeeded (1)} {}
stcl>
```

Module-Based Design

Workspace Setup

Enterprise Design Development Area

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:
     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
- 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
- -gui
- -suite
- -[no]update
- -version

-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 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     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 "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|   Specifies the amount of output supplied by the
 normal|verbose  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 mk

Description

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
```

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
- Example Showing Removing a Development in Interactive Mode

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'.
```

Module Views

view

view Command

NAME

```
view                - Modules views commands
```

DESCRIPTION

```
These commands allow you to create, manipulate, and remove module
views. Module views use filters, extended include filters, and
hreffilters to define a subset of module members. Usually a module view
is created to support a particular user group that does not require
access to the whole module, but would prefer to work with only a
relevant subset of files.

The view commands help provides information on creating and
manipulating the module view. For help defining the module view, see
the ENOVIA Synchronicity DesignSync Data Manager Administration Guide.

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

```
view <view_command> [<view_command_options>]

Usage: view [get, check, list, put, remove]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

EXAMPLES

```
See specific "view" commands.
```

view check

view check Command

NAME

```
view check          - verifies the syntax & semantics of the view
                      definition
```

DESCRIPTION

```
This command parses the supplied view file or files and indicates
whether the views define within are syntactically and semantically
correct.  If they are not correct, DesignSync provides a list of the
failures.

For information about defining module views, see the ENOVIA
Synchronicity DesignSync Data Manager Administration Guide.

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

SYNOPSIS

```
view check <file>[,<file2>[,...]]
```

ARGUMENTS

- File

File

```
<file>              One or more file objects containing a modules
                    view definition.
```

RETURN VALUE

```
In stcl/stclc mode, if all views in the local view file are
syntactically and semantically correct, the view check command
returns an empty string ("").  If the local view file is not
syntactically and semantically correct, the view check command
returns an appropriate error message indicating what was incorrect.
```

SEE ALSO

```
view list
```

EXAMPLES

- Example of Reading Correct View Definitions
- Example of Reading an Incorrect View Definition

Example of Reading Correct View Definitions

```
This example shows reading a view definitions file with two views
defined correctly.

$ view check docviewdef.txt
 /home/ViewDefs/docviewdef.txt: Reading view definition file ...
 All view definitions successfully read.
```

Example of Reading an Incorrect View Definition

```
This example shows reading a view definitions file where the
description is missing in one of the view definitions.

$ view check generalviewdef.txt
 /home/ViewDefs/generalviewdef.txt: Reading view definition file ...
 ERROR: View definition does not contain an even number of elements.
       Please ensure that the definition is a set of name/value
       pairs.
       (Name RTL Description     Filter +.../*.h,+.../*.c,+.../*.tcl
        HrefFilter "")

 Errors occurred while checking view definition files.  Please review
 the command output and correct the errors.
```

view get

view get Command

NAME

```
view get            - Shows the contents of a module view
```

DESCRIPTION

This command displays the contents of a specified module view. The command displays the contents of the first available module view definition that matches the module view name specified with -name. The command takes a URL argument which serves as the starting point for the search. If no matching module view is found in that starting point, the command traverses up the directory tree to the Modules area looking for a match for the specified module view name.

For information about defining module views, see the ENOVIA Synchronicity DesignSync Data Manager Administration Guide.

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

```
view get [-format text|list] -name <viewname> [-output <localfile>]
         [--] <argument>
```

ARGUMENTS

- Module URL

Module URL

<module URL>	URL of the module on which to find the view. 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: The argument is required. You can only specify one argument, however the command will climb the hierarchy in order to find specified view.

OPTIONS

- -format
- -name
- -output

-format

```
-format text|list  Determines the format of the output.
                    Valid values are:
                    o text - Display a text table with headers and
                             columns. (Default)
                    o list - Displays a list with the following
                             format:
                              {{ViewLocation <URL-where-view-found>}
                             {Name <viewname>} {Description {text
                             description}} {Filters
                             {comma-separated-list}} {HrefFilters
                             HrefFilter {comma-separated-list}
                             {ViewPropagationMode none|name|rules}}

                   Note: The output is written to the screen unless
                   the -output option is specified.
```

-name

```
-name <viewname>   Specifies the name of the Module View targeted in
                  the search. This option is required.
```

-output

```
-output <localfile> Specifies the name of the local output file in
                    which to write the command output. If the
                   specified file already exists, the file is
                   overwritten with the new contents.

                    If -output is not specified, the command output
                    echoes to the screen or command output window.
```

RETURN VALUE

```
For a successfully executed command, DesignSync returns either a null
string ("") or a TCL list containing the module view definition.
If there is no view matching the specified name, DesignSync returns
an error.
```

SEE ALSO

```
populate, setview, showstatus, url view, view list, view put,
view remove
```

EXAMPLES

- Example of Getting the View in Text Format
- Example of Getting the View in List Format
- Example using the Extended Include Syntax

Example of Getting the View in Text Format

```
 This example shows running the view get command on the DOC view.
 stcl> view get -name DOC sync://srv2.ABCo.com:2647/Modules/Chip

  Sending get request to server ...

  Name  Filter                                                Href Filter
 Location                            Description
 ------------------------------------------------------------------------
--
  DOC   +*.doc,+*.txt,+*.html,+*.htm,+*.xml,+*.gif,+*.jpg  doc,images
  sync://qelwsun14:30126/Modules/Chip  Basic Documentation view,
  formatted for easy reading
```

Example of Getting the View in List Format

```
 This example shows the DOC view using the -format list option.

 stcl> view get -name DOC -format list sync://srv2.ABCo.com:2647/Modules/Chip
 {Description {Basic Documentation view, formatted for easy reading}
 Location sync://qelwsun14:30126/Modules/Chip Name DOC Filter
 +*.doc,+*.txt,+*.html,+*.htm,+*.xml,+*.gif,+*.jpg HrefFilter
 doc,images}
```

Example using the Extended Include Syntax

```
 This example shows the extended include syntax along with a standard
 include filter and HrefFilter.

view get -format list -name RTLSPECIFICFILES
 sync://srv2.ABCo.com:2647/Modules/SyncInc/DevSuiteDoc
 {Description {RTL view with standard include, extended include, and
 href filters} Location sync://src:2647/Modules/SyncInc/DevSuiteDoc
 Name RTLSPECIFICFILES Filter
 ++|.../RTL/NZ12-1.VHDL|.../RTL/NZ12-2.VHDL|.../RTL/NZ12-3.VHDL|,+.../*.doc
 HrefFilter doc,layout-tools}
```

view list

view list Command

NAME

```
view list           - Lists available module views and location
```

DESCRIPTION

```
This command lists module views available for the indicated server
module URL, along with the URL associated which each module
view. Module views are listed in the order they are found. When
multiple views are found at one level, they are listed
alphabetically. For information about defining module views, see
the ENOVIA Synchronicity DesignSync Data Manager Administration Guide.

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

```
view list [-format text|list] [--] <argument>
```

ARGUMENTS

- Module URL

Module URL

```
<module URL>        URL of the module on which to find the
                    view. 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: The argument is required.  You can only
                   specify one argument, however the command will
                   climb the hierarchy in order to find specified
                   view.
```

OPTIONS

- -format

-format

```
-format text|list  Determines the format of the output.
                    Valid values are:
                    o text - Display a text table with headers and
                             columns. (Default)
                    o list - Displays a list with the following
                             format:
                              {{<ViewName> <ViewLocationURL}...}
```

RETURN VALUE

```
For a successfully executed command, DesignSync returns either a null
string ("") or a TCL list containing the list of module views.
If no views are found, DesignSync returns a null string("").
```

SEE ALSO

```
showstatus, populate, setview, view get, view put, view remove
```

EXAMPLES

- Example of Showing All the Views in Text Format
- Example of Showing All the Views in List Format

Example of Showing All the Views in Text Format

```
This example shows all the views in the Chip category using the
-format text option.

stcl> view list -format text sync://srv2.ABCo.com:2647/Modules/Chip

# Sending list request to server ...

 View              Location
 ------------------------------------------------
 DOC               sync://srv2.ABCo.com:30126/Modules/Chip
 RTL               sync://srv2.ABCo.com:30126/Modules/Chip  VIEW LOCATION
```

Example of Showing All the Views in List Format

```
This example shows all the views in the Chip category using the
-format list option:

stcl> view list -format list sync://srv2.ABCo.com:2647/Modules/Chip
DOC sync://qelwsun14:30126/Modules/Chip RTL
sync://qelwsun14:30126/Modules/Chip
```

view put

view put Command

NAME

```
view put            - Uploads module view definitions to the server
```

DESCRIPTION

```
This command stores the definition of one or more module views from
an input file to the specified module location on the server. For
information about defining the input file, see the ENOVIA
Synchronicity DesignSync Data Manager Administration Guide.  The
input file contains filters and hreffilters used to determine the
contents of the view.

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

```
view put -all|-name <viewName> -infile <inputFile> [-[no]replace]
         [--] <argument>
```

ARGUMENTS

- Module URL

Module URL

```
<module URL>       URL of the module location on which to find the
                   view. Specify the URL as follows:
                   sync[s]://<host>[:<port>]/Modules/[<Category>...]
                    [/<module>;]
                   where <host> is the SyncServer on which the
                   category or module resides, <port> is the SyncServer
                   port number, <Category> identifies the
                   optional category path, and <module> is the name
                   of the module.
```

OPTIONS

- -all

- -infile
- -name
- -[no]replace

-all

```
-all              Stores all module view definitions from the input
                  file on the server.

                  The -all option and the -name option are mutually
                  exclusive. If the -all option is not specified,
                  you must specify the -name option.
```

-infile

```
-infile           Specifies the local file that contains one or more
 <inputFile>      module view  definition to save onto the
                  server. The DesignSync server expects a specific
                 format.  The format is described in detail in the
                 ENOVIA Synchronicity DesignSync Data Manager
                 Administrator's Guide.
```

-name

```
-name <viewName>  Specifies the name of the module view to save on
                  the server. This name should correspond to a
                  defined name in the input file.

                  The -name and -all options are mutually
                  exclusive.  If -name option is not specified, you
                  must specify the -all option.
```

-[no]replace

```
-[no]replace      Specifies whether the module view definition
                  should replace an existing view definition.

                   -noreplace indicates that the module view
                 definition should not replace an existing view
                 defintion for the same module location.  If a
                 module view with that name already exists, the
                 command will fail. (Default)

                 -replace indicates that the module view definition
                should replace an existing module view definition
                with the same name in the same module location.
```

RETURN VALUES

```
For a successfully executed command, DesignSync returns an empty
string ().
For each successfully saved module view, DesignSync displays a
success message.
If errors are detected when a file is being read or saved, DesignSync
displays the error.
```

SEE ALSO

```
populate, setview, showstatus, view get, view list, view remove
```

EXAMPLES

- Example of Loading a Single View from a File
- Example of Loading All the Views in a File

Example of Loading a Single View from a File

```
This example shows the loading of a single module view from a file
containing a list of views.
view put -name DOC -infile ~/viewsdef/chipviews.txt
sync://srv2.ABCo.com:2647/Modules/Chip

 ~/viewsdef/chipviews.txt: Reading view definition file ...

 Sending request to server ...

 View DOC: successfully saved.
```

Example of Loading All the Views in a File

```
This example shows the loading of a view file containing a list of
views. Note that the DOC view is already loaded and can't be replaced
without the -replace option.

dss> view put -all -infile ~/viewsdef/chipviews.txt
sync://srv2.ABCo.com:2647/Modules/Chip

 ~/viewsdef/chipviews.txt: Reading view definition file ...

 Sending request to server ...
 View RTL: successfully saved.
 View DOC: Cannot replace existing module view.
 View RTLNOHREF: successfully saved.
 View RTLNOF: successfully saved.
```

```
View RTLNODESCRIPTION: successfully saved.
Operation partially or completely failed.
```

view remove

view remove Command

NAME

```
view remove          - Removes a module view from the server
```

DESCRIPTION

```
This command removes the specified view from the server. It does not
remove the file containing the view description from its 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.
```

SYNOPSIS

```
view remove -name <viewName> [--] <argument>
```

ARGUMENTS

- Module URL

Module URL

```
<module URL>        URL of the module on which to find the
                    view. 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.
```

OPTIONS

- -name

-name

```
-name <viewName>   Specifies the name of the module view to remove
                   from the server. You may only specify one name per
                   command.
```

RETURN VALUES

```
For a successfully executed command, DesignSync returns the name of
any module view removed and an empty string ().
If the command fails, DesignSync returns an error explaining the
failure.
```

SEE ALSO

```
populate, setview, showstatus, view get, view list, view put
```

EXAMPLES

```
This example shows removing the RTL view from the server.
dss> view remove -name RTL sync://srv2.ABCo.com:2647/Modules/Chip

 Sending delete request to server ...

 View RTL: successfully removed.
```

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
- Folder Path Argument

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
- Example Showing Listing the Exclusions in List Format

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)

- Incremental Versus Full Populate (File-based)
- How Populate Handles Retired Objects (File-based)
- Merging Across Branches (File-based)
- Populate Versus Checkout (File-based)
- Understanding the Output (File-based)
- Forcing, Replacing, and Non-Replacing Modes (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

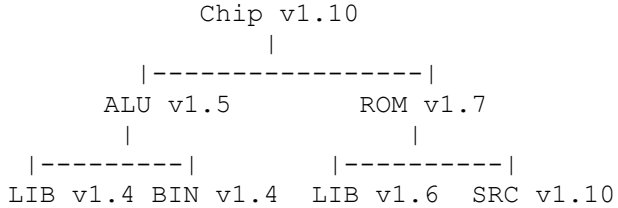
Working in a peer-based structure, where your modules are populated in a flat directory setting, your workspace may look something like this:
/home/workspace/TOP
/home/workspace/ROM
/home/workspace/COM
/home/workspace/LIB

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.

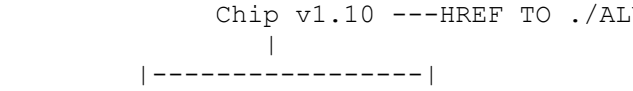
If an href is placed higher in the peer structure, however; it will become the overriding href. So, for example, if you add an href for TOP to LIB, as shown:
```
 TOP -> ROM@1.23 -relpath ../ROM
     -> COM@1.15 -relpath ../COM
     -> LIB@1.5 -relpath ../LIB
```

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:

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

- 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.

- You can have full copies of all the modules in an mcache.
- 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.

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.
```

- 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:

  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]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)

- -lock -reference (Module-based)
- -lock -reference (File-based)
- -log
- -mcachemode (Module-based)
- -mcachemode (Legacy-based)
- -mcachepaths (Module / Legacy-based)
- -[no]merge (Module-based)
- -merge (File-based)
- -mirror (File-based)
- -modulecontext (Module-based)
- -[no]new (Module-based)
- -overlay
- -path (Module-based)
- -path (Legacy-based)
- -path (File-based)
- -[no]recursive (Module-based)
- -[no]recursive (Legacy-based)
- -[no]recursive (File-based)
- -reference
- -[no]replace (Module-based)
- -[no]replace (File-based)
- -report
- -[no]retain
- -savelocal
- -share
- -target (Legacy-based)
- -trigarg
- -[no]unifystate
- -version (Module-based)
- -version (File / Legacy-based)
- -view (Module-based)
- -xtras (Module-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:
  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:
o 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:
- 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"
- 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.

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:
o 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.
o 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.

setfilter

setfilter Command

NAME

```
setfilter           - Sets the persistent filter or hreffilter list
```

DESCRIPTION

```
This command sets the persistent filter or hreffilter for a
module. This filter is applied each time the module is populated.
The persistent filters defined here are applied to the appropriate
commands before any filters or hreffilters specified on the command
line are applied.

If a module is initially populated using a -filter or -hreffilter on
the command line, a persistent filter matching those settings is set
automatically for that module.

After a filter has been changed using the setfilter command, the next
populate of the module is a full populate, since the filter has
changed. Performing a setfilter replaces any previous filters set,
including the filters set automatically with a filtered populate,
with the new filter.
```

SYNOPSIS

```
setfilter [-filter | -hreffilter] [-[no]recursive][--]
          <workspace module> <filter>|<hreffilter>
```

ARGUMENTS

- Workspace Module
- Filter
- Hreffilter

Workspace Module

```
<workspace module>  Specify the module identifier for the module
                    receiving the persistent filter.  The module must
                    have already been populated in the workspace.
```

Filter

<filter>

Specify one or more extended glob-style expressions to identify an exact subset of module objects on which to operate. The expressions should be separated by commas, for example:
+top*/.../*.v,-.../a*

If you specify a null character ("") as the filter argument, all filter values are removed from the persistent filter list including the filters created during a filtered populate. The next time the directory is populated, DesignSync performs a full populate.

Prepend a '-' character to a glob-style expression to identify objects to be excluded.(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'.

Hreffilter

<hreffilter> Excludes href values during recursive populate of module hierarchies, excluding particular submodules from the populate. Note that unlike the -filter option which lets you include and exclude items, the -hreffilter option only excludes hrefs.

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.

Note: You can set both types of hreffilters, simple and hierarchical, for your workspace, but they must set in different operations.

A simple href filter is a simple leaf module name or the href name (specified when you added the href). 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. When you specify a simple href, you must run the setfilter command in -norecursive mode (Default).

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 you specify a hierarchical href filter, you must run the setfilter command in -recursive mode.

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.
```

OPTIONS

- -filter
- -hreffilter
- -recursive
- --

-filter

```
 -filter            Specifies that the persistent filter being set is
                    a filter argument, not an href filter.  Filter
                    arguments can both exclude or include elements.
```

-hreffilter

```
 -hreffilter        Specifies that the persistent filter being set is
                    an hreffilter argument which prevents the
                    updating of the specified hrefs during a populate
                    operation.
```

-recursive

```
 -[no]recursive     Specifies whether the persistent filter is applied
                    recursively through the module hierarchy.

                    -norecursive does not apply the persistent
                     filter recursively (Default). This is the
                     standard operating mode for filters and simple
                     href filters.

                    -recursive applies the persistent filter
```

recursively through the module hierarchy. This is the required mode when using hierarchical href filters.

Note: When setting or removing hreffilters, only one type of hreffilter, simple or hierarchical, may be set at a time because they require different -recursive/-norecursive 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

If the set filter command is successful, DesignSync returns an empty string (""). If the module does not exist in the workspace or the filter cannot be set, the setfilter commands returns an error.

SEE ALSO

populate, url filter

EXAMPLES

- Example of setting a filter
- Example of setting an href filter
- Example of setting a hierarchical href filter
- Example of clearing an href filter
- Example of clearing a hierarchical href filter

Example of setting a filter

This example shows setting a filter on your module, Chip, to filter out documentation files. After setting the filter, you may want to populate to bring the changes into your workspace.

```
dss> setfilter -filter  Chip%0 -.../doc/.../*
 Set Filter operation successfully completed.
```

Example of setting an href filter

```
This filters out any submodule named BIN from your Chip module
hierarchy. After setting the hreffilter, you may want to populate to
bring the changes into your workspace.

dss> setfilter -norecursive -hreffilter Chip%0 BIN
 Set Filter operation successfully completed.
```

Example of setting a hierarchical href filter

```
This filters out the JRE/BIN submodule hierarchy within
the Chip hierarchy.  This operation is recursive through the module.
After setting the hierarchical hreffilter, you may want to populate
to bring the changes into your workspace.

dss> setfilter -recursive -hreffilter Chip%0 JRE/BIN
 <Chip%0> Persistent hierarchical href filters set to <JRE/BIN>.
 Set Filter operation successfully completed.
```

Example of clearing an href filter

```
This example removes all (simple) href filters set on the Chip module.

dss> setfilter -norecursive -hreffilter Chip%0 ""
 <Chip%0> Persistent href filter cleared. It will no longer be used.
 Set Filter operation successfully completed.
```

Example of clearing a hierarchical href filter

```
This example removes all hierarchical href filters set on the Chip
module.
dss> setfilter -recursive -hreffilter Chip%0 ""
 <Chip%0> Persistent hierarchical href filter cleared. It will no
longer be used.
Set Filter operation successfully completed.
```

setselector

setselector Command

NAME

```
setselector         - Sets the persistent selector list
```

DESCRIPTION

- Notes for Using setselector (Module-based)
- Notes for Using setselector (File-based)
- Valid Selectors for Module Objects (Module-based)
- Valid Selectors for Files-Based Objects (File-based)
- Configuration Mapping (Legacy-based)

```
 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>)
- 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 Using the Persistent Selector List in a multi-branch environment (File-based)

- Example of Using Setselector to Auto-Branch (File-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.
```

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 (File-based)
- --

-[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)
- Unsetting the Workspace Root for a Module (Module-based)
- Setting the Workspace Root For Files-Based Objects (File-based)

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.
```

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
- Local Module (Module-based)

- Local Folder (File-based)

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 (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
- Example of Associating a Server Vault with a Specified Directory
- Example of Changing the Vault Association Recursively in a Workspace
- Example of Associating a Local Vault with a Specified Directory

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.
```

setview

setview Command

NAME

```
 setview             - Associates a view with a work area
```

DESCRIPTION

```
 This command sets the persistent module view, as stored in an
 object's local metadata, for the specified workspace. Any previous
 view list is overwritten or cleared. When the persistent view has
 been set or cleared, the next populate is automatically a full
 populate operation and all subsequent populate operations use the
 stored view definition.

 Notes:
 o When the view option is used on an initial populate, it
   creates the workspace with the persistent module view in the
   metadata.
```

```
o When operating recursively, if an mcache link is encountered in the
  workspace the module view associated with the mcache instance is
  updated, but the link is not traversed and the metadata within the
  mcache is not changed.

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

```
setview [-[no]recursive [--] <argument> <view>[,<view>...]
```

ARGUMENTS

- Workspace Module

Workspace Module

```
<workspace       Workspace module instance name, for example
 module>         Chip%1.
```

VIEWS

- Name of View(s)

Name of View(s)

```
<view>           Name of the view(s) to associate with the
[,<view>...]     workspace. If you are associating more than one
                 view with the workspace, you must separate the view
                 names with a comma.

                 To remove all view associations from the workspace,
                 specify, "none" as the view name.

                 Note: The view specified overwrites any existing
                 views set on the workspace.  If you want to add a
                 view to the existing views set on the workspace,
                 you must specify ALL of the views for the
                 workspace.
```

OPTIONS

- -recursive

-recursive

```
-[no]recursive   Determines whether to perform this operation on the
                 instance specified only, or recurse through the
                 module hierarchy.

                 -norecursive performs the setview only on the
                 specified module. (Default)

                 -recursive recurses the operation through the
                 modules hierarchy.

                 Note: When operating recursively, if an mcache link
                 is encountered in the workspace the module view
                 associated with the mcache instance is updated, but
                 the link is not traversed and the metadata within
                 the mcache is not changed.
```

RETURN VALUES

```
If the setview command is successful, DesignSync returns an empty
string ("") and a success message is returned.
If the module does not exist in the workspace or the module view
cannot be set, the setview command returns an error and immediately
terminates.  It does not continue to attempt to traverse the module
hierarchy when run recursively.
```

SEE ALSO

```
populate, showstatus, view get, view list, view put, view remove
```

EXAMPLES

- Example of Setting a View for a Workspace

Example of Setting a View for a Workspace

```
dss> setview Chip%0 DOC
<Chip%2> Persistent view replaced, set to <DOC>.
Set View operation successfully completed.
```

Primary Revision Control

add

add Command

NAME

```
add                  - Adds new objects to a module
```

DESCRIPTION

- Understanding the Output

```
This command adds new objects to the workspace module. When the
objects are checked in, (using ci)  they are added to the module on
the server and a new module version is created on the server.

The -candidates option lists the path(s) to the unmanaged object(s) and
specifies the list of modules to which the object could be added. The
candidate modules listed will all have base directories at or above the
path of the unmanaged object.

Objects that have been added with the add command can be removed from
the list of files to be added being checked into the server, or
removed from the module after being checked in to the server.  Module
members that are removed in either case can be added back to the
server. The add command remembers that the object was added to
the server and indicates an "added back" status for these objects.

Notes:
o If an object has already been added to the module, you
  may also see the "added back" status if you attempt to add the
  object again.

o You cannot add an object located in an external module directory to
  a module regardless of whether the object is being added to a
  DesignSync module or an external module.

o Using syncexclude files, you can automatically exclude objects
  matching a defined pattern, for example, all .log files; from add
  operations. For more information about using exclude files, see the
  DesignSync User's Guide: Working with Exclude Files.

This command supports the command defaults system.
```

Understanding the Output

```
The add 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:

By default, or if you run the command with the '-report brief'
option, the add command outputs the following information:
```

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

If you run the add command with the '-report normal' option, the command displays all the information contained in -report brief, and the following additional information:

- A status message indicating the beginning of the add operation.
- A real-time list of the objects added to the workspace module.
- A status message indicating the end of the add operation.

If you run the add command with the '-report verbose' option, it displays all the information contained in -report normal and the following additional information.

- A status message indicating the directory currently being processed.
- A status message indicating that an object was not added to the module ("skipped") for each of the following conditions:
  - Non-versionable objects matching the wildcard pattern.
  - Objects that are already members of the module.
  - Objects that are already members of a different module.
  - Objects that are filtered out of the operation.
  - A status message indicating if no objects matched the add criteria. This can happen when no objects in the directory structure match a specified wildcard operation.
  - A status message indicating the end of the add operation.

If you run the add command with the -report error option, it displays the following information:

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

Note: If an object is explicitly specified and is already part of a module, you will see an error stating that the object was skipped. If an object is included in a recursive operation and is already part of a module, you will not see the error, it will be silently skipped.

SYNOPSIS

```
add  [-[no]candidates] [-filter <string>] [-modulecontext <workspace>]
[-[no]recursive [-[no]emptydirs]]]
[-report {error | brief | normal | verbose}] [-no[selected]] [--]
[<workspace module>] <argument> [<argument...]
```

ARGUMENTS

- Folder or Unmanaged Object
- Workspace Module
- Symbolic Link

Folder or Unmanaged Object

<folder> | <unmanaged object>	Adds a folder, objects in the folder, or specified objects to the specified workspace module. You can use wildcards in the argument. If the folder isn't a subfolder of the specified module, the add command fails.

Workspace Module

<workspace module>	When specified as the first argument, it indicates the module receiving new objects. You can also specify a workspace module with the -modulecontext option. If a workspace module is specified, it must be as the first argument. The -modulecontext option is mutually exclusive with specifying the workspace module.

Symbolic Link

<symbolic link>	Adds a symbolic links to a folders if management of symlinks is enabled, or follows the link to the target folder and adds the contents of the folder. If a folder symlink points to a folder outside the module base directory, the directory is not added to the module.

OPTIONS

- -[no]candidates
- -[no]emptydirs
- -filter
- -modulecontext
- -[no]recursive
- -report
- -[no]selected
- --

-[no]candidates

-[no]candidates	Determines to which modules the processed unmanaged objects can be added. -nocandidates runs the add command in active mode and does not provide a -candidates (dryrun) type operation. (Default) -candidates runs the add command in a dryrun style mode. When add is run -candidates, the output contains the pathnames to the unmanaged objects and the list of modules to which each object could be added. The modules listed are located at or above the directory holding the unmanaged objects. No objects are added to any workspace module via this option. Any other option that filters the command output also filters the candidates list. Note: any unmanaged objects at or below a directory which has been setvault to a files-based vault URL do not appear in the candidates output.

-[no]emptydirs

-[no]emptydirs	Determines whether an empty folder found during an add -recursive operation is added to the module. -noemptydirs does not add empty subfolders to the module.(Default) This means that any subfolders that do not contain files or subfolders that contain files, are not added to the module. -emptydirs adds empty subdirectories to the module explicitly. This is used when creating the framework for a module. Note: If a directory is not recursed into, for instance, if it is a link outside of the module, then the directory itself is not added, whether empty or not.

-filter

-filter <string>	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.

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; 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'.
```

-modulecontext

-modulecontext <context>	Identifies the module to which the objects are being added. Use the -modulecontext option to explicitly specify a module when there are multiple options for the objects being added, for example, when you have overlapping modules. If no modulecontext is specified, DesignSync uses smart module detection to identify the target module.For more information on how smart module detection identifies the target module, see the ENOVIA Synchronicity DesignSync Data Manager User's Guide topic: Understanding Smart Module Detection. 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 add 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.

-[no]recursive

-[no]recursive	Determines whether to add the specified folder or all objects in the folder and any subfolders. This option is ignored if the argument is not a module folder. -norecursive adds only the specified folder.(Default) Any objects in the folder are not added. This is used to explicitly add a folder to a module. If a folder is explicitly added to a module, it remains even if the folder becomes empty at any time. -recursive adds the contents of specified folder and any subfolders and the objects contained in those folders. Because this adds the folder implicitly, if the folder becomes empty, it is automatically removed from the module. Note: The recursive operation is always folder

recursive, not module recursive. This does not mean that all new members are checked into the same module. Smart module detection is used to determine the appropriate target module.

-report

-report error|brief| normal|verbose	Determines what information is returned in the output of the add command. The information each option returns is discussed in detail in the "Understanding the Output" section above. Valid values are: o error - provides error and warning messages only. o brief - lists all the objects added to the workspace module. o normal - indicates when the command begins and ends processing and lists all the object added to the workspace module. (Default) o verbose - provides full status for each object processed.

-[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.

--

--	Indicates that the command should stop looking for command options. Use this option when arguments to the command begin with a hyphen (-).

RETURN VALUE

By default, this command returns a count showing how many objects

```
succeeded and failed.

For a full description of the output, see the "Understanding the Output"
section.
```

SEE ALSO

```
command defaults, remove, ci, mkmod
```

EXAMPLES

- Adding a File
- Adding Folders Recursively
- Listing the Candidate Modules to Add Objects

Adding a File

```
This is an example of adding a file called "makefile" to the Chip
module.
stcl> add Chip makefile

 Beginning add operation...

 /Chip/makefile : Adding object

 Finished add operation.

 {Objects succeeded (1)} {}
```

Adding Folders Recursively

```
This example shows adding folders verilog and doc and the contents of
those directories to the Chip module using the verbose reporting
option.
stcl>add -recursive -report verbose Chip doc verilog

 Beginning add operation...

 /MyModules/Chip/doc : Process recursive directory
 /Chip/doc/Chip.doc : Adding object
 /MyModules/Chip/verilog : Process recursive directory
 /Chip/verilog/chip.v : Adding object

 Finished add operation.

 {Objects succeeded (2)} {}
```

Listing the Candidate Modules to Add Objects

```
This example shows using the -candidates option to see the list of
modules to which the unmanaged objects found could be added.

stcl> add -recursive -candidates .
PATH                            CANDIDATES
-------------------------------------------------------------------
MyModules/Chip/doc/Chip.doc     Chip%0
MyModules/Chip/ROM/rom.h        Chip%0, ROM%1
```

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
- -filter (Module-based)
- -[no]force
- -hreffilter (Module-based)
- -keep
- -mirror (File-based)
- -modulecontext (Module-based)
- -[no]recursive (Module-based)
- -[no]recursive (File-based)
- -reference
- -[no]retain
- -[no]selected
- -share
- -trigarg
- --

-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
- Understanding the Output
- Object States (Module-based)

- Determining the Objects to be Checked In (Module-based)
- Determining Which Branch is Selected for the Check In (Module-based)
- Filtering or Excluding Objects From Checkin (Module-based)
- Checking in Module Objects (Module-based)
- Branching Modules (Module-based)
- Automerging of Module Objects (Module-based)
- How Checkin Works with Enterprise Design Synchronization (Module-based)
- Checking in Legacy Module Data (Legacy-based)
- Object States (File-based)
- Determining the Objects to be Checked In (File-based)
- Determining Which Branch is Selected for the Check In (File-based)
- Filtering or Excluding Objects From Checkin (File-based)
- Interaction with Mirrors (File-based)

```
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:
 o Failure messages.
 o Warning messages.
 o Informational messages concerning the status of the checkin
   operation.
 o 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:
 o Failure messages.
 o Warning messages.
 o 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:

o 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'.

o 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:

- 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).

- 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.

- 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)
- -branch (Module-based)
- -branch (File-based)
- -[no]comment (Module-based)
- -[no]comment (File-based)
- -cfile
- -datatype (Module-based)
- -datatype (File-based)
- -[no]dryrun
- -exclude (Module-based)
- -exclude (File-based)
- -filter (Module-based)
- -[no]force

- -hreffilter (Module-based)
- -[no]hrefversions (Module-based)
- -[no]iflock (Module-based)
- -[no]iflock (File-based)
- -keep
- -keys
- -lock
- -mirror (File-based)
- -modulecontext (Module-based)
- -[no]new (Module-based)
- -[no]new (File-based)
- -[no]recursive (Module-based)
- -recursive (Legacy-based)
- -recursive (File-based)
- -reference
- -report
- -[no]resume (Module-based)
- -[no]retain
- -[no]retry (Module-based)
- -[no]selected
- -share
- -[no]skip (Module-based)
- -[no]skip (File-based)
- -tag (Module-based)
- -tag (File-based)
- -trigarg
- --

-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>    Performs the checkin on the branch specified by
 | -branch           the branch or version tag, auto-branch selector,
   auto(<branch>)    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      Identifies the module instance from which the
  <context>          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 recurses 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
```

mkmod

mkmod Command

NAME

```
mkmod               - Creates a module on a server
```

DESCRIPTION

- Understanding the Output

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

The data included in a module includes its own objects and
references to other modules, but not the contents of the referenced
modules. The mkmod operation creates the vault directory. The
appropriate vault subdirectories are created when you first check
data into the vault.

When you create the module, the mkmod operation creates all the
necessary server level directories required for the module.

Note: Modules are always created in the Modules area.
      (sync[s]://<host>:<port>/Modules)
```

You cannot use the mkmod command on an existing vault folder. You can upgrade legacy modules or existing vaults to modules with the "hcm' upgrade" command.

This command is subject to access controls on the server. See the ENOVIA Synchronicity Access Control Guide for details. Specifically, the topics "Access Controls for hcm mkmod" and "Creating a New Version of a Module". The latter topic is for the checkin aspect of the mkmod command.

This command supports the command defaults system. Although there is a -checkin option to the mkmod command, none of the command defaults for the "ci" command apply to the "mkmod -checkin" operation.

Understanding the Output

The mkmod 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 mkmod command outputs the following information:
- Failure messages.
- Warning messages.
- Informational messages concerning the status of the mkmod operation.
- Success/failure/skip status.

If you do not specify a -report value, or run the command with the default -report normal option, the mkmod command outputs all the information presented with -report brief and the following additional information:
- Success message for each object successfully checked in.
- Messages for objects excluded from the operation (due to exclusion filters).

If you run the command with the -report verbose option, the mkmod command outputs all the information presented with -report normal and the following additional information:
- Informational messages for the module creation.

If you run the command with the -report error option, the mkmod command outputs only the following information, which is related to the checkin activity:
- Failure messages.
- Warning messages.
- Success/failure/skip status messages.

SYNOPSIS

mkmod [-checkin [-[no]retry]] [-[no]comment "<text>"] [-filter <string>]

```
[-path <workspace_path>] [-report {error|brief|normal|verbose}]
<argument>
```

ARGUMENTS

- Server URL

Server URL

<Server URL>	Specifies the Synchronicity URL for the new module. The server URL takes the following form: <protocol>://<host>[:<port>]/Modules/[<category>]\ /<module name> o protocol specifies 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 specifies the hostname of the SyncServer for the new module. o port specifies the port number of the SyncServer. If no port number is specified, DesignSync uses the default port, which is 2647 for a standard connection or 2679 for an SSL connection. o Modules is the virtual location on the server for all modules. All modules must be in the modules area. o Category provides a means of organizing modules. Specify a virtual path category to group related modules. For example, if you work with two different types of projects, Chip design and CPU design, you can create two categories /Chip and /CPU to store the different modules for each type of project. These paths do not map to actual paths on the server. o Module name specifies the name of the module. Module and category names must conform to the following standards: - Must contain only printable characters - May not contain spaces - May not contain any of the following characters: ~ ! @ # $ % ^ & * ( ) , ; : | ` ' " = [ ] / \ < > Tip: Module names should start with an initial capital letter in order to allow you to easily distinguish them from

```
                    workspaces.
```

OPTIONS

- -checkin
- -[no]comment
- -filter
- -path
- -report
- -[no]retry

-checkin

```
 -checkin            The -checkin option is a performance optimization
                     that is used only in conjunction with the -path
                     option, to checkin unmanaged data from the -path
                     workspace to the module. The option improves the
                    performance for the initial check in of large
                    datasets. Smaller datasets, in the hundreds or
                    thousands of files range, may not benefit from
                     this enhancement.

                     Note: By default, the checkin option
                     automatically retries the operation on failure
                     using the settings in the registry keys to
                     determine the number of retries and timeout
                     interval. Using the -retry option may have an
                     impact on the performance of the operation. This
                     operation will also trigger scripts using the
                     retryOnModuleCiFailure hook.  For more
                     information on setting the registry keys or
                     using the retryOnModuleCiFailure hook, see the
                     ENOVIA Synchronicity DesignSync
                     Administratrator's Guide.

                     Unmanaged files are recursively checked in,
                     respecting the exclusions defined by the exclude
                     files, creating version 1.2 of the module. Empty
                     directories are not checked in.  Already managed
                     files are ignored. The data in the folder
                     specified (by -path) as the base directory is
                     operated on recursively. Use the -filter option
                     to filter objects from the checkin.  The exclude
                     lists in SyncAdmin are used, as described in the
                     -filter option description.

                     Local data that was checked in is left in the
                     workspace as unlocked copies. This is regardless
                     of whether a different default fetch state has
                     been set. The SyncAdmin option for whether to
                     "Check out read only when not locking" is obeyed.

                     Keyword expansion does not take place for this
```

initial checkin, as a performance optimization. That is regardless of whether the SyncAdmin option to "Enable keyword expansion" is set. Keywords are expanded by subsequent commands, if enabled by the command line -keys option or SyncAdmin setting.

Because keywords are not expanded, the file on disk is left as is. Therefore, the file's date/timestamp is retained. This is regardless of whether the SyncAdmin option to "Retain last-modification timestamps" has been set.

DesignSync client triggers for "ci" are fired off if defined, at the command level and the object level.

If any files fail to checkin, they will be left as is, unmanaged. The files that successfully checkin become part of version 1.2 of the module. After addressing whatever caused the checkin failure, you can use "ci -new" to checkin the previously failed objects. This creates another module version. Thus there is no atomicity, whereas "ci" of module data is atomic.

-[no]comment

-[no]comment "<text>" Specifies whether a text description of the newly created module is stored with the module. The description is used as the comment for branch 1 and is shown in ProjectSync's Data Sheet for the module. The built-in comment used for version 1.1 of the module is "First Version". When the -checkin option is used, the specified comment is also for version 1.2 of the module, which contains the checked in member files.

-nocomment creates the module without an initial description.(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.

If the -checkin option is specified, and you do not specify either -comment or -nocomment, DesignSync prompts you to enter a check-in comment.

-filter

```
-filter <string>    Specify one or more extended glob-style
                    expressions to identify an exact subset of objects
                    for the -checkin option to process.

                    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.

                    If your design contains symbolic links, 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'.

-path

-path <path>

Specifies a workspace path to use as the module's base directory. Specifying a workspace path allows you to immediately begin adding objects to the new module. The module base directory must be contained in a workspace root directory. If there is no workspace root directory already defined, DesignSync will create one if AllowAutoRootCreation is enabled using the setting for DefaultRootPath. For more information on auto-setting root path, see the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide.

Tip: To allow you to more easily differentiate between a module name and a workspace folder name, do not use an initial capital letter to begin the folder name and begin your module names with an initial capital letter.

If no path is specified, or if the value for -path is "" (a null value), DesignSync creates the module without an associated local workspace.

Note: If DesignSync cannot create the workspace directory for any reason, the module is still created on the server.

-report

-report error|brief|normal|verbose

Controls the amount and type of information displayed by the mkmod command during checkin, if the -checkin option is used. Otherwise the -report option is ignored. The information each -report 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 messages, some informational messages, and success/failure count.

normal - includes all information that brief does, and lists all the updated objects, and messages

about objects excluded by filters from the operation. (Default)

verbose - includes all information that normal, does and status messages for the module creation.

The -report option only applies to the output from the -checkin option. If the -checkin option is not specified, then the "-report" option is ignored.

-[no]retry

-[no]retry

Specifies whether, if the module checkin performed with the mkmode operation, fails, DesignSync attempts a retry of the checkin. The mkmod -checkin -retry operation disables multithreading of the checkin operation, which may result in a perform impact.

If mkmod -checkin -retry is performed, DesignSync 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 module is still created, however no object are checked in.

If -retry is explitly specified, but -checkin is not, the -retry option is silently ignored.

RETURN VALUE

This command does not return Tcl values.

SEE ALSO

```
 ci, rmmod, showmods, command defaults
'
```

EXAMPLES

- Example Creating a Module with a Specified Workspace Path
- Example Creating a Module and Checking in Workspace Files

Example Creating a Module with a Specified Workspace Path

```
 This example creates a Chip module in the Chip category and sets the
 workspace path to /home/build/Chip.

 dss> mkmod -comment "Chip module" -path /home/build/Chip \
      sync://chip.ABCo.com:2647/Modules/Chip/Chip
  Creating module Chip on the server...
  Module successfully created on the server.
  Creating module Chip with workspace base directory /home/build/Chip
  Created module instance: Chip%0
  Module creation completed.
```

Example Creating a Module and Checking in Workspace Files

```
 This example creates an Alu module on the server, sets the current
 directory as the module base directory, and checks in the files from
 the current directory to the module.

 stcl> mkmod -path . -checkin -report verbose \
       sync://qelwsun14:30148/Modules/Alu
 Creating module Alu on the server...
 Module successfully created on the server.

 Enter a check-in log message to describe your modifications,
 terminated with single '.' on a line by itself, or 'quit' to abort.
 ------------------------------------------------------------
 : Initial files.
 : .

 Attaching Log message:
 Initial files.


 Beginning Check in operation...

 Creating local instance of the module in the workspace
 Local instance 'Alu%0' of the module successfully created in the workspace.

 Starting checkin of module members
 Checking in: alu.gv    : Success - New version: 1.1 on New branch: 'Trunk'
(1)
```

```
 Checking in: mult8.gv  : Success - New version: 1.1 on New branch: 'Trunk'
(1)
 Checking in: alu.v     : Success - New version: 1.1 on New branch: 'Trunk'
(1)
 Checking in: mult8.v   : Success - New version: 1.1 on New branch: 'Trunk'
(1)

 Finished checkin of module members.

 Starting creating module version on server
 Finished creating module version '1.2' on server.

 Starting creating module manifest in workspace

 Alu%0: Version of module in workspace updated to 1.2
 Starting saving module manifest in workspace
 Finished creating module manifest in workspace.

 Checkin operation finished.

 Module creation completed.
 {Objects succeeded (4)} {}
 stcl>
```

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)
- How Populate Handles Retired Objects (File-based)
- Merging Across Branches (File-based)
- Populate Versus Checkout (File-based)
- Understanding the Output (File-based)
- Forcing, Replacing, and Non-Replacing Modes (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:

- 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

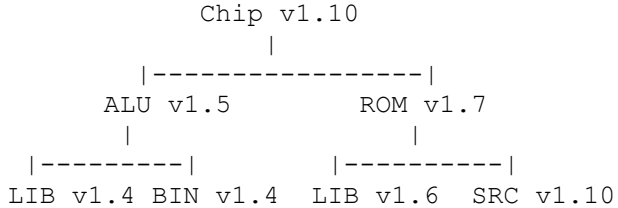
DesignSync may experience a conflict determining what version of LIB (1.3 or 1.5, as referenced in the hierarchy)to populate in the peer directory /home/workspace/LIB.

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:

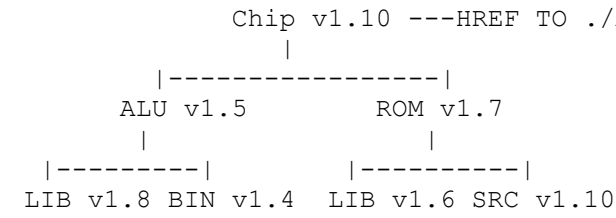
```
 TOP -> ROM@1.23 -relpath ../ROM
     -> COM@1.15 -relpath ../COM
     -> LIB@1.5 -relpath ../LIB
```

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.



Notes:

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

- 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.

- You can have full copies of all the modules in an mcache.
- 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.

- 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:

o 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.

o 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:

- Failure messages.
- Warning messages.
- Version of each module processed as a result of a recursive populate.
- 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.

o 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.

o 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:

    o The current version is set to the fetched version, providing a meaningful branch-point version when you check the object into branch A.
    o The current branch information is undefined.
    o 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:

    o If the object is active in the workspace and retired on the branch version, the workspace version is unchanged.
    o 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.
    o 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:
o 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.
o 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:
 o Failure messages.
 o Warning messages.
 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).

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.

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.

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:
o Unmanaged objects in your work area are not affected by any of

```
  these modes.
o 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)
- -lock -reference (Module-based)
- -lock -reference (File-based)
- -log
- -mcachemode (Module-based)
- -mcachemode (Legacy-based)
- -mcachepaths (Module / Legacy-based)
- -[no]merge (Module-based)
- -merge (File-based)
- -mirror (File-based)
- -modulecontext (Module-based)
- -[no]new (Module-based)
- -overlay
- -path (Module-based)
- -path (Legacy-based)
- -path (File-based)
- -[no]recursive (Module-based)
- -[no]recursive (Legacy-based)
- -[no]recursive (File-based)
- -reference
- -[no]replace (Module-based)
- -[no]replace (File-based)
- -report
- -[no]retain
- -savelocal
- -share
- -target (Legacy-based)
- -trigarg
- -[no]unifystate
- -version (Module-based)
- -version (File / Legacy-based)
- -view (Module-based)
- -xtras (Module-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:
  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.
- 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:
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 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:
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.

-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 <mcache_mode>	Specifies how the populate command fetches 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:
- 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"
- 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:
- 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.
- 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:

o 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.

o 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

o Replaces managed members that belong to both modules with the version belonging to the new module

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.

-[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.

- 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:
o 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.
o 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:
o 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.
o 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 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: o 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). o 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.

showmods

showmods Command

NAME

showmods - Displays the modules available on a server or workspace

DESCRIPTION

- Understanding the Output

This command lists the available modules and external modules within a workspace or on a specified server, and legacy modules on a specified server.

If the command is run within a module directory structure, the showmods command includes the containing modules.

Note: Legacy modules in a workspace do not display with showmods.

The showmods command identifies mcache links to modules within the workspace.

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

The output of the command depends on whether you specify a module on the server or a folder in your workspace.

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.

By default, or if you run the showmods command with the '-report normal' option, the following information is output according to the type of argument being processed:

When the argument is a server module, showmods displays the following information:

Column Titles	Property Names	Description
Name	name	Name of the module. Note: Modules are displayed alphabetically by name.
Owner	owner	User name of the person who created the module.
Path	path	The vault directory containing the contents of the module.

When the argument is a workspace folder, showmods displays the following information:

Column Titles	Property Names	Description
Name	name	Name of the module. Note: Modules are displayed alphabetically by name.
Instance	modinstname	Unique instance name of the module in the workspace.
Base Directory	basedir	Workspace directory containing the contents of the module.
Url	url	Location of the module on the server. Note: For external modules, the URL is sync:///ExternalModule/<external-type>/<external-data>
Selector	selector	Selector used to determine which version to fetch into a workspace. For more information, see the selectors help topic.
Mcache Path	mcachelink	Location of the module cache directory containing the module. Note: If you run the showmods commands with -format text (default) when there are mcaches present in the workspace, the mcache link information: name, instance name and mcache path, display in a separate section below the module information.

If you run the showmods command with '-report brief', it displays the following information.

o For server modules - module path on the server,
  sorted by path. The column title is Module Path. The property name

```
  is path.
o For workspace modules - full directory path for the workspace
  folder, sorted by path. The column title is Unique Address.  The
  property name is address.

If you run the showmods command with '-report verbose', it displays
the information shown with the -normal option and the following
additional information:

For a server module:
Column       Property
Titles       Names          Description
------       -----          ------------
Comment      comment        Comment used when creating a module.

Url          url            Full sync URL of the module.
                            Note: For external modules, the URL is
                    sync:///ExternalModule/<external-type>/<external-data>

Type         type           The type of module being viewed.
                            o standard - a regular module.
                            o legacy module - a legacy module.
                            o external - an external module
                              containing files managed by a different
                              change management system.

For a workspace module:
Column       Property
Titles       Names          Description
------       -----          ------------
Unique
Address      address        Full module address on the client side.
Version      version        The version information for the module
                            version in the workspace.

Top          toplevel       Denotes whether the module is a top-level
                            module (meaning it has no other modules
                            in the workspace containing an href to
                            the reported module). A value of "yes"
                            (or "1") means the module is a
                            top-level. A value of "no" (or "0")
                            means it is not a top-level module.

If you run the showmods command with '-report script', it displays
the same properties as the verbose report in '-format list' mode.

Note: The '-report script' mode is only applicable for workspace arguments.
```

SYNOPSIS

```
showmods [-[no]all] [-format [{text | list}] [-filter <string>]
         [-report {brief | normal | verbose | script}] [-[no]top]
         [<argument>]
```

ARGUMENTS

- Server Path
- Workspace Folder

Server Path

```
<server path>          A server path.  You can use wildcard
                       characters in the path for any piece of the
                       address except the servername and port
                       number. If the path isn't specified, the
                       command will return information for all
                       modules, including legacy modules, on the
                       server.
```

Workspace Folder

```
<workspace folder>     A workspace folder. Specifying a workspace
                       folder displays all modules with a base
                       directory at or below the specified
                       folder. In addition, if the folder is a member
                       of a module whose base directory is above the
                       folder, then that module is also reported.
                       You can use wildcard characters for any part
                       of the workspace folder name.

                       Note: This will never display any legacy
                       modules, since they do not have metadata in
                       the root directory.

Note: If no argument is specified, showmods command uses the current
directory.
```

OPTIONS

- -[no]all
- -format
- -filter
- -report
- -[no]top

-[no]all

```
-[no]all
                       Determines whether to report on the specified
                       workspace folder or on all modules in the
                       workspace using the workspace root directory of
                       the specified workspace folder.
```

-noall reports on the specified workspace folder. (Default)

-all begins at the workspace root directory of the workspace and reports all the modules found in the workspace. Given that the workspace root is usually defined in a directory path one or more levels higher than the argument given to the showmods command, the -all option may list modules that are outside of the directory cone below the argument's path.

This option is only valid when working with workspace folder arguments.

-format

-format <mode>

Determines the format of the output. For information about the information displayed, and the output sort order, see the "Understanding the Output" section.
Valid values are:
o list - Displays a list with the following format:

```
{
 name <name>

}
```

Note: This option replaces the deprecated -report script option.

o text - Display a text table with headers and columns. (Default)

-filter

-filter <string>

Specify one or more extended glob-style expressions to identify an exact subset of module objects on which to operate.

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

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.

For this command, the expressions are matched against the full module URL for each module (sync://host:1234/Modules/MyMod)

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, "... /ProjectA/.../Rom*" matches Rom* modules in a URL that contains "ProjectA", followed by zero or more levels. 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.

-report

-report <mode>

Determines what the output of the command is. For more information on the output of the -report option, see the "Understanding the Output" section.

Valid values are:

- brief - Displays path or workspace location for the specified server or workspace module.
- normal - Displays basic module information for the specified server or workspace module. (Default)
- verbose - Displays extended module information for the specified server or workspace module.
- script - Displays the extended module information for the specified workpace module in a TCL list.

  Note: When -script mode is used, The format mode is automatically set to -list.

-[no]top

-[no]top	Determines whether to display output for all modules or only the top-level modules, modules with no other modules in the workspace containing an href to the reported module. -notop Does not filter the command output and displays all module information. (Defaut) -top Filters the command output to only list top-level modules. This option is only valid when working with workspace folder arguments.

RETURN VALUE

If you run the showmods command with the '-format list' option, it returns a Tcl list. For a description of the output, see the "Understanding the Output" section. If the command fails, it returns a Tcl error.

SEE ALSO

swap show, mkmod, command defaults
,

EXAMPLES

- Example of Showing the Modules on the Server in Text Format
- Example of Showing the Modules on the Server in List Format
- Example Showing the Server Modules Using Verbose Report Mode
- Example Showing a TCL List of Server Modules Using Verbose Report Mode

These examples list the modules available on the DesignSync server sync://srvr1.ABCo.com:2647. The examples show the showmods command running with the two report modes (normal and verbose) and both format options (text and list) to show the different output.

Example of Showing the Modules on the Server in Text Format

The following two examples display to the screen in a user-friendly format the modules available on the DesignSync server sync://srvr1.ABCo.com:2647. The first example specifies a server module. The second example specifies a workspace folder. The workspace folder contains referenced modules, ROM and CPU and mcache module Tools.

```
dss> showmods sync://srvr1.ABCo.com:2647/Modules/Chip

 Beginning showmods operation ...
 Name  Owner    Path
 --------------------------
 Chip  rsmith  Modules/Chip

 Finished showmods operation.

dss> showmods ~/MyModules
 Beginning showmods operation ...

 Name  Instance  Base Directory
     Url                                Selector
 ---------------------------------------------
 ALU   ALU%0     /home/rsmith/MyModules/alu
     sync://srvr1.ABCo.com:2647/Modules/ALU   Trunk:
 Chip  Chip%0    /home/rsmith/MyModules/chip
     sync://srvr1.ABCo.com:2647/Modules/Chip  Trunk:
 Chip  Chip%1    /home/rsmith/MyModules/chipGold
     sync://srvr1.ABCo.com:2647/Modules/Chip  Gold:
 CPU   CPU%1     /home/rsmith/MyModules/chip/CPU
     sync://srvr1.ABCo.com:2647/Modules/CPU   Gold
 ROM   ROM%1     /home/rsmith/MyModules/chip/CPU/ROM
     sync://srvr1.ABCo.com:2647/Modules/ROM   Gold:
 SPC   SPC%0     /home/rsmith/MyModules/spc
     sync://srvr1.ABCo.com:2647/Modules/SPC   Trunk:
 300MM 300MM%0  /home/rsmith/MyModules/300mm
     sync://srv1.ABCo.com:2647/Modules/300MM  Trunk:

 MCACHE LINKS

 Name       Instance     Mcache Path
-------------------------------------------------------------------------
300MM      Chip300MM%0   /home/mcacheDir/300mm/300MM%0


 Finished showmods operation.
```

Example of Showing the Modules on the Server in List Format

The following two examples display to the screen in Tcl format the modules available on the DesignSync server sync://srvr1.ABCo.com:2647. The first example specifies a server module. The second example specifies a workspace folder. The workspace folder contains referenced modules, ROM and CPU.

```
dss> showmods -format list sync://srvr1.ABCo.com:2647/Modules/Chip
  {name Chip owner rsmith path Modules/Chip}

dss> showmods -format list ~/MyModules
  {name ALU modinstname ALU%0 basedir /home/rsmith/MyModules/alu url
  sync://srvr1.ABCo.com:2647/Modules/ALU selector Trunk:} {name Chip
```

```
modinstname Chip%0 basedir /home/rsmith/MyModules/chip url
sync://srvr1.ABCo.com:2647/Modules/Chip selector Trunk:} {name Chip
modinstname Chip%1 basedir /home/rsmith/MyModules/chipGold url
sync://srvr1.ABCo.com:2647/Modules/Chip selector Gold:} {name CPU
modinstname CPU%1 basedir /home/rsmith/MyModules/chip/CPU url
sync://srvr1.ABCo.com:2647/Modules/CPU selector Gold} {name ROM
modinstname ROM%1 basedir /home/rsmith/MyModules/chip/CPU/ROM url
sync://srvr1.ABCo.com:2647/Modules/ROM selector Gold:} {name SPC
modinstname SPC%0 basedir /home/rsmith/MyModules/spc url
sync://srvr1.ABCo.com:2647/Modules/SPC selector Trunk:}
{name 300MM modinstname Chip300MM%0 basedir
/home/rsmith/MyModules/300mm url
sync://srv1.ABCo.com:2647/Modules/300MM selector Trunk: mcachelink
/home/mcacheDir/300mm/300MM%0}
```

Example Showing the Server Modules Using Verbose Report Mode

```
The following two examples display to the screen in a user-friendly
format the modules available on the DesignSync server
sync://srvr1.ABCo.com:2647.  The first example specifies a server
module.  The second example specifies a workspace folder.  The
workspace folder contains referenced modules, ROM and CPU.

dss> showmods -report verbose sync://srvr1.ABCo.com:2647/Modules/Chip
 Beginning showmods operation ...

 Name  Type      Owner    Path
     URL                                    Comment
 ------------------------------------------------------------
 Chip  standard  rsmith  Modules/Chip
     sync://srvr1.ABCo.com:2647/Modules/Chip     Chip design module

 Finished showmods operation.

dss> showmods -report verbose ~/MyModules
 Beginning showmods operation ...

 Name  Instance  Base Directory
     Unique Address
     Url                                Version  Selector
 ------------------------------------------------------------------
 ALU   ALU%0     /home/rsmith/MyModules/alu
     /home/rsmith/MyModules/alu/ALU%0
     sync://srvr1.ABCo.com:2647/Modules/ALU   1.4      Trunk:
 Chip  Chip%0    /home/rsmith/MyModules/chip
     /home/rsmith/MyModules/chip/Chip%0
     sync://srvr1.ABCo.com:2647/Modules/Chip  1.5      Trunk:
 Chip  Chip%1    /home/rsmith/MyModules/chipGold
     /home/rsmith/MyModules/chipGold/Chip%1
     sync://srvr1.ABCo.com:2647/Modules/Chip  1.2.1.1  Gold:
 CPU   CPU%1     /home/rsmith/MyModules/chip/CPU
     /home/rsmith/MyModules/chip/CPU/CPU%1
     sync://srvr1.ABCo.com:2647/Modules/CPU   1.3      Gold
 ROM   ROM%1     /home/rsmith/MyModules/chip/CPU/ROM
```

```
     /home/rsmith/MyModules/chip/CPU/ROM/ROM%1
     sync://srvr1.ABCo.com:2647/Modules/ROM   1.4      Gold:
 SPC   SPC%0     /home/rsmith/MyModules/spc
     /home/rsmith/MyModules/spc/SPC%0
     sync://srvr1.ABCo.com:2647/Modules/SPC   1.4      Trunk:
 300MM 300MM%0  /home/rsmith/MyModules/300mm
     /home/rsmith/MyModules/300mm/300MM%0
     sync://srv1.ABCo.com:2647/Modules/300MM  Trunk:

 MCACHE LINKS

 Name       Instance     Mcache Path
-------------------------------------------------------------------------
300MM      Chip300MM%0   /home/mcacheDir/300mm/300MM%0


 Finished showmods operation.
```

Example Showing a TCL List of Server Modules Using Verbose Report Mode

The following two examples display to the screen in Tcl format the modules available on the DesignSync server sync://srvr1.ABCo.com:2647. The first example specifies a server module. The second example specifies a workspace folder. The workspace folder contains referenced modules, ROM and CPU.

Note: The results of the second command are identical to specifying 'showmods -report script.'

```
dss> showmods -format list -report verbose \
sync://srvr1.ABCo.com:2647/Modules/Chip
  {name Chip type standard owner rsmith path Modules/Chip url
  sync://srvr1.ABCo.com:2647/Modules/Chip comment {}}

dss> showmods -report verbose -format list ~/MyModules
  {name ALU modinstname ALU%0 basedir /home/rsmith/MyModules/alu
  address /home/rsmith/MyModules/alu/ALU%0 url
  sync://srvr1.ABCo.com:2647/Modules/ALU version 1.4 selector Trunk:}
  {name Chip modinstname Chip%0 basedir /home/rsmith/MyModules/chip
  address /home/rsmith/MyModules/chip/Chip%0 url
  sync://srvr1.ABCo.com:2647/Modules/Chip version 1.5 selector
  Trunk:} {name Chip modinstname Chip%1 basedir
  /home/rsmith/MyModules/chipGold address
  /home/rsmith/MyModules/chipGold/Chip%1 url
  sync://srvr1.ABCo.com:2647/Modules/Chip version 1.2.1.1 selector
  Gold:} {name CPU modinstname CPU%1 basedir
  /home/rsmith/MyModules/chip/CPU address
  /home/rsmith/MyModules/chip/CPU/CPU%1 url
  sync://srvr1.ABCo.com:2647/Modules/CPU version 1.3 selector Gold}
  {name ROM modinstname ROM%1 basedir
  /home/rsmith/MyModules/chip/CPU/ROM address
  /home/rsmith/MyModules/chip/CPU/ROM/ROM%1 url
  sync://srvr1.ABCo.com:2647/Modules/ROM version 1.4 selector Gold:}
```

```
{name SPC modinstname SPC%0 basedir /home/rsmith/MyModules/spc
address /home/rsmith/MyModules/spc/SPC%0 url
sync://srvr1.ABCo.com:2647/Modules/SPC version 1.4 selector Trunk:}
{name 300MM modinstname Chip300MM%0 basedir
/home/rsmith/MyModules/300mm address
/home/rsmith/MyModules/300mm/Chip300MM%0 url
sync://srv1.ABCo.com:2647/Modules/300MM version 1.3 selector Trunk:
mcachelink /home/mcacheDir/300mm/300MM%0}
```

hcm showmods Command

NAME

```
hcm showmods        - Displays the modules available on a server
```

DESCRIPTION

- Understanding the Output

```
This command displays the modules available on a given server.

This command should not be used for non-legacy modules.  For
non-legacy modules, use the showmods command with no hcm prefix. The
command syntax for hcm showmods is different than the showmods
command syntax.

Note:
- This command obeys DesignSync's BrowseServer access control; if you
  do not have permission to browse a server, this command displays an
  error.
```

Understanding the Output

```
By default, or if you run the hcm showmods command with the '-report
normal' option, the command displays a list of modules available on
the specified server in a user-friendly format.

The returned table includes the following information:
o NAME              The name of the module.
                    Note:
                    - Modules are displayed alphabetically according to
                      their names.
o OWNER             The user name of the person who created the module.
o VAULT PATH        The vault directory in which the files associated
                    with the module reside.

If you run the hcm showmods command with '-report script', it returns
a Tcl list in the following form:
```

```
   {description <description1> vault_path <vaultPath1>
     name <module_name1> owner <owner1> url <module_URL1>}
    {description <description2> vault_path <vaultPath2>
     name <module_name2> owner <owner2> url <module_URL2>}
    ...
```

The returned information lists the following properties:

o description	A brief text description of the module. This value can be set when you create your module (specified by the -description option of the hcm mkmod command).
o vault_path	The vault directory in which the files associated with the module reside.
o module_name	The name of the module.
o owner	The user name of the person who created the module.
o module_URL	The complete URL of a module including host, port, and vaultPath.

SYNOPSIS

```
hcm showmods  [-report {normal | brief | verbose}]
               -target <server_url>
```

OPTIONS

- -report
- -target

-report

-report	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'.

-target

-target <server_url>	Specifies the URL of the DesignSync server for which you want to see the available modules.

```
Specify the Synchronicity URL as follows:
 sync[s]://<host>[:<port>]
where <host> is the SyncServer on which the
module resides, <port> is the SyncServer port
number.

Note:
- If you do not have access to browse the server
  on which the module is located (set via
  DesignSync's BrowseServer access control), the
  hcm showmods command fails.
```

RETURN VALUE

```
If you run the hcm showmods 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

```
 hcm mkmod
,
```

EXAMPLES

- Example Showing Modules on the Specified Server
- Example Showing Modules in TCL List Form on the Specified Server

```
These examples list the modules available on the DesignSync server
sync://srvr1.ABCo.com:2647. The hcm showmods command is run with
both '-report normal' and '-report script' to show the different
output.

The "Creating and Removing Module Configurations" example in the hcm
Example topic includes the steps for creating the modules that are
presented in this example.
```

Example Showing Modules on the Specified Server

```
This example displays to the screen in a user-friendly format the
modules available on the DesignSync server sync://srvr1.ABCo.com:2647.

dss> hcm showmods -target sync://srvr1.ABCo.com:2647 -report normal

This command displays the following output:
```

```
Modules hosted on server sync://srvr1.ABCo.com:2647

NAME    OWNER   VAULT PATH
--------------------------------
IO      jsmith  /Projects/IO
Mem     jsmith  /Projects/Mem
Top     jsmith  /Projects/Top
```

Example Showing Modules in TCL List Form on the Specified Server

```
This example returns to the screen a Tcl list of the modules available
on the DesignSync server sync://srvr1.ABCo.com:2647.

dss> hcm showmods -target sync://srvr1.ABCo.com:2647 -report script

This command displays the following output:
{description {HCM Module for IO} vault_path /Projects/IO name IO
owner jsmith url sync://srvr1.ABCo.com:2647/Projects/IO}
{description {HCM Module for Mem} vault_path /Projects/Mem name Mem
owner jsmith url sync://srvr1.ABCo.com:2647/Projects/Mem} {description
{Upper-level HCM (Top) Block} vault_path /Projects/Top name Top
owner jsmith url sync://srvr1.ABCo.com:2647/Projects/Top}

The correct usage of the hcm showmods command with the '-report script'
option is as follows:

stcl> catch {hcm showmods -target sync://srvr1.ABCo.com:2647 \
             -report script} module_list
0
stcl> foreach m $module_list {puts $m}
description {HCM Module for IO} vault_path /Projects/IO name IO owner
jsmith url sync://srvr1.ABCo.com:2647/Projects/IO
description {HCM Module for Mem} vault_path /Projects/Mem name Mem
owner jsmith url sync://srvr1.ABCo.com:2647/Projects/Mem
description {Upper-level HCM (Top) Block} vault_path /Projects/Top
name Top owner jsmith url sync://srvr1.ABCo.com:2647/Projects/Top
```

showstatus

showstatus Command

NAME

```
showstatus          - Displays the status of a module in your
                      workspace
```

DESCRIPTION

- Understanding the Output
- Text Formatted Output

- List Formatted Output
- External Module Support
- Legacy Module Output

```
This command lists the status of the hierarchical references of a
module in your local work area as compared to that
module on the server. The main status changes shown by this command
include:
      o Selector changes
      o Added or removed hierarchical references
      o Hierarchical reference conflicts
      o Added, moved, or removed module members
      o Swapped module

Using this command, you can verify that your workspace is up-to-date.
By default the showstatus command does not show the status of module
members. In order to show object status, you must specify the
-objects option.

Notes:
 * When the persistent hrefmode of the workspace is normal,
   showstatus uses the setting of the "Change traversal mode with
   static selector on top level module" option in SyncAdmin (registry
   key "HrefModeChangeWithTopStaticSelector") to determine how the
   module hierarchy should be understood by the command. If the
   setting is enabled and the top-level module is populated with a
   static selector, then the modules populated in the workspace must
   match the expected static versions in order to be considered
   up-to-date.

 * If a module in the workpace is swapped, the showstatus commands
   reports the status of the swapped module as "up-to-date," and
   indicates that the module has been swapped.

 * 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.

 * When working with mcache links in older clients, showstatus may
   report the module as out of date.

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

```
The output can be restricted using the -report option. The -report
brief and -report summary options provide summary information on the
workspace status, while the -report normal option provides more
detail. The -report verbose mode provides additional information
about whether the hierarchical references need updating.
```

Note: You can set the -report normal mode to report on the "needs update" status of hierarchical references with the ShowHrefsNeedCheckinStatus registry key. For more information on setting the registry key, see the DesignSync Administrator's Guide.

The output can be formatted for easy viewing (-format text) or optimized for Tcl processing (-format list). Both formats show similar information, but the forms are very different. The following sections, "Text Formatted Output" and "List Formatted Output," provide specific information about the information returned by the showstatus command.

The module status can change depending on the options specified with the command. For more information, see the output sections below or the options definitions.

Note: For information on legacy module output, see the "Legacy Modules output" section.

Text Formatted Output

The showstatus output is formatted into different sections for ease of use:

- The module information section provides the sync URL of the server modules and the base directory of the workspace module. This section does not appear when -report summary or brief modes are specified.

- The version status section displays the version information for both the workspace and server modules and the unique identifiers (UID) for the modules, if they are different. The -report summary mode displays a single line indicating whether the versions are the same or different. The -report brief mode displays a single line if the version are the same, or the version information if the versions are different.

- The hierarchical reference section compares the hrefs in the local metadata with the appropriate server-side module designated by the selector found in the workspace. The -report summary mode displays a single line notice indicating whether the hrefs are the same or different. The -report brief mode displays a single line if the hrefs are the same, or the list of differences if they are different. This section does not appear at all if the -nohrefs option is specified.

- The href conflicts section compares the expected hrefs (those stored with the parent module) with the actual contents of the workspace. If the URL, selector, or static version doesn't match, the parent module shows a status of "Needs Checkin." For any swapped modules, a line appears for the referenced module providing information about the swapped module version. This section does not appear in -report brief or summary modes or if the -nohrefs option is specified.

- The hrefs missing section highlights any referenced objects which are not present in the workspace. This section does not appear in -report brief or summary modes or if the -nohrefs option is specified.

- The contents section displays the results of comparing the objects in the workspace to the objects in the server. This section does not appear in -report summary mode or if the -noobjects option is specified. If -report brief mode is specified, the command uses the -report brief form of compare.

  Note: When you run showstatus on a DesignSync vault, this is the only section displayed.

- The hierarchical status section recursively reports the status for all hierarchical references in the workspace. This section does not appear if the -norecursive option is specified.

At the end of the showstatus output, the command displays an overall status of the workspace and a recommended action, if an action is needed.

Status	Description
Up-to-date	The workspace and the server versions are identical.
Out-of-date	There is at least one discrepancy between the version present in the workspace and the version on the server.
Unknown	The status could not be determined. This may happen if a server is unreachable.

Recommended Action	Solution
Needs update	The workspace version needs to be repopulated to synchronize it with the server.
Needs checkin	The workspace contains information that has not been captured in a version on the server. Check in your changes, to update the server version.

List Formatted Output

When showstatus is used with the -format list option, the output is returned in the form of a tcl list describing the objects and their status. The properties returned for each object depend on the type of object being examined and the options specified on the command line.

Note: If the object doesn't exist (exist is 0) then most of the values return a null ("") value.

Note: This table lists the properties in alphabetical order, not necessarily the output order of the command.

Property Names	Description
actual	Property list for the object in a workspace. The property list consists of the following properties: o url o selector o version o uid
basedir	Absolute path of the base directory of the object.
conflicts	List of differences between the list of expected hrefs in the workspace and the actual workspace content.
content	Results of running a compare command on the contents of the workspace and the server. This report only appears when the -objects option is specified.
content_status	Status of the objects in the module or DesignSync vault. o Up-to-date - Objects contained in the workspace and server versions are synchronized. o Out-of-date - There are differences in the objects listed in the workspace and server versions. These differences include different object versions and added or removed objects.
exists	Indicates whether the object exists in the workspace. Possible values include: o 1 - object exists in the workspace. o 0 - object does not exist in the workspace.
fullname	Full, unique workspace address.
hier_status	Overall status of the referenced objects. o Up-to-date - Workspace and server versions are synchronized. o Out-of-date - Workspace and server versions are different.
hierarchy	An array of the referenced objects present in the workspace, indexed by href name. o name - Href name. o type - type of object referenced. Possible values include Module, Branch, Selector, External, Release, Alias, Deliverable, or Vault. o url - Server-side vault of the referenced object. o selector - selector for the referenced object. o version - if the reference object is a module, the numeric version ID. For any other object, a null ("") value.

o relpath - relative path of the referenced object.
o status - the status of the referenced object.
Possible values include:
- Up-to-date where the workspace and server metadata for the href instance match.
- Out-of-data where a difference exists among one or more of the values in the hierarchy array. The notes section in the array explains the difference.
- Local-only where an href with the href instance name is present only in the workspace metadata.
- Server-only where an href with the href instance name is present only in the server metadata.
o notes - List of strings describing the discrepancies mentioned in the status value. Possible values include:
- "Relative path changed on server to <relpath>"
- "Url changed on server to <URL>"
- "Selector changed on server to <selector>"
- "Version changed on server to <version>"

Note: In -report brief mode, this property only lists out-of-data hrefs. This property does not appear in -report summary mode or when the -norecursive option is specified.

hrefs — Combined array of hrefs found on the workspace and on the server. This property does not appear in -report summary mode or when the -nohrefs option is specified. The array contains the following fields:
o name - Href name.
o type - type of object referenced. Possible values include Module, Branch, Selector, Release, Alias, Deliverable, or Vault.
o url - Server-side vault of the referenced object.
o selector - selector for the referenced object.
o version - if the reference object is a module, the numeric version ID. For any other object, a null ("") value.
o relpath - relative path of the referenced object.
o status - the status of the referenced object.
Possible values include:
- Up-to-date where the workspace and server metadata for the href instance match.
- Out-of-data where a difference exists among one or more of the values in the hierarchy array. The notes section in the array explains the difference.
- Local-only where an href with the href instance name is present only in the workspace metadata.
- Server-only where an href with the href instance name is present only in the server metadata.
o notes - List of strings describing the discrepancies mentioned in the status value. Possible values include:
- "Relative path changed on server to <relpath>"
- "Url changed on server to <URL>"

```
                - "Selector changed on server to <selector>"
                - "Version changed on server to <version>"

href_status     Status of the hierarchical reference metadata in the
                workspace.
                o Up-to-date - Workspace and server versions are
                  synchronized.
                o Out-of-date - Workspace and server versions are
                  different.

legacy_status   Legacy show status for legacy configurations.  For
                more information, see the "Legacy module output"
                section.

missing         Names of the hrefs expected in the workspace that
                are not present in the workspace. This property does
                not appear in -report brief or summary modes.

                Note: Missing hrefs do not automatically mean the
                workspace and server are out of sync.  An href may
                have been filtered out during the workspace populate,
                or removed from the workspace manually.

modinstname     Workspace module instance name.

needs_checkin   Status of the object in the workspace:
                0 Indicates that the object is up-to-date and does
                  not require a checkin.
                1 Indicates the object is locally modified and does
                  require a checkin.

needs_update   Status of the object in the workspace:
               0 Indicates that the object is up-to-date and does not
                 need to updated by the server.

               1 Indicates that the object is out-of-date and does
                 need to be updated by the server.

server          Property list for the object on a server which which
                the workspace object is being compared. These are
                the properties of the module or DesignSync vault to
                which the workspace selector resolves.
                o url
                o selector
                o version
                o uid

status          Status of the workspace object.
                o Up-to-date where the workspace and server versions
                  are synchronized.
                o Out-of-date where the workspace and server versions
                  are different.
                o Unknown where the status of the object could not be
                  determined. This might occur if a server is
                  unavailable.

swap_conflict   Shows the properties of the swapped modules, including
```

	the href name, selector and version for each swapped module version.
swapped	Indicates whether the href is swapped. o 1 - object is swapped. o 0 - object is not swapped.
type	Workspace object type. Possible values include: o standard - module o legacy module o external - external module Note: External modules are always considered "up-to-date." o DS vault o deliverable - IP Gear deliverable.
uid	UID of the module in the workspace.
unknown	Status of the object hierarchy in the external module. 0 Indicates that the external module hierarchy status is known. 1 Indicates that the external module hierarchy status is not known.
version_status	Status of the workspace version. o Up-to-date - Workspace version and server version associated with the workspace by the designated select are synchronized. o Out-of-date - Workspace version and server version associated with the workspace by the designated selector are different.

External Module Support

DesignSync supports showing the status of an external module to determine whether the objects are current or out of date. After an external module has been populated, the showstatus command can be available to query the status of the external module members and return the results.

For information on populating an external module, see the populate command. For information on configuring showstatus for external modules, see the DesignSync Administrator's Guide.

Legacy Module Output

The legacy module output is unchanged from previous versions.

The showstatus command, by default, displays the following information:
o Configuration: Identifies the configuration for which the status

is shown. This value is a Synchronicity URL.

o Base Directory: Identifies the local file system directory in which the configuration resides. Each module in a hierarchy has its own base directory.

o Information about each hierarchical reference:

- STATUS The status of the hierarchical reference of the configuration in the work area as compared to the server. Possible values are:

  Local Only — Indicates the hierarchical reference exists only in the local work area.

  Out-of-date — Indicates that the hierarchical reference in the local work area does not match the hierarchical reference on the server; for example, an alias on the server may have changed to refer to a new release or the relative path of the hierarchical reference on the server may have changed.
  Displays a table of conflicts if conflicts exist between the submodule configuration that a parent module expects to find in the workspace and the submodule configuration that actually exists there. Such conflicts can be caused by: a relative path that contains a configuration different from the one that the parent configuration expects; a relative path that contains no configuration; or a relative path that doesn't exist.

  Server Only — Indicates the hierarchical reference was added on the server.

  Unknown — Indicates that status of the hierarchical reference cannot be determined. This status is displayed only if you specify a recursive showstatus operation. For example, if you specify 'showstatus -recursive' and the server on which a referenced configuration resides is unavailable, the showstatus operation lists the status for that configuration as Unknown.

  Up-to-date — Indicates that the hierarchical reference in the local work area matches the hierarchical reference on the server.

- HREF Identifies the submodule to which the configuration of the upper-level module is connected. This value is a Synchronicity URL.

- RELATIVE PATH Identifies the path from the upper-level module to the submodule.

o Configuration status
Indicates the status of the configuration. (Displayed only when you specify the -recursive option.)

o Summary Indicates the overall status of the configuration. By default (or if you specify the '-report normal' option), this value is a summary of the status of the configuration's

```
                  hierarchical references. Note: If you specify
                  the -objects option, this value represents the
                  status of the objects contained in the
                  configuration in combination with the status of
                  its hierarchical references.
                  Possible values are:
   - Local Only   Indicates the configuration exists only in
                  the local work area.
   - Out-of-date  Indicates that the configuration in
                  the local work area does not match the
                  configuration on the server.
   - Server Only  Indicates the configuration was added
                  on the server.
   - Unknown      Indicates that status of the configuration
                  cannot be determined. This status is
                  displayed, for example, if the server on which
                  the configuration resides is unavailable.
   - Up-to-date    Indicates that the configuration in
                  the local work area matches the hierarchical
                  reference on the server.

To show the status of the objects contained in your work area
configuration (as compared to objects in the configuration on the
server), you can use the showstatus command with the -objects
option. Output from the command first shows the status of the
configuration's hierarchical references (as described above) and then
shows the status of its objects. When '-objects' is specified, the
value for configuration status reflects the status of its objects in
combination with the status of its hierarchical references.

Information for each object includes:

o Workspace Version       Identifies the version of the object in
                          the work area configuration. "Unmanaged"
                          indicates the object is not managed by
                          DesignSync; "Unknown" indicates that the
                          status cannot be determined. (For
                          example, the command might display
                          "Unknown" for an object if the server is
                          not available.) If no information is
                          displayed, it indicates that the version
                          is absent from the work area.

o Configuration Version   Identifies the version of the object in the
                          configuration on the SyncServer.

o Object Name             Identifies the name of the object for
                          which status information is given.

If you use the showstatus command with '-format list' option, it
returns a Tcl list in the following form:

target <module_URL>
relpath <relative_path>
[notes {
         "Old aliased release: <release_name>" |
         "New aliased release: <release_name>" |
```

```
          "Relative path changed to '<path>'" |
          "Cannot determine current value of alias on server." |
          <miscellaneous other information>
        }
]
status <Local Only | Out-of-date | Server Only | Unknown | Up-to-date>
hierstatus <Local Only | Out-of-date | Server Only | Unknown | Up-to-date>
[hrefs {{<submodule_status>} {...}}] ...
```

The returned information includes the following:

o target — The URL of the module configuration.

o relpath — The relative path from the base directory of the upper-level module configuration (Parent) to the submodule configuration (Target). This path is used when you fetch (populate) the module into your work area.

o status — The status of the configuration in your work area (as compared to the configuration on the server). Note: This status reflects the status of the configuration's hierarchical references. If the -objects option is specified, status reflects the status of configuration's hierarchical references and objects.

Possible values are:

- Local Only — Indicates the hierarchical reference exists only in the local work area.
- Out-of-date — Indicates that the hierarchical reference in the local work area does not match the hierarchical reference on the server; for example, an alias on the server could have changed to refer to a new release or the relative path of the hierarchical reference on the server could have changed.
- Server Only — Indicates the hierarchical reference was added on the server.
- Unknown — Indicates that status of the hierarchical reference cannot be determined. This status is displayed, for example, if the server on which the configuration resides is unavailable.
- Up-to-date — Indicates that the hierarchical reference in the local work area matches the hierarchical reference on the server.

o hierstatus — Indicates the overall status of the configuration. By default (or if you specify the '-report normal' option), this value is a summary of the status of the configuration's hierarchical references. If you specify the -objects option, this value represents the status of the objects contained in the configuration as well as the status of its hierarchical references. If you specify the -recursive option, the value indicates the status of the entire configuration hierarchy. Possible values are:

```
    - Out-of-date   Indicates that the configuration in
                    the local work area does not match the
                    configuration on the server.
    - Unknown       Indicates that status of the configuration
                    cannot be determined. This status is
                    displayed, for example, if the server on which
                    the configuration resides is unavailable.
    - Up-to-date    Indicates that the configuration in
                    the local work area matches the hierarchical
                    reference on the server.
o hrefs             A list of property lists, one for each of the
                    configuration's hierarchical references.
                    (Displayed only if the configuration has
                    hierarchical references.)

Note:
o Your output will include 'notes' if an alias in your work area is
  out of date with respect to the server. For example, if DRAM@Silver
  initially references the DRAM@R1 configuration and the alias is
  changed such that DRAM@Silver now references the DRAM@R2
  configuration, your output would include the following notes:

  notes {{Old aliased release: R1} {New aliased release: R2}}

o When you run the showstatus command recursively, your output will
  include 'hrefs' status for each submodule containing hierarchical
  references. The <submodule_status> is a Tcl list of the module
  status information.

To show the status of the objects contained in your work area
configuration, use the showstatus command with the -objects and the
'-format list' options. Output from the command lists the status of
the configuration and its hierarchical references (as described
above). In addition, the output includes a Tcl list (content) that
describes the status of each of the objects contained in the
configuration.

target <module_URL>
relpath <relative_path>
[notes {{Old aliased release: <release_name>}
        {New aliased release: <release_name>}}]
status <Local Only | Server Only | Up-to-date | Unknown | Out-of-date>
content
       {
         path1 <path>
         path2 <URL>
         type <folder | file>
         objects
             {
                {
                   name <object_name>
                   type <file | folder>
                   objects {object_list}
                   name <file_name> type file
                   props1
                       {
                         state <absent | modified | present | reference
```

```
                                | unknown | unmanaged>
                          version <version_number>
                        }
                    props2
                        {
                          state <absent | modified | present | reference
                                | unknown | unmanaged>
                          version <version_number>
                        }
                }
            }
        }

   hierstatus <Up-to-date | Out-of-date | Unknown>
   [hrefs {{<submodule_status>} {...}}] ...

o content           Lists the objects in the configuration
                    and reports the status of each. (Displayed
                    only if you specify the -objects option with
                    the showstatus command.)
o path1             The path to the work area directory containing
                    the configuration.
o path2             The URL of the configuration on the server.
o type              The type of object contained in the work area
                    path (folder or file).
o objects           A list of objects and their properties.
                    (Displayed only when type is
                    folder.) For each object, the following
                    information is provided:
   - name           The name of the object
   - type           The object's type (folder or file)
   - props1 {...} props2 {...}
                    Properties of the objects in the configuration.
                    (Displayed only when object type is file.)
                    Properties are:
                    o version - The version number of the object.
                      (In certain cases, this property may not be shown.)
                    o state - The status of the object in the path
                      (your work area or configuration on the server.)
                      Possible values are:
                       - absent - Indicates that the object is not
                         present on this path (work area or
                         configuration on the server).
                       - modified - Indicates that the object has been
                         locally modified.
                       - present - Indicates that the object is
                         present on this path. (The version that is
                         present is reported in the version property.)
                       - reference - Indicates that the object is a
                         referenced object.
                       - unknown - Indicates that the object exists
                         in the work area but the fetched version
                         is unknown. This state is most commonly
                         reported when an object has been removed from
                         the workspace (with the rmfile command)
                         and then recreated.
```

SYNOPSIS

```
showstatus  [-format <type>] [-[no]hrefs] [-[no]objects]
            [-[no]recursive] [-releases]
            [-report {brief | normal | verbose | summary | script}]
            [-xtras <xtras>] <argument>
```

ARGUMENTS

- Workspace Module
- Legacy Module Base Directory
- External Module Instance

Workspace Module

<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.

Legacy Module Base Directory

<legacy module base directory>	Specifies the workspace legacy module base directory. It is compared against the corresponding server folder. Note: You cannot run the showstatus command against a module sub-folder. The command must be run against the top level module directory.

External Module Instance

<external_mod>	Specifies the external module instance. The external module must be populated into the workspace.

OPTIONS

- -format
- -[no]hrefs
- -[no]objects
- -[no]recursive
- -releases
- -report
- -xtras

-format

-format <type>	Determines the format of the output.
	Valid values are:
	o list - Displays a list with the following format:
	{
	name <name>
	}
	For a list of properties displayed, see the "Understanding the Output" section above.
	o text - Display a text table with headers and columns. (Default) Objects are shown in alphabetical order.

-[no]hrefs

-[no]hrefs	Determines whether to follow the hierarchical references to determine if their status is current. -nohrefs does not trace the hierarchical references. This eliminates the time and server load that might be required to follow the hierarchical trail. (Default) -hrefs traces the hierarchical references to determine if the reported status is current.

-[no]objects

-[no]objects	Indicates whether command should run the compare command to compare the status of each object in the workspace with the corresponding object version on the server. -noobjects skips the object comparison. (Default) -objects checks the status of the workspace objects.

-[no]recursive

-[no]recursive	Indicates whether the command should return the status for the specified module, or the specified module and all referenced modules. -norecursive displays the status for the

specified module only. (Default)

-recursive displays the status for the specified modules and all referenced modules and identifies why particular hierarchical references are not recursed.

Notes: If you run the showstatus command with the '-format list' option, the showstatus command captures all errors encountered in the hierarchy and displays a message containing all the error messages.

-releases

-releases	Indicates that the showstatus command should run recursively against a legacy module releases. (Legacy modules only.) Note: If this option is not supplied, the status of hierarchical references to releases is always listed as up-to-date.

-report

-report <mode>	Specifies the type of status information to be displayed. Valid values are: o brief - Displays a summary for all data and detailed data for any items that are out of sync. For a description of the status information, see "Understanding the Output". o normal - Displays the status of the hierarchical references (and optionally, file status) for the module. (Default) Also displays a table of conflicts if conflicts exist between the expected submodule and the actual submodule. For a description of the status information, see "Understanding the Output". Note: You can set the -report normal mode to report on the "needs Â update" status of hierarchical references with the ShowHrefsNeedCheckinStatus registry key. For more information on setting the registry key, see the DesignSync Administrator's Guide. o verbose - Displays the status of the hierarchical references and additional

information about whether the hierarchical references need updating (and optionally, file status) for the module. Displays a table of conflicts if conflicts exist between the expected submodule and the actual submodule, and additional information. For a description of the status information, see "Understanding the Output".

- summary - Displays the target and base directory of the module, the status of each module, and the overall status of the module in the workspace. Also displays a table of conflicts if conflicts exist between the expected submodule and the actual submodule. For a description of the status information, see "Understanding the Output".

- script - Returns a Tcl list of config_name/property_list pairs. This is identical to using running showstatus with -report verbose -format list.

-xtras

-xtras <xtras> 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

If you run the showstatus command with the '-format list' option, it returns a Tcl list. For a complete description of the output, see the "Understanding the Output" section.

SEE ALSO

addhref, rmhref, compare, ls, swap show, edithrefs, command defaults
,

EXAMPLES

- Module Hierarchy for Module Examples
- Example Showing Module Href Status Where Hrefs are Current
- Example Showing Module Href Status Where Hrefs are Outdated

- Example Showing Outdated Module Href Status in List Format
- Example Showing Legacy showstatus Command Formats
- Example of using showstatus on a legacy module

Module Hierarchy for Module Examples

```
All of the modules example assume the following hierarchy in your
work area.
Top                    stored in ~/MyModules/Chip
   CPU;Trunk:Gold      stored in ~/MyModules/Chip/CPU
     ALU               stored in ~/MyModules/Chip/CPU/ALU
```

Example Showing Module Href Status Where Hrefs are Current

```
This example lists the status of the hierarchical references in your
local work area as compared to the server.

 dss> showstatus -recursive Chip%0
  Beginning showstatus operation ...

  Status of module Chip%0 ...

  Chip%0: url - sync://srv2.ABCo.com:2647/Modules/Chip;Trunk:
  Chip%0: base directory - /home/rsmith/MyModules/chip

  Chip%0: Workspace version 1.7
  Chip%0: Server version    1.7
  Chip%0: Version is Up-to-date

  Href Name  Status      Url                                Selector \
           Version  Relative Path
  ------------------------------------------------------------------
  CPU     Up-to-date  sync://srv2.ABCo.com:2647/Modules/CPU Trunk:Gold
            1.3      CPU
  ROM     Up-to-date  sync://srv2.ABCo.com:2647/Modules/ROM Trunk:
            1.2      /ROM

  Chip%0: No hierarchical reference conflicts found.

  Chip%0: Hrefs are Up-to-date

  Status of module CPU%1 ...

  CPU%1: url - sync://A/Modules/CPU;Trunk:Gold
  CPU%1: base directory - /home/rsmith/MyModules/chip/CPU

  CPU%1: Workspace version 1.3
  CPU%1: Server version    1.3
  CPU%1: Version is Up-to-date

  Href Name  Status      Url                                Selector
          Version  Relative Path
   ------------------------------------------------------------
```

```
  ALU        Up-to-date  sync://srv2.ABCo.com:2647/Modules/ALU  Trunk:
          1.2      ALU

  CPU%1: No hierarchical reference conflicts found.

  CPU%1: Hrefs are Up-to-date

  Status of module ALU%2 ...

  ALU%2: url - sync://srv2.ABCo.com:2647/Modules/ALU;1.2
  ALU%2: base directory - /home/rsmith/MyModules/chip/CPU/ALU

  ALU%2: Workspace version 1.2
  ALU%2: Server version    1.2
  ALU%2: Version is Up-to-date

  ALU%2: No hierarchical references.

  ALU%2: Module hierarchy is Up-to-date.

  ALU%2: Module is Up-to-date.

  CPU%1: Module hierarchy is Up-to-date.

  CPU%1: Module is Up-to-date.

  Chip%0: Module hierarchy is Up-to-date.

  Chip%0: Module is Up-to-date.

  Finished showstatus operation.
```

Example Showing Module Href Status Where Hrefs are Outdated

```
This example shows output of an showstatus operation where a
hierarchical references in the module is out of date.

stcl> showstatus -objects Chip%0

 Beginning showstatus operation ...

 Status of module Chip%0 ...

 Chip%0: url - sync://srv2.ABCo.com:2647/Modules/Chip;Trunk:
 Chip%0: base directory - /home/rsmith/MyModules/chip

 Chip%0: Workspace version 1.5
 Chip%0: Server version    1.6
 Chip%0: Version is Out-of-date

 Href Name  Status       Url
          Selector    Version  Relative Path
-----------------------------------------------
```

```
 CPU        Up-to-date   sync://srv2.ABCo.com:2647/Modules/CPU
          Trunk:Gold  1.3      CPU
 ROM        Server Only  sync://srv2.ABCo.com:2647/Modules/ROM
          Trunk:      1.2      /ROM


 Chip%0: No hierarchical reference conflicts found.

 Chip%0: Hrefs are Out-of-date

 Workspace       Configuration          Status             Object
 Version         Version                                   Name
 ---------       -------------          ------             ------
 1.1              1.1                    Identical          chip.c
 1.1              1.1                    Identical          chip.doc
 1.1              1.1                    Identical          chip.h



Chip%0: Module is Out-of-date.
Chip%0: Needs update.

Finished showstatus operation.
```

Example Showing Outdated Module Href Status in List Format

```
This example shows the same data as in the previous example, an out
of data hierarchical reference, but the output is presented in list
format.

stcl> showstatus -format list -objects Chip%0
 href_status Out-of-date exists 1 basedir /home/rsmith/MyModules/chip
 type standard needs_checkin 0 content {path1
 /home/rsmith/MyModules/chip/Chip%0 path2
 sync://srv2.ABCo.com:2647/Modules/Chip@Trunk: type folder props1
 {type module url sync://srv2.ABCo.com:2647/Modules/Chip version 1.5
 relpath {} basedir /home/rsmith/MyModules/chip} props2 {type module
 url sync://srv2.ABCo.com:2647/Modules/Chip version 1.6 relpath {}
 modulepath {}} objects {{name chip.doc type file state identical
 props1 {state present version 1.1} props2 {state present version
 1.1}} {name chip.c type file state identical props1 {state present
 version 1.1} props2 {state present version 1.1}} {name chip.h type
 file state identical props1 {state present version 1.1} props2
 {state present version 1.1}}}} conflicts {} content_status
 Up-to-date hierarchy {} server {selector Trunk: uid
 1ba413d31cfbd405591dba00f2ef564a version 1.6 url
 sync://srv2.ABCo.com:2647/Modules/Chip} hrefs {{status Up-to-date
 relpath CPU selector Trunk:Gold name CPU version 1.3 type Module
 basedir /home/rsmith/MyModules/chip/CPU url
 sync://srv2.ABCo.com:2647/Modules/CPU modinstname CPU%1} {status
 {Server Only} relpath /ROM selector Trunk: name ROM version 1.2 url
 sync://srv2.ABCo.com:2647/Modules/ROM}} needs_update 1
 version_status Out-of-date missing {} actual {version_ci {} selector
```

```
 Trunk: uid 1ba413d31cfbd405591dba00f2ef564a version 1.5 url
 sync://srv2.ABCo.com:2647/Modules/Chip} hier_status Up-to-date
 fullname /home/rsmith/MyModules/chip/Chip%0 status Out-of-date
 modinstname Chip%0 #
```

Example Showing Legacy showstatus Command Formats

o Show the status of hierarchical references of a module configuration hierarchy in the work area as compared to the server:

```
dssc> showstatus -recursive <ModInstance>

For example:
dssc> showstatus -recursive Chip%0
```

o Show the status of hierarchical references and objects contained in a module configuration hierarchy in the work area as compared to the server:

```
dssc> showstatus -recursive -objects <ModInstance>
```

o Show the status of each configuration in the module configuration hierarchy, followed by a summary of the overall status of the hierarchy. (Note: Because '-files' is specified, each configuration's status represents the status of the configuration's hierarchical references and its objects.)

```
dssc> showstatus -recursive -objects -report summary \
     <ModInstance>
```

Example of using showstatus on a legacy module

This example lists the status of the hierarchical references in your local work area as compared to the server. It assumes the following data hierarchy in your work area.

```
 Top             kept in directory Designs/Top
   IO@TEST      kept in directory Designs/Top/IO
   Mem@DEV      kept in directory Designs/Top/Mem

 dss> showstatus -recursive Chip%0
```

This command displays the following output:

```
Target:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top

STATUS      HREF                                          RELATIVE PATH
-------------------------------------------------------------------------
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/IO@TEST  IO
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Mem@DEV  Mem
```

```
Configuration status: Up-to-date

============================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/IO@TEST
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/IO

No local or remote hierarchical references found for configuration.

Configuration status: Up-to-date

============================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/Mem

No local or remote hierarchical references found for configuration.

Configuration status: Up-to-date

================================================================

Status of all visited configurations.
STATUS      TARGET                                          PATH
----------------------------------------------------------------
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
/home/jsmith/Designs/Top/Mem
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/IO@TEST
/home/jsmith/Designs/Top/IO
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Top
/home/jsmith/Designs/Top

Summary: Up-to-date
```

tag

tag Command

NAME

```
tag                 - Assigns a tag to a version or a branch
```

DESCRIPTION

- Working with Tags
- Branch Tags Versus Version Tags
- Tagging Modules (Module-based)
- Module Snapshots (Module-based)
- Tag Name Syntax (Module-based)

- Determining the Objects to be Tagged (Module-based)
- Using Tags on Module Versions (Module-based)
- Interaction with Legacy Modules (Legacy-based)
- Tagging Files-Based DesignSync Objects (File-based)
- Tag Name Syntax (File-based)
- Determining the Objects to be Tagged (File-based)
- Interaction with Objects from a Mirror (File-based)

```
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:

 o A module, specified explicitly as a server module URL, in this format:
   sync://<machine>:<port>/Modules/<category>/<module_name>;<selector>

o  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.

 o 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:

 o 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.

 o 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.

 o 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.

 o 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:

o 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.

o 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
- -[no]comment (Module-based)
- -[no]delete (Module-based)
- -[no]delete (File-based)
- -exclude (Module-based)

- -exclude (File-based)
- -filter (Module-based)
- -[no]modified (File-based)
- -modulecontext (Module-based)
- -[im]mutable (Module-based)
- -[no]recursive (Module-based)
- -[no]recursive (Legacy-based)
- -[no]recursive (File-based)
- -[no]replace (Module-based)
- -[no]replace (File-based)
- -report
- -[no]selected
- -trigarg
- -version (Module-based)
- -version (File-based)
- -warn
- -xtras (Module-based)
- --

-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        Specifies whether to tag the specified
"<text>"            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:

- When -replace is used on a module snapshot, it replaces the tag on the specified module members in the snapshot.
- 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:
```

```
                    o A UNIX 'rm' command was used. (Note: 'rm'
                      is not recommended; use 'rmvault' instead.)
                    o 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)
- Example of Tagging All Files Matching a Wildcarded String (File-based)
- Example of Tagging a Specified Version of Files (File-based)
- Example Showing Tagging Modified File in the Workspace (File-based)
- Example of Tagging Locked Files (File-based)
- Example of Using Exclude to Restrict Which Files are Tagged (File-based)
- Example of Tagging a Branch (File-based)
- Example of Deleting a Tag (File-based)
- Example of Adding a Tag to a Branch or Version (File-based)
- Examples of Tagging an Object on the Server (File-based)

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

duplicatews

duplicatews Command

NAME

```
duplicatews          - creates or updates a copy of a workspace.
```

DESCRIPTION

- Creating a Duplicated Workspace
- Updating a Duplicated Workspace
- Replacing an Existing Duplicated Workspace

The duplicatews command creates, modifies or updates a workspace copy (or duplication) from another workspace. By creating an exact copy of the source workspace in the target directory, including metadata files, the command reduces the initial populate time of the new workspace and simplifies the process of creating the workspace, since the user only needs to know where the workspace being duplicated is located, not the information about the data in the workspace.

Note: This command cannot be executed if cache reference count is enabled. If the workspace being duplicated has reference counting enabled, you must change the options to disable cache reference counting. For more information on enabled and disabling cache reference, see the DesignSync Administrator's Guide.

Important: The duplicatews command can only be run on a linux system and the source and target directories must also be on a linux system.

This command supports the access control system.

Creating a Duplicated Workspace

In order to duplicate a workspace, designate the source workspace as a workspace that be duplicated.

To designate the workspace as a duplicable workspace, create a file, ".clone" in the .SYNC directory for the workspace. The .clone file can be empty; the contents are not used by the system. The workspace .SYNC folder must be writable by the users or group members to perform the clone operations.

Use the duplicatews command to create a workspace copy using the source workspace.

When creating an initial duplicated workspace, these are the relevant command options:
duplicatews [-dir <target>] [-dryrun] -refws <source>
[-[no]validaterefws]

See the Options section for descriptions of the options.

Note: If the -dir <target> isn't specified, the command uses the current directory (.).

Updating a Duplicated Workspace

Once the workspace has been created, it can be updated using the -update option, which acts as an incremental populate, updating the

```
workspace with any changes from the source workspace. Since an update
always uses the original source workspace, you do not specify a
reference (source) workspace. The duplicated workspace much be on the
same branch as the source workspace and use the same selector.

Note: Any objects, including referenced submodules that are present
in the duplicated workspace, but are not in the source workspace are
removed and any new objects in the source workspace are added to the
duplicated workspace.

When updating a workspace, these are the relevant command options:
duplicatews [-dir <target>] [-dryrun] -update [-[no]validaterefws]

See the Options section for descriptions of the options.

Note: If the -dir <target> isn't specified, the command uses the
current directory (.).
```

Replacing an Existing Duplicated Workspace

```
Running duplicatews -replace replaces the existing workspace with
a new one, which can be a different module configuration, or
module. The operation replaces all objects in the target workspace with
the source workspace objects.

When replacing a workspace, these are the relevant command options:
duplicatews [-dir <target>] [-dryrun] [-refws <source>]
[-[no]validaterefws] [-replace]

See the Options section for descriptions of the options.

Note: If the -dir <target> isn't specified, the command uses the
current directory (.). If the -refws <source> option isn't specified,
the command uses the source stored in the metadata from the last
duplicatews operation.
```

SYNOPSIS

```
duplicatews [-dir <target>] [-dryrun] [-refws <source>] [-replace]
[-status] [-update] [-[no]validaterefws]
```

OPTIONS

- -dir
- -dryrun
- -refws
- -report
- -status
- -update

- -validate

-dir

```
 [-dir <target>]  Valid path to the target workspace. The path can be
                  specified as an absolute or relative path. If no
                 -dir option is specified, the default is the current
                 directory (.).
```

-dryrun

```
 -dryrun          Performs a test of the command with the options
                   specified to verify that the operation will succeed
                  before performing the operation. This option can be
                  run with any of the report modes.
```

-refws

```
 -refws           Valid path to the source workspace to clone. The path can
be
                  specified as an absolute or relative path. The path
                 must be the workspace root. The workspace root must
                 contain .clone file in the root level .SYNC directory for
                 the workspace.

                  This option is required when creating an initial
                 duplicate workspace. This option is not allowed when
                 updating a duplicate workspace.
```

-report

```
 -report error|     Controls the amount and type of information
  brief|normal|     displayed by command.  The information each
  verbose           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, some
                    informational messages, and success/failure
                    count.

                    normal - includes all information from brief and
```

lists all the objects created, modified or removed from the new workspace. (Default)

verbose - provides full status for each object processed, even if the object is not updated by the operation.

-status

-status Returns target and source workspace information. The information returned depends on selected the report mode.

When used with -report normal (default) or report brief, this option reports the status of the target workspace.

When used with -report error, this option includes additional information about any blocking conditions, such as locked or modified files in the workspace.

When used with -report verbose, this option includes all the information from -report normal and error and includes status and blocking conditions present in the source workspace.

-update

-update Updates an existing duplicated workspace from the original source workspace. For more information, see Updating a Duplicated Workspace.

This option is mutually exclusive with the -refws option.

-validate

-[no]validaterefws Determines whether to perform a pre-operation check for locked files, locally modified objects and swapped modules in the workspace.

-validaterefws performs the precheck and does not start the operation if there are locked files, locally modified objects, or swapped modules in the workspace. This adds some preprocessing time, but insures the integrity of the resulting workspace.

-novalidaterefws does not validate whether objects in the workspace are locked, swapped, or

```
modified. This can affect the integrity of the new
workspace, but the operation may complete more
quickly without the precheck. For update or
replace operation run with the -novalidaterefws,
any workspace locks are removed and any modified
objects in the target workspace are overrwitten by
the source workspace.

This value overrides the value of the workspace
registry key, ValidateReferenceWorkspace. By
default, ValidateReferenceWorkspace is enabled,
meaning that if nothing is changed in the system,
-validaterefws is enabled.
```

RETURN VALUE

```
The command does not have a TCL return value. If the command succeeds
it creates a new workspace in the target directory.  If the command
fails, it returns an appropriate error message.
```

SEE ALSO

```
access, eda createrefws, mkmod, sda mk
```

EXAMPLES

- Example showing the status of a workspace in report verbose mode

Example showing the status of a workspace in report verbose mode

```
stcl> duplicatews -status -report verbose -novalidate
 ------------------------------------
 Duplicate workspace
 ------------------------------------
 path: /home/rsmith/MyModules/
 Module name: CPU
 base directory: /home/rsmith/MyModules/cpu
 url: sync://srv2.ABCo.com:2647/Modules/CPU
 selector: Trunk:
 version: 1.3
 last duplicatews operation: Mon Jun 10 12:25:02 EDT 2019
 Workspace /home/rsmith/MyModules has locked files

 ------------------------------------
 Reference workspace
 ------------------------------------
 path: /home/syncadmin/CommonModules
 Module name: CPU
 base directory: /home/syncadmin/CommonModules/cpu
```

```
 url: sync://srv2.ABCo.com:2647/Modules/CPU
 selector: Trunk:
 version: 1.3
stcl>
```

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
```

freezemod

freezemod Command

NAME

```
freezemod           - Sets access controls on a module to prevent changes
```

DESCRIPTION

```
This command modifies the access controls for the specified module so
that no changes can be made to the module.  This reduces the need for
a merge when performing complex module operations, such as modifying
hierarchical references in a batch mode or moving the module to a different
server.

Note: It is not necessary to freeze a module when performing a backup
operation.

When the changes are complete, unfreezemod releases the module for
modifications.

Note: If you have moved the module to a different server, you may
wish to either delete the old module or retain it in a frozen state
to prevent users from making modifications in the wrong location.

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

SYNOPSIS

```
freezemod <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
```

RETURN VALUE

```
Does not return any TCL values.
```

SEE ALSO

```
unfreezemod, mvmod, importmod, exportmod, edithrefs
```

EXAMPLES

- Freezing a module

Freezing a module

```
This example shows freezing a module.
dss> freezemod sync://serv.ABCo.com:2647/Modules/Chips/chip-nx1
 sync://qelwsun14:30126/Modules/Chips/chip-nx1 : Module is frozen.
```

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
```

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.
```

lock

lock Command

NAME

```
lock                - Creates a server-side lock on a module branch
```

DESCRIPTION

```
This command locks a module branch. Locking the branch is useful when
creating new branches; creating, moving, or deleting tags; or adding
```

or removing hierarchical references. By locking the module version, you insure that no one else can alter the data while you make your changes.

If any module objects are already locked on the requested branch, the lock command fails, unless all the object locks are owned by the same user placing the module lock.

To remove a lock on a module branch, use the unlock command. This will release the lock on the module version.

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

```
lock [-[no]comment [<string>] [--] argument
```

ARGUMENTS

- Module Branch

Module Branch

<module branch>	A module branch or version in the form of a server URL.

OPTIONS

- -[no]comment

-[no]comment

-[no]comment <string>	Indicates whether to leave a comment with the lock. The lock comment is not saved when the module is unlocked. The comment provides a way for the locker to communicate the purpose of the lock. Note: To view the lock comment on a module branch, run the vhistory command on the locked module branch. -nocomment stores no comment to explain the purpose of the lock.(Default)

```
                        -comment <string> stores the value of <string>
                         as the lock comment. Use the vhistory command
                         to view the comment.
```

SEE ALSO

```
command defaults, ci, populate
```

RETURN VALUE

```
The command returns a status message indicating whether the lock
succeeded or failed.  If the lock failed, the message provides a
reason for the failure.
```

EXAMPLES

```
This example shows locking branch 1.2.1 on the Chip module to apply a
tag. After the lock has been applied, the example shows the lock
comment using the vhistory command.

stcl>lock -comment "Adding a tag to the module branch" \
     sync://srv2.ABCo.com:2647/Modules/Chip;1.2.1

 Beginning Lock operation...

 Locking:   sync://srv2.ABCo.com:2647/Modules/Chip;1.2.1 : Locked

 Lock operation finished.

{Objects succeeded (1)} {}

 vhistory sync://srv2.ABCo.com:2647/Modules/Chip;1.2.1
 Object:           sync://srv2.ABCo.com:2647/Modules/Chip;1.2.1
 ---------------------------
 Branch:           1.2.1
 Branch tags:      Gold
 Locked by:        rsmith; 1.2.1.1
 Comment:          Adding a tag to the module branch

 ---------------------------
 Version:          1.2.1.1
 Version tags:     Latest
 Derived from:     1.2
 Date:             Mon Oct 30 04:24:58 PM EST 2006
 Author:           rsmith

 ============================================================================
```

migratetag

migratetag Command

NAME

```
migratetag          - Migrates old DesignSync tags to modules
```

DESCRIPTION

The migratetag command provides a method for associating private tags or selectors used with legacy modules or DesignSync vaults that are not converted to the new module during the upgrading to modules.

The process of upgrading a legacy module or DesignSync vault, using the upgrade command, only migrates the configurations defined in the associated sync_project.txt files. There can be additional tags on versions and branches of the member vaults that are not defined in the sync_project.txt files, such as a private tags used to define a local configuration. The upgrade process preserves this tag information so private tags or selector lists using these tags can be migrated seamlessly to the new module. This command can only be run on a module upgraded with the upgrade command.

The migratetag command creates a new module branch containing the member versions that resolve to the supplied selector in the legacy module or DesignSync vault. This branch is tagged with a user-specified mutable tag name or the version tag supplied as the selector if the -branchname option is not specified.

The migratetag command can also be used to generate a list of the tags available to be migrated to the new module or module members with their version numbers that would be added to the new branch with the specified selector.

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

SYNOPSIS

```
migratetag [-branchname <branchtag>] [-list] <argument>
           [<selector>[,<selector>...]]
```

ARGUMENTS

- Server Module URL
- Selector(s)

Server Module URL

<server module>	Specify the URL as follows 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.

Selector(s)

<selector>	The selector used to identify the module member versions being migrated. This selector comes from the individual vault objects of the legacy module or DesignSync vault prior to the object being upgraded. The possible selector values include: o version tag selectors, such as Gold. o branch selectors, such as "Gold:" or Gold:Latest. o selector list. o version number selectors, such as 1.4. Note: The migratetag operation adds each member at the specified version number to the module version's manifest, and does not check to see if the member version exists. Note: Date selectors, auto-branch selectors and configurations defined in the sync_project.txt files do not identify any members to migrate. For more information on selectors, see the selectors help topic.

OPTIONS

- -branchname
- -list

-branchname

-branchname <branchtag>	The mutable branch tag to assign to the branch being created. If this option is not specified and the selector argument provided is a version tag, migratetag uses the provided version tag as the branchname. Tip: Tag names must be unique within a module. Using the -branchname option allows you to migrate a version tag, even if another tag already exists with that name, by allowing you

```
                        to specify a new, unique, branch tag name.
```

-list

```
 -list                   Specifies that the command should return
                         either a list of the tags that can be migrated
                         using the migratetag command or the list of
                        members and their version numbers that would
                        be added to the new branch during the
                        migration of the specified selector.
```

RETURN VALUE

```
When migrating tags (not using the -list option), if the versions
containing the tag-name are successfully migrated, the command
returns a Tcl list containing the new branch name and the new branch
number.  If the migratetag command fails, it returns an error.

When using the list option without a tag-name specified, the command
returns a Tcl list all the tags available to be migrated,
sorted alphabetically and the tag type, version or branch.

When using the list option with a selector specified, the command
returns a Tcl list of the module member versions identified by the
selector, sorted by natural path. A member version is returned with
the natural path of the member created during the upgrade and the
version number (for example: <path>;<Version.Number>).

If any of the following conditions are met, the migratetag command
fails and returns an appropriate error:

 o The branch tag being applied already exists.
 o No members resolve to the specified selector.
 o The module doesn't exist.
 o A selector was not specified when not using the list option.
```

SEE ALSO

```
command defaults, selectors, upgrade
```

EXAMPLES

- Example Showing the Available Tags for Migration
- Example Showing Migrating a Tag To a Module

Example Showing the Available Tags for Migration

```
This example provides a list of the tags available for migration on
the Doc module.

stcl> migratetag -list sync://serv1:2647/Modules/Doc

{bronze Branch} {gold Branch} {platinum Branch} {silver Branch}\
{Trunk Branch} {VTAG Version}

This example shows a list of the files associated with a specified
selector, in this case, gold:Latest.

stcl> migratetag -list sync://serv1:2647/Modules/Doc gold:Latest

{/install.doc;1.1} {/readme.txt;1.1} {/getstart.txt;1.1}
```

Example Showing Migrating a Tag To a Module

```
This example shows migrating the Beta version tag selector to the new
Doc module.  In this example, the -branchname is used to specify a
new tag name, DocBeta, because the "Beta" tag is already in use in
the module.

stcl> migratetag -branchname DocBeta sync://serv1:2647/Modules/Doc \
Beta

DocBeta 1.1.3
```

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

- Example Showing Module Branching (Module-based)
- Example of Branching Two Files From Your Workspace (File-based)
- Example Showing Branching The File Objects in the Workspace Recursively (File-based)
- Example Showing Branching the Server Version of a File (File-based)
- Example Branching the Entire Project from the Server (File-based)

```
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
```

mkedge

mkedge Command

NAME

```
mkedge              - Creates merge edges between specified
                      versions
```

DESCRIPTION

```
Merge edges are created for individual objects or entire modules
```

```
after a merge to simplify subsequent merges. For individual
objects, and module merging within a single branch, a merge edge is
created automatically at the checkin of the merged items after the
merge.  For cross-branch module merging, which contain a series of
changes committed in different commit actions, DesignSync can not
automatically determine what changes belong to the merge.

After the merge has been performed and all the desired changes have
been checked in, you can manually create a merge edge to create a
link between the version on the branch being merged with and the
post-merge module version created.

This command is subject to access controls.
```

SYNOPSIS

```
mkedge [-modulecontext <module>] -- <argument>
```

ARGUMENTS

- Workspace Module
- Module Version

Workspace Module

```
<Workspace module> Specify the local workspace module.  The command
                   identifies the version merged into the workspace
                   from a local record and, using the currently
                   populated version, creates a merge edge between
                   the two versions.
```

Module Version

```
<Module            The numeric version number of the version being
Version>           merged from. When you specify a module version,
                   you must also specify the module context field to
                   provide the server URL for the module being merged
                   to.
```

OPTIONS

- -modulecontext
- --

-modulecontext

```
 -modulecontext      Identifies the URL for the module being
  <context>          merged from the server, for example:
                     sync://server1:2647/Modules/Chip.

                     Note: When you specify the numeric module version
                    of the server module being merged from, you must
                    use the -modulecontext option to specify the
                    server module URL.
```

--

```
 --                  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 a TCL value of null ("").
If the command succeeds, it reports that the merge edge was created
from the merge from version to the merge to version.
If the command fails, it returns an error message explaining the
failure.
```

SEE ALSO

```
populate, ls, add, remove, addhref, edithrefs, rmhref, rmedge
```

EXAMPLES

```
This example shows a simple cross-branch module merge and the mkedge
command that creates the merge edge between the two versions.

The first step in the cross-branch module merge is the populate with
-merge and -overlay selected.

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, create the
merge edge.

dss> mkedge ROM%1
 Edge from 1.3.1.3 to 1.5 for module
 sync://srv2.ABCo.com:2647/Modules/ROM created successfully.
```

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
```

mvmember

mvmember Command

NAME

```
mvmember            - Changes the natural path of a module member
```

DESCRIPTION

- Moving Folders

```
This command is used to rename a module object.  Renaming the object
can include changing the name of the object, moving the object to a
different location, or both. Moving an object in the workspace
updates the object on the server. Moving a specified server object
directly does not affect any workspaces containing the moved object
until the workspace is updated.

Tip: Performing the mvmember command on the workspace object removes
any properties set on the object by a cross-branch module merge.

The mvmember command moves a module member by performing the
following actions:
       - In the workspace, the module member is renamed or moved to a
         specified new location.
         Note:  If the specified member is a folder, the folder itself is
         not moved, only the folder contents.
       - On the server, a new version of the module is created with
         a changed natural path for the specified object (-immediate)
         or the workspace metadata is updated to indicate that the
         module member will be moved during the next module
         checkin (-noimmediate).

When moving a workspace member, you must be working with the Latest
version of the member present in the workspace. You can modify your
copy of the object and move the object before checking the object
back in.  The mvmember command does not check in the modified object,
it will only change the path or object name of the specified name.
```

When you perform the next checkin operation, the modifications are checked in. The natural path of the object is checked to ensure that no characters that have been deliberately restricted are used. For more information on excluded characters, see Exclude Lists in the DesignSync Data Manager Administrator's Guide.

Note: If you are using -immediate, the module version created with the mvmember command does not have the local modifications. If you are using -noimmediate, the module version created with the checkin operation has both the name change and the updated content.

You cannot move an object that another user has locked. If you move an object you have locked, you retain the lock after the move has completed.

When mvmember is performed on objects that have been added to the workspace but not checked in, the workspace member is renamed and remains in the added state to be processed on checkin.

Notes:
- Objects in share or reference mode cannot be moved with the -noimmediate flag.
- Collections can be moved to a different location, but cannot be renamed with the mvmember command.

IMPORTANT: When using mvmember with the -noimmediate option, you cannot rename files to use the same name as another file being moved within the change set. For example: Using file a.txt and b.txt, you could not rename a.txt to c.txt; then b.txt to a.txt. You could rename a.txt to c.txt, perform a checkin and then rename b.txt to a.txt.

Note: This command cannot be run on module members in a module snapshots. Module snapshots cannot be content modified after creation.

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.

Moving Folders

The mvmember command can be used to move folders on the server. When a folder is renamed or moved on the server, the contents of the folder, including subfolders, move with it.

Note: If any module members in the folder are locked by another user, the folder does not move.

After the contents of a folder are moved, the folder is removed from the server, and if the module is populated with the -force option, removed from the workspace.

When the mvmember command is run on a workspace folder, it moves the contents within the folder (files or collection objects) to the specified location, and removes the folder on the workspace, but does not move the folder or any subfolders on the server, which means that those folders may still remain as empty directories within the module manifest. To recursively move folders in the module, use the mvmember command on the server folder, and then update your workspace with the populate command.

Tip: The best way to move a module folder is to move the folder on the server. This guarantees that all module members move with the folder. If you move the workspace folder, you only move the module members present in the workspace.

SYNOPSIS

```
mvmember  [-[no]comment] [-[no]immediate] [-modulecontext <context>]
          [-[no]selected] [--] [<fromargument>...] <toargument>
```

FROMARGUMENTS

- Server Module Folder
- Workspace Module Folder
- Module Member
- Select List

Server Module Folder

<server module folder>	Selecting a server module folder as a fromargument moves the entire folder and its contents to the folder specified as the toargument. The toargument must be the path to a module folder if the fromarguments are module folders. Note: When a server -modulecontext is specified, then both the fromargument(s) and toargument must use the full directory path, in UNIX form, beginning after the workspace root. For more information, see the Examples section.

Workspace Module Folder

<workspace module folder>	Selecting a workspace module folder as a fromargument moves the contents of the folder to the folder specified as the toargument. It does not move the folder or subfolders on the server, which can result in empty folders in the module manifest. The toargument must be the path to a module folder

if the fromarguments are module folders.

Module Member

<module member> Selecting a module member as the fromargument moves the file to the location or name specified by the toargument.

If only one fromargument is supplied, and the toargument does not exist already, the fromargument object is renamed to the name specified in the toargument.

If more than one fromargument is supplied, and the toargment does not exist already, a directory named toargument is created and the fromargument objects moved into it.
Note: When a server -modulecontext is specified, then both the fromargument(s) and toargument must use the full directory path, in UNIX form, beginning after the workspace root. For more information, see the Examples section.

Select List

<select list> Using a select list to provide the list of objects to move allows you to easily specify a set of objects to move. When using a select list, you do not have to provide a fromargument. Use the -selected option to specify that the command use the select list. For more information on select lists, see the "select" command. You may use the select list in conjunction with other fromarguments.

TOARGUMENTS

- Module Folder
- Module Member

Module Folder

<module folder> Specifying a module folder as the toargument moves all appropriate objects into the specified folder. If no folder exists by the name specified as the toargument, and there is more than one fromargument, the system creates the folder on the workspace and the server.

Note: When a server -modulecontext is specified, then both the fromargument(s) and toargument must use the full directory path, in UNIX form, beginning after the workspace root. For more information, see the Examples section.

Module Member

<module member> Specifying an object as the toargument renames the fromargument to the toargument.
Note: When a server -modulecontext is specified, then both the fromargument(s) and toargument must use the full directory path, in UNIX form, beginning after the workspace root. For more information, see the Examples section.

OPTIONS

- -[no]comment
- -[no]immediate
- -modulecontext
- -[no]selected
- --

-[no]comment

-[no]comment "<text>" Specifies whether to remove 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 ampersand (&) and equal (=) characters are replaced by the underscore (_) character in revision control notes.

-comment specifies a reason for the object removal which is included in the module history.

-nocomment does not specify a reason for the object removal.

-[no]immediate

-[no]immediate Determines whether to immediately perform the mvmember operation or mark the objects for moving during the next module checkin.

-noimmediate renames object in the workspace, but does not create a new module version on the server. When the next module version is checked

in, the name changes are checked in to the server version. (Default) Objects populated in -share or -reference mode cannot be moved with -noimmediate.

-immediate removes the object and creates a new module version immediately. Note that any modifications to the content of the file are not checked in to this module version. You must perform a check in to update the file contents.

Note: Objects in the Add state are always immediately renamed. No new module version is created.

The -immediate option is ignored when mvmember is performed on server objects.

-modulecontext

-modulecontext <context>	Identifies the workspace module or server module version in which the objects are being moved. If no module context is selected, DesignSync attempts to identify the desired module. If there is an obvious module target, for example, only a single module in the workspace, DesignSync automatically selects the module. If DesignSync cannot identify the module, the command returns an error stating that the module can not be identified an recommending the use of the -modulecontext option. 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]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.

--

--	Indicates that the command should stop looking for command options. Use this option when arguments to the command begin with a hyphen (-).

SEE ALSO

```
command defaults, select
```

EXAMPLES

- Example of Moving a Folder
- Example of Renaming a File
- Example of Renaming a File and Immediately Creating a New Version

Example of Moving a Folder

```
This example shows moving a folder, doc on a server.  Note that
the operation is recursive automatically, even though there is no
-recursive option for the command.  The contents command is used to
show that all the contents of the /doc directory moved.

stcl> contents -nodefaults -recursive -report normal -hrefmode normal
 sync://serv1/Modules/Chip

 Gathering data from vault sync://serv1/Modules/Chip@Trunk:Latest

 Module: sync://serv1:2647/Modules/Chip@1.6
  ...
 Contents of folder: /doc
 Object Name
 -----------
 cpu.doc

stcl> mvmember -immediate -modulecontext sync://serv1/Modules/Chip
 /doc /cpu
Beginning mvmember operation...

Moving objects in module sync://serv1:2647/Modules/Chip...
/doc : Moved object to /cpu/doc
Finished mvmember for module sync://serv1:2647/Modules/Chip :
Created new version 1.7

Finished mvmember operation.

{Objects succeeded (1)} {}

stcl> contents -nodefaults -recursive -report normal -hrefmode normal
sync://serv1/Modules/Chip

 Gathering data from vault sync://serv1/Modules/Chip@Trunk:Latest

 Module: sync://serv1:2647/Modules/Chip@1.7
  ...
 Contents of folder: /cpu/doc
```

```
 Object Name
 -----------
 cpu.doc
```

Example of Renaming a File

```
This example shows moving a file, libcreate.c, populated to the
"chip" workspace, to a different file name, libcreateclass.c.

stcl> mvmember -modulecontext Chip%1 libcreate.c libcreateclass.c

 Beginning mvmember operation...

 Moving objects in module Chip%1...
 /libcreate.c : Moved object in workspace to /libcreateclass.c

 Finished mvmember operation.

 {Objects succeeded (1)} {}


stcl> ci -comment "moved libcreate to libcreateclass and add files"
 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.libcreateclass.c         Success - Renamed from
 /libcreate.c
 Checking in: /doc/chip.doc                  Success - New version:
 1.1
 ...
 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 (11)} {}
```

Example of Renaming a File and Immediately Creating a New Version

```
This example shows moving a file, libcreate.c, populated to
the "chip" workspace, to a different file name, libcreateclass.c
using the -immediate option.

stcl> mvmember -immediate -modulecontext Chip%0 libcreate.c
libcreateclass.c

Beginning mvmember operation...

Moving objects in module Chip%0...
/libcreate.c : Moved object to /libcreateclass.c
Chip%0 : Created new module version 1.9
Chip%0 : Auto-merge operation - workspace version of module not updated.

Finished mvmember operation.

{Objects succeeded (1)} {}

Note: This is an auto-merge operation, meaning that the object was
moved in the workspace and the new module version, but because there
is a later module version in the vault, the fetched version in the
workspace was not updated. To update the module version in the
workspace, and fetch other pending changes, perform a populate.
```

mvmod

mvmod Command

NAME

```
mvmod               - moves a module to another location on the server
```

DESCRIPTION

```
This command moves your module from the current location to another
location on the server.  This allows you to re-categorize modules, if
needed.

Note: This does not allow you to move the module to a different
server. To move a module to a different server, use the exportmod and
importmod commands.

The module move process moves all the module members and hierarchical
references and preserves the module history.

BEST PRACTICES TIP: Before performing a module move, verify that you
have a current backup of the server containing the module.

Note: The process does not update notes, access controls,
subscriptions or mirrors.

When you run the module move, the DesignSync server performs the
```

```
following operations:

o by default, freezes module in the old location so no changes can be
  made while the module is being moved.
o bundles the module into a compressed format containing the history,
  hierarchical reference information, and module contents.
o imports the module to the new location and recreates the
  hierarchical references.
o updates the hierarchical references that point to the old module to
  the new module location.
```

SYNOPSIS

```
mvmod [-[no]freeze] <OldServerURL> <NewServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL> Both the oldServerURL and the NewServerURL use the same
            format to specify the old and new URL for 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]freeze

```
-[no]freeze Specifies whether to freeze all branches for both the
            original and new module so no additional changes can be
            made before the module is released for normal usage.

            -nofreeze immediately releases the freeze on the new
             module after the import has completed. This means the
             module is immediately available for changes upon
```

```
              completion of the mvmod operation.

             -freeze leaves the new module in a frozen state after the
              move so no changes can be made, allowing the module
              mover to perform any additional structural changes
              required. (Default)

             Note: You can remove the module freeze for either module
             using the unfreeze command.
```

RETURN VALUE

SEE ALSO

```
 exportmod, importmod, freezemod, unfreezemod, reconnectmod
```

EXAMPLES

```
 This example shows moving a module to a new location on the server.
 dss> mvmod sync://serv1.ABCo.com:2647/Modules/Components/ROM
  sync://serv1.ABCo.com:2647/Modules/ChipComponents/ROM
 Beginning module move ...
 Updating back references of referenced objects ...
 sync://serv1.ABCo.com:2647/Modules/Components/ROM : Module is frozen.
 Reconnecting parent modules ...
 Beginning module reconnect ...

 The following hierarchical reconnection will be made to each parent module:
     From: sync://serv1.ABCo.com:2647/Modules/Components/ROM
     To:   sync://serv1.ABCo.com:2647/Modules/ChipComponents/ROM

 sync://serv1.ABCo.com:2647/Modules/Components/ROM : Updating
 hierarchical references ...
 sync://serv1.ABCo.com:2647/Modules/Components/ROM : Updating
 reconnect history ...
 sync://serv1.ABCo.com:2647/Modules/ChipComponents/ROM  : Adding back
 references ...
Module successfully moved.
```

purge

purge Command

NAME

```
 purge               - Purges specified branches or versions of
                       objects from the vault
```

DESCRIPTION

- Restrictions
- Triggers and Revision Control Notes and 'purge'
- Error Handling
- Using Purge with Modules (Module-based)
- Using Purge with Files-Based Objects (File-based)

```
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          Specifies the module instance name or path of the
 module>            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
- Example of Using Keep Since to Maintain 30 Days of Versions
- Example of Using both the -keepsince and -keepversions Options
- Example of Purging Versions from the Server
- Example of Making then Purging a Branch
- Example Showing Module Purge on the Trunk Branch (Module-based)

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)}
```

reconnectmod

reconnectmod Command

NAME

```
reconnectmod        - Updates the hrefs from a module to a new module
```

DESCRIPTION

```
After you have moved a module (with the exportmod/importmod
commands), you can update hierarchical references pointing to the old
module to the new module. This operation does not create a new module
version of the referencing modules, it modifies the reference within
the module version to point at the new location.  This maintains the
integrity of both static and dynamic hierarchical references after a
module has changed location.

You can update the references all modules or a single module, or a
list of modules.

Note: When doing a move module with the mvmod command, the reconnect
command is called automatically as part of that operation.

This command supports the command defaults system.

This command is subject to access controls on the server.
```

SYNOPSIS

```
reconnectmod  [-[no]force] [(-from <oldServerURL>) |
              (-parents <TCLlist>)] <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>     Specifies the URL of the new 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

- -force
- -from
- -parents

-force

```
 -[no]force            Determines whether to force the modulereconnect
                        to module specified with the -from option, even
                        if url/uid for the modules doesn't match the
                        information contained in the imported
                        transportable module.

                        -noforce does not recreate the href is the
                        url and uid information does not match the
                        expected module information.  (Default)

                        -force recreates the href even if the url and
                        uid information contained in the imported
                        transportable module does not match the information
                        for the target module specified with the -from
                        option.
```

-from

```
 -from <OldServerURL>  Specifies the URL of the old 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

                        Note: This option is mutually exclusive with -parents.
```

-parents

```
-parents <TCLlist>    Specify the list of urls identifying parent
                      modules to update. By default, the list of
                      parents is retrieved from the whereused
                      information included with the module when the
                      module is imported.

                      Note: This option is mutually exclusive with
                      -from.
```

RETURN VALUE

SEE ALSO

```
mvmod, addbackref, addhref, rmhref, edithrefs
```

EXAMPLES

```
This example shows how to use reconnectmod after moving a module.
dss>  reconnectmod sync://serv2.ABCo.com:2647/Modules/ChipDesign/Chip-NX2
 Beginning module reconnect ...

 The following hierarchical reconnection will be made to each parent
 module:
    From: sync://serv1.ABCo.com:2647/Modules/Chips/chip-nx1
    To:   sync://serv2.ABCo.com:2647/Modules/ChipDesign/Chip-NX2

sync://serv1.ABCo.com:2647/Modules/Components/ROM : Updating
hierarchical references ...
sync://serv1.ABCo.com:2647/Modules/Components/ROM : Updating
reconnect history ...
sync://serv3.ABCo.com:2647/Modules/Components/CPU : Updating
hierarchical references ...
sync://serv3.ABCo.com:2647/Modules/Components/CPU : Updating
reconnect history ...

sync://serv2.ABCo.com:2647/Modules/ChipDesign/Chip-NX2 : Adding back
references ...
sync://serv1.ABCo.com:2647/Modules/Chips/chip-nx1 : Removing back
references ...
{Objects succeeded (4)} {}
```

remove

remove Command

NAME

```
remove              - Removes a module member from a module
```

DESCRIPTION

- Removing a folder from a module

```
This command removes the selected module members from a module and
creates a new module version without the selected module
members. Unlike add, which creates a new module version only after
the new module members have been checked in, the remove command can
either immediately create a new module version without the removed
module member or queue a series of changes for the next module
checkin.

Note: The object being removed is removed only from the new version
of the module and subsequent modules. It is not removed from the
vault and remains in previous module versions.

Important: This command is not used to remove modules.  To remove
modules, see the rmmod command.  This command is not used to remove
hierarchical references.  To remove hrefs, use the rmhref command.

Module members can be removed from either a populated workspace or
directly on the server. Module members include files, folders,
and symbolic links.

Tip: Performing the remove command on the workspace object will
remove any properties set on the object by a cross-branch module
merge.

Note: How a module folder remove is handled depends on whether the
remove is done on the workspace or on the server.  For more
information see "Removing a folder from a module" section.

If an object was added to a module, but never checked in, and it is
removed, the object is deleted.  To retain the object, but revert it
to an unmanaged state, specify the remove with the -keep
option. Note: When removing an object that has never been checked in,
you must use the -force option. If you do not use -force, the remove
fails.

Unmanaged items or module members from legacy modules are not part of
a module, and cannot be removed from a module.

This command never removes objects locked by another user. If the
user who owns the lock performs the remove, the lock is removed first
and then the object is removed from the module.

Note: This command is not applicable to module snapshots which cannot
be content modified after creation.
```

You can use the remove command to remove a folder from the workspace or from the server. You can also use the remove command to remove a folder that was added to a module, but not checked in.

Important: Performing a remove on a workspace folder never affects the server version of the folder. It only removes the workspace folder from the workspace.

If you perform a remove on a workspace folder without the -recursive option, it will remove the contents of the folder, but not the folder itself. If you specify -noimmediate, it does not remove the folder immediately, but marks it for removal and removes it during the next module checkin.

If you perform a remove on workspace folder recursively and do not specify the -keep option, DesignSync removes the specified folder, if it becomes empty and all subfolders that become empty as a result of removing their contents. The empty folders remain on the server, but are no longer part of your workspace.

Notes: If you want to recreate the folders so your workspace reflects the server module structure, you must perform a full populate.

If the folders contain non-module data, that data is not removed with the remove command. Folders that contain the non-module data are not empty and are not removed.

If you remove a workspace folder recursively and specify the -keep option, the empty folders remain in the workspace and on the server.

If you remove a server folder recursively, you remove the specified folder and all folders on the server. In order to update the individual workspaces with the server changes, repopulate the workspaces with the -force option selected.

If a folder was explicitly added with the add command but has not been checked in, you can remove it from list of objects to be added to the module by using remove without specifying the recursive option. Any added objects within the folder are still added to the module when checked in, but the directory is implicitly added to the module, rather than explicitly added, which means it will be removed automatically if it becomes empty.

If you specify the -recursive option with the remove command, the folder and any objects within the folder are not added to the 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.

SYNOPSIS

```
remove [-[no]comment "<text>"] [-filter <string>] [-[no]force]
       [-[no]immediate] [-[no]keep] [-modulecontext <context>]
       [-[no]recursive] [--] <argument>
```

ARGUMENTS

- Module Folder
- Module Member

Module Folder

`<module folder>` Removes module objects in the folder from a module. Specify the -modulecontext option if more than one module exists in the workspace or if you want to remove items directly from the module on the server; the remove fails if DesignSync cannot determine from which module to remove the folder. If the folder exists in multiple modules, it is removed from all the modules, unless a specific module context is specified.

Workspace - You must use -recursive to remove a folder from a workspace. If a folder is specified, the workspace folder is removed after all the files have been removed from the module. The empty module folder on the server will not be removed unless it is explicitly removed.

Server - In order to remove files or folders that are not populated into the workspace, you must specify the -modulecontext option and provide the natural path of the server folder. You may use wildcard characters in the natural path.

Tip: When removing a folder, you should specify a server module. This assures that you remove the entire folder and its contents from the module. If you perform the remove on the workspace folder, you can only remove the elements that were populated in the workspace.

For more information on how remove works on folders, see the "Removing a folder from a module" section above.

Module Member

`<module member>` Removes the module member. If the member has not

```
been checked in, but it has been added to the
module using the add command, the file
reverts to being unmanaged and the module version
is never updated to include the file. Specify the
-modulecontext option if more than one module
exists in the workspace; the remove fails if
DesignSync cannot determine from which module to
remove the objects. If the object being removed
is not populated to the workspace, you must
specify both the -modulecontext option and
provide the natural path of the server folder.
Wildcards may be used in the natural path
```

OPTIONS

- -[no]comment
- -filter
- -[no]force
- -[no]immediate
- -[no]keep
- -modulecontext
- -recursive
- --

-[no]comment

```
-[no]comment
"<text>"        Specifies whether to remove 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
                ampersand (&) and equal (=) characters are
                replaced by the underscore (_) character in
                revision control notes.

                -comment specifies a reason for the object
                 removal which is included in the module history.

                -nocomment does not specify a reason for the
                 object removal.
```

-filter

```
-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; 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 modified objects should be removed from the workspace. -noforce does not remove the specified object if it has been modified. (Default) -force removes the specified object even if it has been locally modified. Note: If you are removing objects that were added to a module, but never checked in with ci, you must use -force to remove the objects, since they are considered locally modified.

-[no]immediate

-[no]immediate	Determines whether to immediately perform the remove operation or mark the objects for removal during the next module checkin. -noimmediate removes the object from the workspace immediately, unless the -keep option is also specified, but does not immediately create a new module version. When the next module version is checked in, the objects specified are removed from the server module version.(Default) -immediate removes the object and creates a new module version immediately. Note: Objects in the Add state are always immediately removed from the Add state. No new module version is created.

-[no]keep

-[no]keep	Specifies whether to delete local copies of objects after the object has been removed from the module. This option is ignored if the remove is done directly on a server object. -nokeep removes local copies of the removed object and empty subfolders from the workspace. (Default) -keep leaves local copies of the removed objects and empty subfolders in the

workspace. Retained objects become unmanaged objects in the workspace.

-modulecontext

-modulecontext <context>	Identifies the module version from which the objects are being removed. Use the -modulecontext option to restrict the remove to only a particular module if your workspace has overlapping modules so that you can indicate which module you want the objects removed from. 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 remove 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.

-recursive

-[no]recursive	Determines whether only the specified folder should be removed, or whether all objects in the folder and any subfolders should be removed as well. This option is ignored if the argument is not a module folder. -norecursive removes only the specified folder if it is empty, or, removes the folder from the list of objects to be added to a module. (Default). If the folder is already a module member (not in the added state), and is not empty, the command returns an error. -recursive removes the specified folder and all module objects in the folder and all subfolders. The -recursive option is required to remove any non-empty folder. A folder can only be removed if there are no objects remaining in the folder. If a folder contains objects from different modules only objects from the module specified with -modulecontext are removed. The other objects remain and the folder remains a member of the non-specified modules.

--

```
--                  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 remove command is a count of how many
objects were successfully and unsuccessfully removed from the module.
```

SEE ALSO

```
command defaults, add, mkmod, rmmod, rmfile, rmfolder, ci
```

EXAMPLES

- Example of Removing Added Files
- Example of Removing a Folder
- Example Showing Removing Using Wildcards for Pattern Match

Example of Removing Added Files

```
This example shows removing files from two directories that were
added to a module, but never checked in.  Note: -force is required to
remove these objects.

stcl> remove -recursive -force doc verilog

 Beginning remove operation...

 /MyModules/Chip/doc/Chip.doc : Removed object that had never
  been checked in
 /MyModules/Chip/verilog/chip.v : Removed object that had never
  been checked in
 /MyModules/Chip/doc : Folder no longer has module members, removed
  from workspace and workspace module Chip%0
 /MyModules/Chip/verilog : Folder no longer has module members,
  removed from workspace module Chip%0

 Finished remove operation.

 {Objects succeeded (4)} {}

Note: Because the doc folder contained no other members and no other
```

```
files, it was removed from both the workspace module and physically
removed from the workspace. The verilog folder, on the other hand,
still contained an object (unmanaged)and was removed from the
workspace module but the directory left in the workspace. The Chip
folder itself still contained module members and was not removed from
the module.
```

Example of Removing a Folder

```
This example shows removing a folder called Doc from the server
module.  The folder is not empty, so it requires -recursive.

stcl> remove -recursive -immediate \
  -modulecontext sync://myserver:2647/Modules/Chip;1.3 /Doc

Beginning remove operation...

Removing objects from module sync://myserver:2647/Modules/Chip;1.3...
/Doc : Removed object
Finished remove for module sync://myserver:2647/Modules/Chip;1.3 :
Created new version 1.4

Finished remove operation.

{Objects succeeded (1)} {}
```

Example Showing Removing Using Wildcards for Pattern Match

```
This example shows removing all the of the html and pdf files from
the Doc directory and then updating the module with the changes.

stcl> remove -recursive -noimmediate -modulecontext Chip%1 doc/*.html

 Beginning remove operation...

 Deleting objects from workspace module Chip%1...
 /doc/index.html : Removed object in workspace
 /doc/interface.html : Removed object in workspace
 /doc/commands.html : Removed object in workspace

 Finished remove operation.

 {Objects succeeded (3)} {}

stcl> remove -recursive -modulecontext Chip%1 doc/*.pdf

 Beginning remove operation...

 Deleting objects from workspace module Chip%1...
 /doc/manual.pdf : Removed object in workspace

 Finished remove operation.
```

```
 {Objects succeeded (1)} {}

stcl> ci -comment "Removed obsolete documentation" Chip%1

 Beginning Check in operation...


 Checking in objects in module Chip%1

 Total data to transfer: 0 Kbytes (estimate), 0 file(s), 0 collection(s)
 Progress: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
 Progress: 2 Kbytes, 0 file(s), 0 collection(s), 100.0% complete

 Checking in: /doc/commands.html             Success - Removed
 Checking in: /doc/index.html                Success - Removed
 Checking in: /doc/interface.html            Success - Removed
 Checking in: /doc/manual.pdf                Success - Removed

 Chip%1: Version of module in workspace updated to 1.9

 Finished checkin of Module Chip%1, Created Version 1.9

 Time spent: 0.3 seconds, transferred 2 Kbytes, average data rate 5.9 Kb/sec

 Checkin operation finished.

 {Objects succeeded (4)} {}
```

rmedge

rmedge Command

NAME

```
rmedge              - Removes previously set merge edges
```

DESCRIPTION

```
Merge edges are created for individual objects or entire modules
after a merge to simplify subsequent merges. If a merge edge is no
longer needed, you can remove it.

This command is subject to access controls.
```

SYNOPSIS

```
rmedge -modulecontext <moduleURL> -- <argument>
```

ARGUMENTS

- Module Version

Module Version

<Module Version>	The numeric version number of the module version on which the merge edge was created.

OPTIONS

- -modulecontext
- --

-modulecontext

-modulecontext <context>	Identifies the URL for the module the merge edge was created on, for example: sync://server1:2647/Modules/Chip. Note: The modulecontext option is required.

--

--	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 a TCL value of null ("").

If the command succeeds, it reports that the specified merge edge was removed.

If no merge edge exists on the specified version, the command notes that no merge edge exists.

If the command fails, it returns an error message explaining the failure.

SEE ALSO

populate, ls, mkedge

EXAMPLES

```
In the example for mkedge, a merge edge was created between two
versions of the ROM module.

 dss> mkedge ROM%1
 Edge from 1.3.1.1 to 1.4 for module
 sync://srv2.ABCo.com:2647/Modules/ROM created successfully.

This example shows the removal of that merge edge.

 dss> rmedge -modulecontext sync://srv2.ABCo.com:2647/Modules/ROM;1.4 \
      1.3.1.1
 Edge from 1.3.1.1 to 1.4 for module sync://serv2.ABCo.com:2647/Modules/ROM
 removed successfully.
```

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
- Example of Removing Two Files
- Example of Removing a File with a Leading "-"
- Example of Removing a Member of a Collection

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
- Example of Removing Folders Recursively
- Example of Removing a Folder on the Server
- Example of Removing a Folder Containing References

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
```

rmmod

rmmod Command

NAME

```
rmmod                - Removes a module from a server or workspace
```

DESCRIPTION

- Removing a Workspace Module Recursively
- Removing an External Module
- Removing a legacy module from a server

```
This command removes a module (data that represents a level of
the design hierarchy), module cache link, and, for
legacy modules, its module configurations, from a server or
workspace.

For information on deleting legacy modules, see the "Removing a
legacy module from a server" section.

For information on removing a module (a data replication instance)
from data replication, see the replicate rmdata command.

This command also detaches any ProjectSync notes that may have been
attached to the module, module branches, or module versions. These
notes can be retained in the ProjectSync system for future reference,
or removed using the -notes option.

This command can also be used to detach an external module workspace
from the server so you can delete the external module objects. Using
rmmod, you remove the linkage between the server and the workspace,
turning the objects in the external module into unmanaged objects
which can then be removed by the file system delete commands. To
remove the external module reference from the parent module, use the
rmhref command.

The rmmod operation generates a RevisionControl note for the deleted
module. The generated RevisionControl note is detached and, if the
-notes option is specified, deleted along with the module.

If the rmmod command removes all the contents of the base
directory of a module, DesignSync automatically removes the base
```

directory as well. If the base directory contains information from other modules or unmanaged objects, DesignSync does not remove the base directory.

When the rmmod operation completes, it displays a success/failure count. This count only includes modules and, when run recursively in the workspace, hierarchical linked objects. It does not include module members or individual objects within in hierarchically referenced set of objects.

Notes:
- The rmmod command supports removing a module hierarchy recurisively within a workspace. It can not be run recursively against a server module. To remove an entire module hierarchy on the server, you must remove each module separately.

- When rmmod is used to remove a workspace module, external module, or a link to an mcache, it does not affect the module on the server or in the mcache. To remove a server module, you must explicitly use rmmod on the server module. To remove a module from the mcache, you must use rmmod on mcache module instance.

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.

Removing a Workspace Module Recursively

DesignSync allows you to remove a workspace module hierarchy in a single operation. Starting from a (modern) module, you may remove a module hierarchy that can include modern modules, legacy modules, DesignSync vaults, deliverables, external modules, and mcache links to modern and legacy modules.

When removing a workspace module hierarchy, if DesignSync encounters an error removing objects within the hierarchy, it does not remove the module or reference that generated the error, but does continue attempting to remove the remaining hierarchy.

Note: If there are locked or modified objects in the workspace, the rmmod operation fails for that module unless the -force option is specified. If -force is used, the member locks held in the workspace are canceled on the server and locally modified members are removed..

Any folders that become empty as a result of the rmmod operation are removed. If a referenced submodule is also referenced by a different module in the workspace, the referenced submodule is not removed.

When the -unmanaged option is specified, any unmanaged data is removed if the folder containing the unmanaged data does not contain other managed data, for example from a overlapping modules, populated vault data, legacy modules, etc.

If the unmanaged option is not used and unmanaged data remains in the workspace, the data can be removed later by using the rmfolder -recursive option, or by deleting the objects using the operating system commands.

Note: If the module remove fails, no unmanaged data in that module directory structure is removed.

After the module data has been removed, the workspace module metadata is removed.

Removing an External Module

Removing an external module is largely governed by the external CM system. Running rmmod on an external module, or removing an external module as part of running rmmod recursively over the workspace module hierarchy uses the -xtras option to send the appropriate argument to the external CM system.

As with all external module operations, DesignSync must have an external module procedure and the -xtras option is used to pass any additional parameters and options to the procedure. If there is no definition for rmmod within the external module procedure, DesignSync considers this a failure to remove the external module; however DesignSync will remove the module metadata, effectively disconnecting the external module from DesignSync. The data then is considered unmanaged data in the workspace and is handled according to the value selected for the -[no]unmanaged option.

Removing a legacy module from a server

Important: The information in this section only applies to legacy modules.

Removing a module with the rmmod command automatically removes the vault associated with the module. When removing legacy modules, you can leave the vault data data intact by specifying the -vaultdata option to the rmmod command. For more information, see the -vaultdata option description.

If you remove a legacy module without removing the vault folder (running the rmmod command without the -vaultdata option), the showconfs command continues to display the configurations for the deleted module. Additionally, if the module resides in the /Projects folder, the module and its configurations appear in ProjectSync as a ProjectSync project.

Removing a module detaches any ProjectSync notes that may have been attached to the module configurations. These notes can be retained in the ProjectSync system for future reference, or removed using the -notes option.

```
Removing a module generates a RevisionControl note for the deleted
module. The generated RevisionControl note is attached to the module
and includes the URLs (shown as paths relative to the server vault)
of the module and the module's configurations. This note remains
attached to the module.  When a ProjectSync note query is run on that
folder the note is retrieved, showing that the folder once contained
the data of a deleted module. If you specify -notes this revision
control note is deleted along with the rest of ProjectSync notes
associated with the module. For more information, refer to the -notes
option description.

WARNING: Using this command, you can delete legacy module
         releases. By default, the use of this command is
         restricted. It is recommended that administrators remove
         this restriction temporarily, if needed, and with extreme
         care.

When the rmmod command is run on legacy modules, it only affects
only the server; it does not affect your work area. If you fetched a
configuration of the module into local work area, that configuration
still exists in your local work area.

This command is not recursive, it does not follow the hierarchical
references of the module configurations and delete the referenced
modules on the server.  To remove an entire module hierarchy, you
must remove each module separately.
```

SYNOPSIS

```
rmmod [-[no]force] [-[no]keep] [-[no]notes] [-[no]recursive]
      [-report error|brief|normal|verbose][-[no]unmanaged]
      [-[no]vaultdata] [-xtras <string>] <argument>
```

ARGUMENTS

- Server Module
- Workspace Module
- Mcache Link
- Mcache Module
- External Module

Server Module

```
<server module>        URL of the module being removed.
                       Specify the URL as follows:
                       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 remove.

Workspace Module

<workspace module>	Name of the module being removed from the workspace. Specifying a workspace module removes the module and its contents from the workspace. It does not remove the module from the server. Note: Legacy modules can only be removed on the server. You cannot specify a legacy workspace module.

Mcache Link

<mcache link>	Module instance name for the mcache link module being removed from the workspace. Specifying an mcache link module removes mcache link from the workspace. It does not remove the module from the cache or the server.

Mcache Module

<mcache module>	Module instance name for a module in a module cache. Specifying a mcache module does not remove the module from the server. It does break any existing links in workspaces linking to the mcached module.

External Module

<external module>	Module instance name for an external module. Specifying an external module removes the workspace metadata that tells DesignSync that the objects within the workspace are under revision control. It does not remove the objects from the workspace and it does not remove the hierarchical reference from the parent module to the external module.

OPTIONS

- -[no]force
- -[no]keep
- -[no]notes

- -[no]recursive
- -report
- -[no]unmanaged
- -vaultdata
- -xtras

-[no]force

-[no]force	Specifies whether the locked and modified module members in the workspace should be removed. -noforce does not remove any locked or modified module members in the workspace. If any module members are not removed, the module itself is not removed.(Default) Important: Any unmodified and unlocked items in the module are removed even though the module is not removed. -force removes the module and all module contents, including any locked or locally modified objects. If the -force option is used with a server object, it is silently ignored.

-[no]keep

-[no]keep	Specifies whether to keep the module member data in the workspace after the module has been removed. This option is only applicable when rmmod is run on a workspace module. -nokeep removes all module data; module members and DesignSync metadata from the workspace. (Default) -keep removes only the module metadata, it does not remove the module members. If the module members are links from a cache (populated in -share mode), the server copies the files locally and removes the links. Note: If modules members are references (populated in -reference mode), rmmod always uses -nokeep, removing the references and any additional metadata. The -keep option is silently ignored when used with a workspace option. The -keep option is

mutually exclusive with the -unmanaged option.

-[no]notes

-[no]notes	Indicates whether to delete the orphaned notes or preserve them. -nonotes retains all the ProjectSync notes associated with the deleted module. (Default) -notes deletes all the notes, including the RevisionControl note generated by the module deletion, attached to the deleted module if the notes have no ties to any other live projects or modules. Note: Legacy modules may still retain the RevisionControl note indicating that the module was deleted unless the default RmVaultKeepVid behavior has been overridden and version information for the removed objects is not retained. This option is silently ignored when used with a workspace argument.

-[no]recursive

-[no]recursive	Specifies whether the operation should run in recursive or non-recursive mode. This option is only applicable when rmmod is run on a workspace module -norecursive remove only the selected module from the workspace. (Default) -recursive removes the selected module from the workspace and all the referenced sub-modules. Submodules can be modern modules, legacy modules, DesignSync vaults, deliverables, external modules, and mcache links to modern and legacy modules

-report

-report error|brief| normal|verbose	Determines what information is returned in the output of the rmmod command. Valid values are: o error|brief - lists any errors encountered

during the rmmod operation.

o normal - lists errors and a single line for each module removed. (Default)

o verbose - lists errors and provides full status for each object processed.

-[no]unmanaged

-[no]unmanaged

Specifies whether the operation should remove or retain any unmanaged files within the workspace module directory structure.

-nounmanaged does not attempt to remove any unmanaged data and any folders containing unmanaged data remain after the operation completes. (Default)

-unmanaged removes any unmanaged data within the module directory structure after the module is removed.

IMPORTANT: The rmmod command does not list the removed unmanaged objects nor does it include them in the success/failure count. The success/failure count only includes totals for modules.

This option is only valid for workspace module arguments. It is ignored for all other arguments. This option is mutually exclusive with the -keep option.

-vaultdata

-[no]vaultdata

Specifies whether the vault folder in which the legacy module contents reside should be removed. (Legacy modules only)

-novaultdata leaves the vault folder containing the legacy module contents on the server. (Default)

-vaultdata deletes all objects, including directories and locked files, within the vault.

Notes:
- The vaultdata option only applies to legacy modules, in the current module schema, the vault is an integrated part of the module

```
                          and is always removed with the module.
                        - If -vaultdata is specified, the rmmod operation
                          calls the rmfolder operation to delete the vault
                          data. The rmfolder command obeys the
                          RmVaultKeepVid client-side registry key to
                          determine if version information for the deleted
                          files is retained. For more information, see
                          the "Advanced Registry Settings" topic in the
                          DesignSync Data Manager User's Guide.
                        - If you chose not to remove the vault folder,
                          the following occurs:
                          o showconfs continues to display the
                            configurations for the deleted legacy
                            module.
                          o if the module resides in the /Projects
                            folder, the legacy module and its
                            configurations appear in ProjectSync as a
                            ProjectSync project.

                        If this option is used incorrectly, it is
                      silently ignored.
```

-xtras

```
-xtras <string>       List of command line options to pass to the
                      external module change management system. Any
                      options specified with the -xtras option are
                      sent verbaGtim, with no processing by the
                      rmmod command, to the Tcl script that defines
                      the external module change management system.
```

RETURN VALUE

```
If the rmmod command is successful, DesignSync returns an empty
string ("").  If the command fails, DesignSync returns an error
explaining the failure.
```

SEE ALSO

```
mkmod, rmfolder, command defaults
```

EXAMPLES

- Example of Removing a Module from a Server
- Example of Removing a Module Hierarchy from a Workspace
- Example of Removing a Module from a Workspace
- Example of Removing a Legacy Module from a Server

- Example of Removing a Module Cache Link from a Workspace

The following examples show removing modules from DesignSync.

Example of Removing a Module from a Server

This example removes the RAM module from the DesignSync server, sync://srvr2.ABCo.com.

```
dss> rmmod -report verbose sync://srv2.ABCo.com:2647/Modules/RAM

 Beginning rmmod operation ...

 sync://srvr2.ABCo.com/Modules/RAM: Removing module from server ...
 sync://srvr2.ABCo.com/Modules/RAM: Removed database entry.
 sync://srvr2.ABCo.com/Modules/RAM: Deleted module folder.
 sync://srvr2.ABCo.com/Modules/RAM: Detached all notes.

 Finished rmmod operation.
```

Example of Removing a Module Hierarchy from a Workspace

This example shows removing a module hierarchy in report normal (default) mode from the workspace.

```
dss> rmmod -recursive Chip%4
 c:/Workspaces/DesignSync/Chip/CPU/ROM/ROM%3: Removed module from workspace.
 c:/Workspaces/DesignSync/Chip/CPU/CPU%0: Removed module from workspace.
 c:/Workspaces/DesignSync/Chip/Chip%4: Removed module from workspace.
 {Objects succeeded (3)} {}
```

Example of Removing a Module from a Workspace

This example removes the RAM module from the DesignSync workspace using report verbose..

```
dss> rmmod -report verbose RAM%0

 Beginning rmmod operation ...

 /home/rsmith/MyModules/ram/RAM%0: Current working directory is
   within the base directory of the module being removed.
   Changing directory to /home/rsmith/MyModules ...
 /home/rsmith/MyModules/ram/RAM%0: Removing workspace module content ...
 /home/rsmith/MyModules/ram : Folder deleted from workspace and
   workspace module.
 /home/rsmith/MyModules/ram/RAM%0: Removing workspace module metadata ...
 /home/rsmith/MyModules/ram/RAM%0: Workspace module removed.
```

```
   Finished rmmod operation.
```

Example of Removing a Legacy Module from a Server

```
 This example removes the BIST module from the DesignSync
 server, sync://srvr2.ABCo.com.

  dss> rmmod -vaultdata sync://srvr2.ABCo.com:2647/Projects/BIST

 In this example, the rmmod command:
 o Deletes all configurations of the module.
 o Deletes vault data associated with module.
 o Detaches all of the ProjectSync notes attached to the module and
   any of its configurations.
 o Retains the latest version ID of all removed vaults (depending
   on the value of the DesignSync RmVaultKeepVid client-side
   registry setting).
```

Example of Removing a Module Cache Link from a Workspace

```
 This example removes the 300MM module cache link from a workspace.
 It uses the instance name of the mcache link.
 Note:  It does not remove the module from the server or from the
 mcache directory.

 stcl>  rmmod -report verbose 300MM%0

  Beginning rmmod operation ...

  /home/rsmith/MyModules/300mm/300MM%0: Removing workspace module metadata
...
  /home/rsmith/MyModules/300mm: Removing mcache symlink ...
  /home/rsmith/MyModules/300mm: Mcache symlink removed.
  300mm: Success deleted

  Finished rmmod operation.
```

rmversion

rmversion Command

NAME

```
 rmversion           - Deletes versions from the vault
```

DESCRIPTION

- Notes for Modules (Module-based)

- Removing Orphaned Module Members (Module-based)

```
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)
- -force (File-based)
- -report (Module-based)
- -report (File-based)
- -[no]scrub (Module-based)
- -trigarg
- --

-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
```

rollback

rollback Command

NAME

```
rollback            - Reverts a module back to a previous version,
                      rolling back any structural changes
```

DESCRIPTION

```
The rollback command provides a mechanism for making an earlier
module version the "Latest" module version, "rolling back" any
changes to the previous state. All module versions between the
rollback versions remain in the vault.

The rollback command creates a new module version using the next
available module version number to store the rollback version which
is now tagged "Latest".

Because the rollback command exactly reinstates the module as it
was, any structural changes to the module, such as added or removed
files or hrefs between the rollback versions are removed.  If files
```

were added between the versions, those files are removed. If files were removed, those files are added back. Any new hrefs are removed. Any removed hrefs are reinstated.

Notes:

- The href is reinstated even if the target no longer exists. You should verify the consistency of the hrefs after the rollback operation has completed.
- If a renamed or removed file contains characters in the name or the object, or the object natural path, that were previously legal, but are now restricted characters which can not be used by the system, the rollback of the object(s) succeeds. After the rollback operation completes, the user should fix the object name or natural natural path to remove characters that have been disallowed. For more information on restricted characters, see Exclude Lists in the DesignSync Data Manager Administrator's Guide.

Rollback is only available for modules. To revert to any single module member, or DesignSync object to an earlier version, populate a workspace with the desired version of the object, and check in the object using the -skip option. To add a removed module member back to the module without rolling back all the changes made since the remove, or populating an earlier module member version, use the unremove command.

Note: The rollback command is not applicable to module snapshots.

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

SYNOPSIS

```
rollback [-[no]comment ["text"]] [-version <selector>]
[--] <argument>
```

ARGUMENTS

- Server Module Version

Server Module Version

<server module version>	Specify the URL as follows: sync[s]://<host>[:<port>]/<vaultPath>[;<selector>] where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, <vaultPath> identifies the module, and <selector> is the branch and version information. You may use this format to specify a module, module branch or module version. The default branch is "Trunk." The default version is "Latest".

Note: If you use the -version option, you do not need to specify the version as part of the URL.

OPTIONS

- -[no]comment
- -version

-[no]comment

-[no]comment ["<text>"]	Specifies whether a text description of the reason for the module rollback is stored with the module. -nocomment performs the rollback 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. If -comment is specified without text, or set as the default, 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."

-version

-version <selector>	Specifies the version of the module being rolled back. If no version is specified here or in the argument, the default version is "Latest" on the "Trunk" 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.

RETURN VALUE

If the rollback command is successful, DesignSync returns an empty string (""). If the command fails, DesignSync returns an error explaining the failure.

SEE ALSO

ci, populate, vhistory, unremove, setselector

EXAMPLES

This example shows how rollback can be used to undo mistakes made to a module. In this example the project leader thought an error was introduced in version 1.3 and then later realized that 1.3 was correct.

```
dss> rollback -comment "Reverting to last good version" \
     -version 1.2 sync://srv2.ABCo.com:2647/Modules/ROM
New version 1.5 was created by rollback sync:///Modules/ROM;1 to the
version 1.2
```

The next version created is version 1.6. This example shows checking in a modification to one of the files and updating the module to version 1.6.

```
dss>  ci rom.doc

 Beginning Check in operation...

 Checking in objects in module ROM%0

 Total data to transfer: 1 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
 Progress: 1 Kbytes, 1 file(s), 0 collection(s), 100.0% complete

 Checking in: /rom.doc                        Success - New version: 1.3

 ROM%0: Version of module in workspace not updated due to auto merge

 Finished checkin of Module ROM%0, Created Version 1.6

 Time spent: 0.4 seconds, transferred 1 Kbytes, average data rate 2.6 Kb/sec

 Checkin operation finished.

  {Objects succeeded (1)} {}
```

Note: Because rollback preserves all changes, you could rollback your module version to one of the versions that was removed from the active code stream by a previous rollback. For example, you could roll back to version 1.3 or 1.4 to pick up those changes. It removes the 1.6 changes from the module.

```
dss> rollback -comment "These changes should not have been removed" \
     -version 1.3 sync://srv2.ABCo.com:2647/Modules/ROM
 New version 1.7 was created by rollback sync:///Modules/ROM;1 to the
 version 1.3
```

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>
```

setfilter

setfilter Command

NAME

```
setfilter           - Sets the persistent filter or hreffilter list
```

DESCRIPTION

```
This command sets the persistent filter or hreffilter for a
module. This filter is applied each time the module is populated.
The persistent filters defined here are applied to the appropriate
commands before any filters or hreffilters specified on the command
line are applied.

If a module is initially populated using a -filter or -hreffilter on
the command line, a persistent filter matching those settings is set
automatically for that module.

After a filter has been changed using the setfilter command, the next
populate of the module is a full populate, since the filter has
changed. Performing a setfilter replaces any previous filters set,
including the filters set automatically with a filtered populate,
with the new filter.
```

SYNOPSIS

```
setfilter [-filter | -hreffilter] [-[no]recursive][--]
          <workspace module> <filter>|<hreffilter>
```

ARGUMENTS

- Workspace Module
- Filter
- Hreffilter

Workspace Module

```
<workspace module>  Specify the module identifier for the module
                    receiving the persistent filter.  The module must
                    have already been populated in the workspace.
```

Filter

<filter> Specify one or more extended glob-style expressions to identify an exact subset of module objects on which to operate. The expressions should be separated by commas, for example:
+top*/.../*.v,-.../a*

If you specify a null character ("") as the filter argument, all filter values are removed from the persistent filter list including the filters created during a filtered populate. The next time the directory is populated, DesignSync performs a full populate.

Prepend a '-' character to a glob-style expression to identify objects to be excluded.(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'.

Hreffilter

<hreffilter> Excludes href values during recursive populate of module hierarchies, excluding particular submodules from the populate. Note that unlike the -filter option which lets you include and exclude items, the -hreffilter option only excludes hrefs.

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.

Note: You can set both types of hreffilters, simple and hierarchical, for your workspace, but they must set in different operations.

A simple href filter is a simple leaf module name or the href name (specified when you added the href). 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. When you specify a simple href, you must run the setfilter command in -norecursive mode (Default).

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 you specify a hierarchical

href filter, you must run the setfilter command in -recursive mode.

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.

OPTIONS

- -filter
- -hreffilter
- -recursive
- --

-filter

-filter — Specifies that the persistent filter being set is a filter argument, not an href filter. Filter arguments can both exclude or include elements.

-hreffilter

-hreffilter — Specifies that the persistent filter being set is an hreffilter argument which prevents the updating of the specified hrefs during a populate operation.

-recursive

-[no]recursive — Specifies whether the persistent filter is applied recursively through the module hierarchy.

-norecursive does not apply the persistent

```
                    filter recursively (Default). This is the
                    standard operating mode for filters and simple
                    href filters.

                   -recursive applies the persistent filter
                    recursively through the module hierarchy.  This
                    is the required mode when using hierarchical href
                    filters.

                    Note: When setting or removing hreffilters, only
                    one type of hreffilter, simple or hierarchical,
                    may be set at a time because they require
                    different -recursive/-norecursive 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

```
If the set filter command is successful, DesignSync returns an empty
string ("").  If the module does not exist in the workspace or the
filter cannot be set, the setfilter commands returns an error.
```

SEE ALSO

```
populate, url filter
```

EXAMPLES

- Example of setting a filter
- Example of setting an href filter
- Example of setting a hierarchical href filter
- Example of clearing an href filter
- Example of clearing a hierarchical href filter

Example of setting a filter

```
This example shows setting a filter on your module, Chip, to filter
out documentation files. After setting the filter, you may want to
populate to bring the changes into your workspace.

dss> setfilter -filter  Chip%0 -.../doc/.../*
```

```
 Set Filter operation successfully completed.
```

Example of setting an href filter

```
This filters out any submodule named BIN from your Chip module
hierarchy. After setting the hreffilter, you may want to populate to
bring the changes into your workspace.

dss> setfilter -norecursive -hreffilter Chip%0 BIN
 Set Filter operation successfully completed.
```

Example of setting a hierarchical href filter

```
This filters out the JRE/BIN submodule hierarchy within
the Chip hierarchy.  This operation is recursive through the module.
After setting the hierarchical hreffilter, you may want to populate
to bring the changes into your workspace.

dss> setfilter -recursive -hreffilter Chip%0 JRE/BIN
 <Chip%0> Persistent hierarchical href filters set to <JRE/BIN>.
 Set Filter operation successfully completed.
```

Example of clearing an href filter

```
This example removes all (simple) href filters set on the Chip module.

dss> setfilter -norecursive -hreffilter Chip%0 ""
 <Chip%0> Persistent href filter cleared. It will no longer be used.
 Set Filter operation successfully completed.
```

Example of clearing a hierarchical href filter

```
This example removes all hierarchical href filters set on the Chip
module.
dss> setfilter -recursive -hreffilter Chip%0 ""
 <Chip%0> Persistent hierarchical href filter cleared. It will no
longer be used.
Set Filter operation successfully completed.
```

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>
```

unfreezemod

unfreezemod Command

NAME

```
unfreezemod           - Releases access controls on a frozen module
```

DESCRIPTION

```
This command releases the access controls on a frozen module so
that changes can be made to the module, freeing it up for normal use.

This command supports the command default system.

This command is subject to access controls on the server.
```

SYNOPSIS

```
unfreezemod <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
```

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

```
freezemod, mvmod, importmod, exportmod, edithrefs
```

EXAMPLES

- Example of Unfreezing a module

Example of Unfreezing a module

```
This example shows unfreezing a module.

dss> unfreezemod sync://serv.ABCo.com:2647/Modules/Chips/chip-nx1
 sync://qelwsun14:30126/Modules/Chips/chip-nx1 : Module is unfrozen.
```

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 (File-based)
- -exclude
- -modulecontext (Module-based)
- -[no]recursive (Module-based)
- -[no]recursive (File-based)
- -[no]selected
- -trigarg
- --

-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.
```

unremove

unremove Command

NAME

```
unremove            - Restores removed files to a module
```

DESCRIPTION

- Understanding the Output

```
The unremove command provides a way to locate and recreate an object
that has been removed from a module. The object is restored into your
local workspace and automatically Added to the module, to be checked
in with the next module creation operation.

Use the unremove command when you do not know which module version
contains the object being re-added to the module. The command has the
ability to locate member objects in module version that do not share
a common ancestor with the current workspace version.  This allows
you to unremove a member that was never created on the workspace
module branch.

This command is used for a single object.

The object must be restored to its original natural path.  After the
object has been checked in, it may be moved or renamed.
```

Understanding the Output

```
The unremove 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 unremove
command alerts the user to any unusual events, such as failures or
warnings, and the Success/failure/skip status.

By default, or if you run the unremove command with the '-report normal'
option, the command displays all the information contained in -report
brief, and standard report information, such as the list of objects
added back to the module.

If you run the unremove command with the '-report verbose' option, it
displays all the information contained in -report normal and any
```

```
operation steps taken or decisions made.

If you run the unremove command with the -report error option, it displays
the following information:
 o Failure messages.
 o Warning messages.
 o Success/failure/skip status.

 Note: If an object is explicitly specified and is already part of a
 module, you will see an error stating that the object was skipped.
 If an object is included in a recursive operation and is already
 part of a module, you will not see the error, it will be silently
 skipped.

This command supports the command defaults system.

This command is subject to access controls.
```

SYNOPSIS

```
unremove [-modulecontext <context>]
         [-report error|brief|normal|verbose] [-version <selector>]
         <member-path>[;<member-version>]
```

ARGUMENTS

- Member Path

Member Path

<member path> [;<member-version>]	Specify the path of the removed object. You may optionally, specify the specific member version of the removed object. If no version is selected, the version that was Latest at the time of the remove being undone is used. You can specify a modulecontext for the argument, or, if no modulecontext is specified, DesignSync uses the local metadata in the workspace to determine which module to operate on.

OPTIONS

- -modulecontext
- -report
- -version

-modulecontext

-modulecontext <context>	Identifies the module on which the unremove operates. Specify the desired module using the 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.

-report

-report error|brief normal|verbose	Determines what information is returned in the output of the unremove command. The information each option returns is discussed in detail in the "Understanding the Output" section above. Valid values are: o error - provides error and warning messages only. o brief - lists all the objects unremoved (added back) for the workspace module. o normal - indicates when the command begins and ends processing and lists all the objects unremoved for the workspace module. (Default) o verbose - provides full status for each object processed.

-version

-version <selector>	Specifies the version of a module to begin searching for the module member to unremove. If no version is specified, DesignSync uses the version loaded in the workspace. (Default) 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 an objects to check into the local workspace branch that was previously not present on the branch.

RETURN VALUE

By default, this command returns a count showing how many objects

```
 succeeded and failed.
```

SEE ALSO

```
remove, add, ci
```

EXAMPLES

```
This example shows unremoving a module member from a workspace.
dss>  unremove -modulecontext Chip%5 chip.doc

 Determining the module version from which to restore the member ...

 Populating member from module version 1.2 ...

 Beginning populate operation at Fri Sep 17 03:52:08 PM EDT 2010...

 Populating objects in Module          Chip%5
                       Base Directory  /home/rsmith/workspaces/chip
                       Without href recursion

 Fetching contents from selector '1.2', module version '1.2'

 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

 Chip%5/chip.doc : Success - Checked out version: 1.1
 Chip%5 : Version of module in workspace not updated (Due to not
 operating on entire module contents).

 Finished populate of Module Chip%5 with base directory
 /home/rsmith/chip

 Time spent: 0.2 seconds, transferred 1 Kbytes, average data rate 5.7
 Kb/sec

 Finished populate operation.

  {Objects succeeded (1)} {}

 Add chip.doc to module Chip%5 ...

 Beginning add operation...

 /chip.doc: Adding object back to module

 Finished add operation.

  {Objects succeeded (1)} {}
```

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
```

upgrade

upgrade Command

NAME

```
upgrade             - Upgrades a DesignSync vault folder or a legacy
                      module to the current module structure
```

DESCRIPTION

- The Upgrade Process
- Module Name
- Module Branches
- Module Versions
- Migrating Module Tags
- Hierarchical References
- Hierarchical Reference Names
- Hierarchical Reference Types
- Hierarchical Reference Static Versions
- ProjectSync Module Notes and Subscriptions
- Access Controls
- The ModuleUpgrade Directory
- Post-Upgrade Tasks
- Understanding the Output

```
This command upgrades a DesignSync vault folder or a legacy module
```

to the current module structure.

Note: For ease of use, the documentation for upgrade refers to data from either a DesignSync vault or a legacy module as legacy data or a legacy object unless the information applies specifically to a DesignSync vault or legacy module.

Important: Before you upgrade legacy data, you must check in any modifications and release any locks.

During sync_setup, while configuring the server, DesignSync verifies that the "Modules" directory is configured for the current module structure. If the directory is not configured for the current module structure, DesignSync sets a registry flag to indicate that the upgrade command cannot upgrade legacy data to the new module format. When upgrade is run on the server, the command fails with a note explaining that there is legacy data in the Modules directory. For information on removing the legacy data, see the ENOVIA Synchronicity DesignSync Installation available from the Program Directory along with the release information. The registry flag also controls the ability to create modules with mkmod.

Notes:

- The legacy object is only upgraded on the server and is placed in a new location in the Modules vault directory on the server. All existing workspaces continue to point to the legacy data even after the upgrade occurs. To use the new module, the users need to populate a new workspace with the new module data.

- Previous versions of DesignSync recommended that the following special characters, which are prohibited in module names, also not be used in the natural path of module members (for module member or folder names):

  ~ ! @ # $ % ^ & * ( ) , ; : | ` ' " = [ ] / \ < >

  The site or server administrator can now restrict any or all of these characters from being used, however, the upgrade process does not modify folder or module members names during processing. After processing, if your folder or module members contain these special characters, you will need to manually modify the names to remove the prohibited characters. For a list of the characters disallowed on your system, view the Exclude List pane in the DesignSync Administrator.

The upgrade command can be run multiple times without harming your legacy data. For example, if legacy data is still being updated, you can run the upgrade command to re-upgrade the data. The data is not upgraded on top of the previous upgrade. You would have to either remove the upgraded module (with the rmmod command) before running the upgrade again, or provide a different module name for the new upgrade.

Note: If an object is upgraded multiple times, the last module it was upgraded to is used when mapping the hierarchical references.

The upgrade command creates a new module from the specified legacy object. All legacy vaults are copied to the new module location as SmartVaults.

The upgrade command creates the following module elements:

- A module with the specified name (optionally in the category path, if provided).
- Module branches
- Module versions
- Module tags
- Hierarchical references

The module elements, in large part, come from the contents of sync_project.txt files. The upgrade process traverses the legacy object hierarchy, to create the branches, versions and tags based on the data in the sync_project.txt files. For DesignSync vaults, the hierarchical references derive from the REFERENCE statements in the sync_project.txt files. For legacy modules, the hierarchical references are carried over.

Tip: After running the upgrade, you can run the migratetag command to port all your existing files-based version tags to your newly created module. For more information, see the migratetag command.

Module Name

The first thing the upgrade creates is the new module. The name of the module can be specified using the -name argument. If the module name is not specified, the module is created using the leaf node name of the legacy object. The combination of module and category name must be unique for each module. If you use the same name for more than one module, the modules must have different category names.

Module categories are virtual folders used to organize modules and group modules into related groups. Specific usage is dependent on your site setup, but categories can organize modules based on engineering groups, hardware or software functions, or other classifications. There is no limit placed by DesignSync on the number of categories or sub-categories a site may have. The category for the new module is specified with the -category option.

Tip: Use module names beginning with an initial capital letter. This provides an easy method of distinguishing between a folder name, which is conventionally lower-case, and the module.

Module names cannot contain the following special characters:
~ ! @ # $ % ^ & * ( ) , ; : | ` ' " = [ ] / \ < >

Note: The following two characters, back quote (`) and slash (/) were allowed in legacy module names, but are no longer valid characters. If you upgrade a legacy module or DesignSync vault containing either of these two characters, you must specify a valid name with the -name

option or the upgrade will fail.

Module Branches

The default module branch 1 is created and is tagged with the immutable module branch tag named "Trunk". Since the tag is immutable, it can not be changed. The Trunk branch of the DesignSync vault or legacy module is created as branch 1 (Modulename.1) of the new module. This branch is automatically tagged with a "Trunk" tag. This tag is immutable meaning it can not be changed. Even if there are no objects in the legacy data with the Trunk branch tag, the Trunk branch is still created with version 1.1, but contains no module members.

The upgrade process creates a module branch for each configuration present in the sync_project.txt files of the legacy object. Each branch stems from module version 1.1. The configuration name is added to the module branch as a mutable module branch tag. If the configuration is mapped to a branch selector such as, "Branch:" or "Branch:Latest," then the branch name is also added as a mutable module branch tag, unless the name already exists as another configuration.

Branch for legacy module release configurations:

If the legacy module has any "release" configurations, a "Releases" branch, branched from module version 1.1, is created. The mutable module branch tag "Releases" is added to this branch. The legacy module "releases" and "aliases" are placed on this branch.

Module Versions

Version 1 is created on each module branch. This is the only version created on a branch during upgrade, with the exception of the "Releases" branch, where multiple module versions may be created.

Version 1 is created on the default "Trunk" branch. This is module version 1.1. This version contains all modules members matching the Trunk:Latest selector.

NOTE: If a "Trunk" configuration is not mapped to "Trunk:Latest", the upgrade command still creates module version 1.1 with member versions that resolve to "Trunk:Latest". If you want to migrate the Trunk legacy configuration you may either use the migratetag utility to create a different branch containing members for the legacy "Trunk" configuration or create a new configuration with this mapping prior to running upgrade. If the object being upgraded is a legacy module, an older client would be required to add the configuration.

If the "Releases" branch is created, a module version is created for each release configuration. The module version contains each member version tagged with the release tag (which indicates that is part of

that release) The release name is added as an immutable module version tag. If there are any alias configurations mapped to this release, then the alias name is added to the module version as a mutable module version tag.

Version 1 is created on all other branches containing the members that resolve to the selector mapped to the configuration. If the selector is a version tag, then the module version is assigned this tag as a mutable module version tag, unless the name already exists as another configuration.

If any branch created has no members matching the selector for the legacy configuration, the module branch is still created; however the first version on the branch contains no members.

Migrating Module Tags

The upgrade command provides automatic migration of the tags associated with the configurations defined in the associated sync_project.txt files. There can be additional tags on versions and branches of the member vaults that are not defined in the sync_project.txt files, such as a private tags used to define a local configuration. The upgrade process preserves this tag information so private tags or selector lists using these tags can be migrated seamlessly to the new module.

In order to migrate the private tags or selectors used with legacy modules or DesignSync vaults to the new module, you should run the migratetags command after the upgrade command has completed.

If you are not sure whether you have tags that you need to migrate with the migratetags command, you can run the migratetag command with the -list option to generate a list of the tags available to be migrated to the new module or module members with their version numbers that would be added to the new branch with the specified selector.

Hierarchical References

By default, all hierarchical references (hrefs) to objects that have been upgraded map to the upgraded module. When an object is upgraded, the information about the legacy configurations, including configuration names and their mapped module branches, versions, and static module versions is retained. This, along with the upgraded module property set on the legacy object, is what allows mapping of hierarchical references to take place.

Note: If you do not want to map hrefs to the upgraded modules, use the -nomaphrefs option to the upgrade command.

If the object being upgraded has hrefs to modules that have not been upgraded, or will not be upgraded, those hrefs continue to point

to the original legacy object locations. If you subsequently decide to upgrade those objects, you must manually remove the href to the legacy object and add the href to the upgraded module.

The new href is added to the module branch that was created from the legacy configuration specified with the fromargument of the legacy module href.

If the hrefs are mapped to upgraded modules, the new hrefs link to the branch and version of the upgraded module specified by the toargument of the legacy module href. If the hrefs are not mapped, the href will continue to point to the legacy object. The legacy objects can be an IPGear Deliverable, a DesignSync vault folder or a legacy module.

If there is no matching module branch for the configuration referenced by the fromargument of the href, a new module branch, along with the first module version on this branch is created. The configuration name is used as the version tag to identify any members that were part of the configuration. The matching members are added to this new module version. If no module members are identified, the href is the only object added to the newly created module version.

Tip: When upgrading legacy data, you should upgrade the leaf submodules or references first, and then work backwards through the hierarchy to convert the top-level module or DesignSync vault directory last. This will allow the upgrade of the upper-level objects to map legacy module hierarchical references or DesignSync references to their upgraded modules when appropriate. You can optionally choose to have the new references refer to the legacy data.

Hierarchical Reference Names

The new hrefs, unlike the legacy module hrefs are identified by unique names. Href names are automatically assigned when the upgrade process creates the href. The href name is unique within the module version. The name defaults to the module name or the leaf name of the object being referenced. If the name already exists within the module version, a unique name is generated by concatenating the module name or leaf name of the referenced object with the next available number, beginning with one (1) in the following format:

```
<moduleName>_<number>
```

For example: the first reference to a module called "Chip" is "Chip." The next reference to a different module named "Chip," is "Chip_1". A reference to a third module named, "Chip," is "Chip_2" etc.

Hierarchical Reference Types

All hrefs are assigned a type to allow for easy identification of the href target.

- Module - href to an upgraded module.
- Alias - href to a legacy module alias.
- Branch - href to a legacy module branch configuration.
- Release - href to a legacy module release configuration.
- Selector - href to a legacy module selector configuration.
- Vault - href to a DesignSync vault.
- Deliverable - href to an IP Gear deliverable.
- Unknown - indicates that the object type could not be determined.

Hierarchical Reference Static Versions

When an href is mapped to an upgraded module, a static version number is assigned to the href. The static version of the sub-module is the module version created for the corresponding legacy object configuration.

ProjectSync Module Notes and Subscriptions

Notes associated with legacy module or DesignSync vault being upgraded are associated with the new module as well.

The following table shows how legacy object ProjectSync attachments are copied to the new module:

Legacy Object location	New object location
Top-level legacy object	New module
Legacy configuration	New module's branch created from the configuration
Release configuration	New module's version associated with the release
Alias	New module's version tagged with the upgraded tag created from the alias

Note: Notes attached to sub-folders of the legacy object or to the vaults that lie beneath the legacy object vault folder are not attached to the new module.

Email subscriptions on the legacy object are not transferred to the new module. Also, subscriptions to the new module need to be added.

After the upgrade is complete, an upgrade RevisionControl note is created and logged in ProjectSync against the upgraded module and legacy object. The note is broadcast to all users subscribed to notes on the legacy object, the new module, upgrade RevisionControl notes or all hcm RevisionControl notes. The upgrade note includes the Sync URL for the new module and the entire contents of the upgrade

log file.

An mkmod note is created for the new module.

Access Controls

During the upgrade process, access controls on the legacy data are added to restrict write access so no changes can be made to legacy data during the upgrade process. This guarantees that the new module will contain all the data present in the legacy object. After the upgrade, the legacy object is left in read-only mode to prevent users from accidentally updating the legacy data instead of the new module. The access controls can be removed after the upgrade completes if further development on the legacy object is desired.

In addition, access controls on the legacy object are not carried over to the new module. During the upgrade, read and write access is restricted for the new module until the upgrade completes. After the upgrade completes, the new module is left with no access controls. Access controls can be added for the new module before or after the upgrade is run.

The ModuleUpgrade Directory

The upgrade command creates the ModuleUpgrade directory as an unmanaged directory in the new module that stores all the supporting information used in module creation as well as maintaining some information about the legacy object that is not carried over.

The upgrade log file, upgrade_<date>_<timestamp>.log, stores the complete output information from the upgrade command. You can review this log file if you need additional information about what occurred during the upgrade. If upgrade fails for any reason, and you rerun upgrade after correcting the source of the failure, upgrade recreates a new module and preserves all prior log files.

Empty directories that exist in the legacy object are not added to the new module. The LegacyEmptyDirs file stores the names of the empty directories. After the upgrade has completed, you can use this list to recreate the original module object directory structure. including empty directories. The file consists of a Tcl list containing the natural path to the empty directory. Only vault folders that are completely empty are carried over. This means that an empty directory that contains shell vaults removed with keepvid or only contains vaults active on some branches is not listed in this file.

Note: The natural path is the path where that object is placed under the module base directory. For instance, in the following sync URL, sync://svrc2.ABCo.com/Modules/Chip/doc/chip.doc, the natural path of the chip.doc file is "/doc/chip.doc".

User properties set on the top-level legacy object are carried over to the module, and set on the module itself. User properties that

are not set on the top-level legacy object, but instead set on an object below the top-level legacy vault directory, are not applied to the new module. The LegacyUserProperties file stores the user property information that is not associated with the new module. After the upgrade has completed, you can use this list to recreate the original module member's user properties. These properties can be for any of the following objects: vault directory, vault file, vault branch or vault version. The file consists of a Tcl list containing the following three elements for each property found:

- Natural path to the member object,
- Property name
- Property value

The sync_project.txt files for legacy objects contain project owner and team member lists that are used for projects and configurations by ProjectSync's SyncUserList fields. New modules do not have the same functionality. In order to preserve owner and team member lists for projects and configurations, the sync_project.txt files for the legacy objects are saved into the LegacyProjectFiles directory. If the legacy object contains multiple sync_project.txt, a naming scheme is used in order to avoid file name clashing.

Post-Upgrade Tasks

After you have completed the migration, you should perform the following tasks:

- Run the migratetag command to port any existing files-based tags to the new module.
- Check the module member and folder names to verify that the natural path of the module members confirms with the naming restrictions in place on your system. Your system or site administrator can restrict the character set used by DesignSync to avoid special characters, such as "#" or "@." You can check which characters are not permitted on your system by viewing the Exclude List panel in the DesignSync Administrator (SyncAdmin) application.
- Set the appropriate Access Controls for the new module. By default, the old file-based data is left in a locked state. If you are still developing that module, perhaps to maintain it for minor or legacy releases, update the Access Controls to allow modifications.
- Provide users with the new information to connect to the module.

Understanding the Output

By default, or if you run the upgrade command with the '-report normal' option, the command displays the following information:

```
        o Beginning upgrade message
        o Status message for each phase of the upgrade
        o Name of the legacy object being upgraded to the new module
          structure
        o Name of each new Module branch and version being created and
          the source of the information: for example:
          the configuration and the selector defined in the sync_projects
          file.
        o Information about the tags added to the module.
        o Information about hrefs, including if the legacy object
          references were mapped.
        o Sync URL for the newly created Module


If you run the upgrade command with the '-report brief' option, it
displays the following information:

        o Beginning upgrade message
        o Vault path of the legacy object being upgraded to the new
          module structure
        o Sync URL for the newly created Module

If you run the upgrade command with the '-report verbose' option, it
displays all the information available in normal mode, plus the
following information:

        o The member versions that were added to each module version.
        o Information about the ProjectSync revision control notes
          generated for the upgrade and the associated mkmod used to
          create the new module.
        o The notes that were attached to the new 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.
```

SYNOPSIS

```
upgrade [-category <path>] [-[no]maperror] [-[no]maphrefs]
        [-name <modulename>] [-report {brief|normal|verbose}]
        <argument>
```

ARGUMENTS

- Legacy Module URL
- DesignSync Vault URL

Legacy Module URL

```
<Module URL>           Specifies the legacy module you want to
```

upgrade to the new modules structure.
Specify the URL as follows:
sync[s]://<host>[:<port>]/Projects/<path>
where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, Projects is the root directory, and and <vaultPath> identifies the specific legacy module to upgrade.

DesignSync Vault URL

<DesignSyncVault URL> Specifies the URL of the DesignSync vault you want to upgrade to the new module structure.

Specify the URL as follows:
sync[s]://<host>[:<port>]/<vaultPath>
where <host> is the SyncServer on which the DesignSync vault resides, <port> is the SyncServer port number, and <vaultPath> identifies the DesignSync vault you want to convert to a module.

OPTIONS

- -category
- -[no]maperror
- -[no]maphrefs
- -name
- -report

-category

-category <path> Specifies a category in which to store the new module. A category is a virtual directory that allows you to group Modules by a common characteristic. You may specify any number of sub-categories.
Using a category provides a number of benefits such as:
o Allowing you to have multiple modules with the same name for different uses.
o Providing a logical organization structure for grouping modules.

-[no]maperror

-[no]maperror This option works in conjunction with the

maphrefs option. This option determines how the upgrade process responds if a mapping error occurs. This option is ignored if -nomaphrefs is specified.

-nomaperror allows upgrade to continue upgrading the legacy data if href mapping errors occur during the upgrade. It reports the mapping errors as warnings and preserves them in the log file.

-maperror stops the upgrade, after all hrefs are processed, when a mapping error occurs. (Default)

-[no]maphrefs

-[no]maphrefs

Determines whether existing existing DesignSync references or legacy module hrefs will attempt to be mapped to an upgraded module. The reference or href can only be mapped if the target object was upgraded. If mapping is not done, new hrefs are still created referencing the target object of the DesignSync reference or legacy module href. The default is -maphrefs.

References to IP Gear deliverables remain as is after the module is upgraded.

An href might not be mapped to a new module for the following reasons:

- The legacy object has not been upgraded to a new module.
- The referenced module was upgraded, but it has since been deleted, moved to a different server, or the server is inaccessible.
- The original reference can not be accessed.
- The reference is to an IPGear deliverable. No conversion is required.

If an href is not mapped to a new module during the upgrade for any reason, you can manually update it later by removing the old href and adding an href to the upgraded module.

-name

-name <modulename>

Name of the new module. The module name should start with an initial capital letter.

Module names must conform to the following standards:
o Must contain only printable characters
o May not contain spaces
o May not contain any of the following characters:
~ ! @ # $ % ^ & * ( ) , ; : | ` ' " = [ ] / \ < >

If no module name is specified, DesignSync uses the leaf folder name for module or vault being upgraded.

Important: The full module name (category and module name together) must be unique on the server. For instance, if you have several projects using different modules which are all named Chip, specify a project identifier using the -category option and Chip using the -name option.
If no name is specified, and the leaf folder containing the legacy module or vault contains illegal characters, or illegal characters are specified with -name, the upgrade command fails and the module is not upgraded.

Note: Specifying a module name with an initial capital letter allows you to easily distinguish between a Module, which by convention always begins with an initial capital letter, and a workspace path, which by convention begins with a lower-case letter.

-report

-report brief|normal|verbose

Determines what the information is returned in the output of the upgrade command. The information each option returns is discussed in detail in the "Understanding the Output" section above.
Valid values are:

o brief - notes the time the upgrade began, and lists the path of the object being upgraded and the URL of the new module created.

o normal - provides information on each phase completed, information about each module branch and version created, and information about the tags and hrefs added to the module. (Default)

o verbose - provides full status for each

```
                        stage of the upgrade.
```

RETURN VALUE

```
 This command does not return Tcl values.
```

SEE ALSO

```
 access allow, access deny, addhref, mkmod, rmhref, migratetag,
 edithrefs, command defaults
,
```

EXAMPLES

```
 This example shows upgrading your Projects/Block0 legacy module. Default
 reporting (normal) is used along with mapping of hrefs, which is also
 the default.

 Note: If server customizations have been imported to a different
 server after a module upgrade, the upgrade log may not appear at the
 URL provided for monitoring the upgrade.  If the URL is not found,
 you may still access the upgrade log from the ModuleUpgrade
 directory for the module.

stcl> upgrade -name Block0 -cat ASIC_TOP sync://sting:30046/Projects/Block0

[Mon Jan 15 07:01:27 PM EST 2007]
  Starting upgrade of sync://sting:30046/Projects/Block0


 You can monitor the upgrade process using the following URL in your
 browser:
 http://sting:30046/syncserver/upgrade/upgrade_ASIC_TOP_Block0.html

 [Mon Jan 15 07:01:27 PM EST 2007] Locating references on the server ...


 Mapped reference to: sync://sting:30046/Modules/Support/tf13
 Original selector: (default) is mapped to Trunk: with static version
  number: 1.1

 Mapped reference to: syncs://sting:30047/Modules/ASIC/ALU1
 Original selector: (default) is mapped to Trunk: with static version
 number: 1.1

 Mapped reference to: syncs://sting:30047/Modules/ASIC/ALU1
 Original selector: Gold is mapped to Gold: with static version
  number: 1.1.3.1
```

```
 [Mon Jan 15 07:01:29 PM EST 2007] Mapping of references ...

 Contacting server to get href mappings and href type for
  syncs://qewflx7:30047/Projects/MEM2 ...

 Mapped reference to: syncs://qewflx7:30047/Modules/ASIC/MEM2
 Original selector: Bronze is mapped to Bronze: with
  static version number: 1.1.2.1


 Done getting href mappings for syncs://qewflx7:30047/Projects/MEM2.

 Contacting server to get href type for sync://poulenc:30046/Projects/DSVF
...

 Contacting server to get href mappings and href type for
   syncs://qewflx7:30047/Projects/MEM1 ...


 Mapped reference to: syncs://qewflx7:30047/Modules/ASIC/MEM1
 Original selector: Bronze is mapped to Bronze: with
  static version number: 1.1.2.1


 Done getting href mappings for syncs://qewflx7:30047/Projects/MEM1.

 Contacting server to get href mappings and href type for
  syncs://qewflx7:30047/Projects/MEM0 ...


 Mapped reference to: syncs://qewflx7:30047/Modules/ASIC/MEM0
 Original selector: gamma-alias is mapped to gamma-alias with
  static version number: 1.1.1.3


 Done getting href mappings for syncs://qewflx7:30047/Projects/MEM0.

 Contacting server to get href type for
  sync://poulenc:30046/Projects/Block42@Gold
  sync://poulenc:30046/Projects/Block42 ...

 [Mon Jan 15 07:01:31 PM EST 2007] Creating module on the server ...


 Creating module from sync://sting:30046/Projects/Block0 ...

 Adding Upgrade ACs to restrict write access for original object.
 ACs added to
 /home/larry/sync_custom/servers/sting/30046/share/AccessControl.
 Adding Upgrade ACs to restrict read access for new module.
 ACs added to
 /home/larry/sync_custom/servers/sting/30046/share/AccessControl.
 Resetting AccessControls ...

 Converting vaults ...

 Creating legacy tags database ...
```

```
=======================================

Creating the Default branch "Trunk" with selector "Trunk:" ...

Adding href named "TF23" referencing sync:///Projects/TF/TF23
Selector: (Default)
Relative path: TF23
Static version number:
Href type: Vault

Adding href named "MEM2" referencing
syncs://qewflx7:30047/Modules/ASIC/MEM2
Selector: Bronze:
Relative path: MEM2
Static version number: 1.1.2.1
Href type: Module

Adding href named "DSVF" referencing
sync://poulenc:30046/Projects/DSVF
Selector: (Default)
Relative path: dsvf
Static version number:
Href type: Vault

Adding href named "240" referencing
sync://desprez:30012/Deliverable/240
Selector: (Default)
Relative path: ipgdeliv
Static version number:
Href type: Deliverable

Adding href named "tf13" referencing sync:///Modules/Support/tf13
Selector: Trunk:
Relative path: TF13
Static version number: 1.1
Href type: Module

Adding href named "MEM1" referencing
syncs://qewflx7:30047/Modules/ASIC/MEM1
Selector: Bronze:
Relative path: MEM1
Static version number: 1.1.2.1
Href type: Module

Adding href named "MEM0" referencing
syncs://qewflx7:30047/Modules/ASIC/MEM0
Selector: gamma-alias
Relative path: MEM0
Static version number: 1.1.1.3
Href type: Module

Adding href named "Block42" referencing
sync://poulenc:30046/Projects/Block42
Selector: Gold
Relative path: block42
Static version number:
```

```
 Href type: Branch

 Adding immutable module branch tag "Trunk"

 Default module branch "Trunk" and module version "1.1" created.
 Unique id "a33d77fbcd69141d0af3daa65c8ffffe" assigned to module.

 Locating RevisionControl notes with sync:///Projects/Block0@Trunk
 and adding sync:///Modules/ASIC_TOP/Block0;Trunk: to the existing Objects
list
 No RevisionControl notes found with original object affected.

 Locating notes attached to sync:///Projects/Block0@Trunk
 and attaching them to sync:///Modules/ASIC_TOP/Block0;Trunk:
 No notes found attached to original object.

 ========================================

 Creating module branches for configurations ...

 --------------------
 Creating module branch "NMGold" ...

 Adding href named "MEM2" referencing
 syncs://qewflx7:30047/Modules/ASIC/MEM2
 Selector: Bronze:
 Relative path: MEM2_G
 Static version number: 1.1.2.1
 Href type: Module

 Adding href named "ALU1" referencing sync:///Modules/ASIC/ALU1
 Selector: Trunk:
 Relative path: ALU1_G
 Static version number: 1.1
 Href type: Module

 Adding href named "TF23" referencing sync:///Projects/TF/TF23
 Selector: (Default)
 Relative path: TF23
 Static version number:
 Href type: Vault

 Adding href named "ALU1_1" referencing sync:///Modules/ASIC/ALU1
 Selector: Gold:
 Relative path: ALU1_G
 Static version number: 1.1.3.1
 Href type: Module

 Adding href named "Block42" referencing
 sync://poulenc:30046/Projects/Block42
 Selector: (Default)
 Relative path: BL42G
 Static version number:
 Href type: Branch


 Adding mutable module branch tag "NMGold"
```

```
Module branch "1.1.1" created.

Locating RevisionControl notes with sync:///Projects/Block0@NMGold
and adding sync:///Modules/ASIC_TOP/Block0;NMGold: to the existing
Objects list
Added objects to 6 notes.

Locating notes attached to sync:///Projects/Block0@NMGold
and attaching them to sync:///Modules/ASIC_TOP/Block0;NMGold:
Attached 6 notes.

Locating RevisionControl notes with sync:///Projects/Block0
and adding sync:///Modules/ASIC_TOP/Block0 to the existing Objects list
Added objects to 45 notes.

Locating notes attached to sync:///Projects/Block0
and attaching them to sync:///Modules/ASIC_TOP/Block0
Attached 42 notes.


New module completed: sync://sting:30046/Modules/ASIC_TOP/Block0

Resetting AccessControls ...


NOTE: ACs to restrict write access to sync:///Projects/Leg8Block0
      were left in place in the server AccessControl file.
      /home/larry/sync_custom/servers/sting/30046/share/AccessControl
```

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
- -[no]collection
- -[no]comment
- -localtmpdir
- -[no]new
- -report (Module-based)
- -report (File-based)
- -servertmpdir
- -tag
- -vault (Module-based)
- -vault (File-based)
- -workspace

-branch

```
 -branch              Specifies the branch on which to place the
  <branchname>        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
```

Module Hierarchy Management

Module Swapping

swap

swap Command

NAME

```
swap                - Commands enabling edit-in-place of sub-module
```

DESCRIPTION

- populate of a swapped sub-module
- ci of a swapped sub-module

```
The swap commands allow a module version that has been populated by an
href to be manually replaced by another. This is analogous to allowing
a brick to be removed from a wall and replaced (in the same location)
with a different version of the same brick. A primary use of this
edit-in-place methodology is to replace a statically fetched
sub-module within a baseline (i.e. static) module hierarchy with the
latest version on a branch so that the sub-module can be developed
within a baseline framework.

The swap capabilities:
o Change the selector of a sub-module already present in the workspace
```

```
  and re-populate it recursively using the new selector. This swaps
  the entire sub-module hierarchy.
o Avoid reverting the sub-module via a recursive populate of a parent
  module.

This results in a workspace in which a sub-module can be replaced with
a different version of the same module and developed/tested within the
surrounding framework of other modules that define a released
hierarchy.

"swap replace" replaces the version of a module in the workspace with
a different version of the same module. The replace operation updates
the selector and href mode, and calls populate recursively to replace
one version of a workspace module with another version of the same
module. The populate operation uses all persistent populate controls
(such as filters).

"swap show" shows the currently swapped module versions in the
workspace. This information is useful when an end user needs to know
what modules have been updated for development and test.

"swap restore" restores a previously swapped module to the version
defined by a parent module in the workspace. The restore operation
calls populate recursively using all persistent populate controls
(such as filters).
```

populate of a swapped sub-module

```
The "swap replace" and "swap restore" commands always perform a full
recursive populate, applying the full mode to the entire hierarchy of
the swapped module. The populate operation uses the selector and
hrefmode specified to the "swap replace" command, rather than using
the selector and hrefmode determined by the parent. The edit-in-place
methodology changes the selector and href mode as necessary to ensure
that the desired sub-module versions are replaced in the workspace.
Persistent settings (such as filters) associated with the original
module version will be applied to the new swapped module version.

If all of these conditions are met:
- the module being swapped is an mcache link
- the default mcache mode is to link to modules in the module cache
- the specified selector is static
- the modified selector resolves to a module version found in a
  module cache
Then the populate operation will replace the existing mcache link with
a new link pointing to the new version of the module.

The populate command does not replace fetched module instances with
mcache links. If the selector of a fetched module instance is modified
the populate command will not replace the existing module instance
with an mcache link even if all other conditions for mcache linking
are met. Instead, the populate command will refetch the module
instance using the modified selector.
```

When populate is run in verbose mode, its output indicates when the selector of a swapped sub-module is being used.

ci of a swapped sub-module

The ci command will not checkin a module with a static selector. For a recursive operation, the ci command will continue following the module hierarchy in the workspace looking for modules that can be checked in (e.g. swapped modules with dynamic selectors).

During a recursive checkin, the href from a current parent module version to a swapped module is carried over to the new parent module version without change. I.e., The href to a swapped module is not updated. The purpose of swapping a module is to develop and test it within a module framework, not capture new versions of the parent module that reflect a static hierarchy containing the swapped module versions. Whoever is responsible for integration will capture new versions of the parent module.

The output from ci indicates when modules are not checked in because they have static selectors. The output also indicates when hrefs are not updated because their selectors do not match the actual selector of the sub-module in the workspace.

SYNOPSIS

```
swap <swap_command> [<swap_command_options>]

Usage: swap [replace|restore|show]
```

OPTIONS

Vary by command.

RETURN VALUE

Varies by command.

SEE ALSO

ci, populate, swap replace, swap restore, swap show

swap replace

swap replace Command

NAME

```
swap replace        - Replaces the version of a workspace module
```

DESCRIPTION

- Replacing mcache Links
- Understanding the Output

```
This command replaces the version of a module in the workspace with a
different version of the same module.

The replace operation:

o Updates the selector
  The persistent selector of a module instance is changed to the
  selector specified on the command line.

  Note: This overrides any previous swap or overriding hierarchical
  reference.

o Updates the href mode
  The persistent href mode is set to the value specified by the user
  on the command line. If no value was specified by the user, the
  persistent href mode is changed to normal if the persistent href
  mode is static, and the specified selector is dynamic. In all other
  cases the persistent href mode will not be modified.

  Note: When using href mode "normal," the href mode behavior
  respects 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.

o Runs the populate command on the module instance
  The populate replaces one version of workspace module with another
  version of the same module.

The swap replace command always populates recursively. Replacing the
entire hierarchy of a module that is being swapped ensures that all
portions of the new module version are present in the workspace.

Persistent filters set on the original module version are applied to
the swapped module version. This is so the same module content is
populated when replacing one module version with another.

The default fetch state is always used, to fetch the replacement
module in the same state that the rest of the workspace likely is in.
The user can re-populate any portion of the replaced module hierarchy
as necessary to change the fetch state for individual objects.
```

The swap replace command fails if the module hierarchy being replaced contains any local modifications (locally modified, added, moved, or removed objects), unless the -force option is used. The command also cancels locks on locally locked, unmodified objects, failing if it could not successfully remove all locks.

Running the swap replace command on a module that has already been replaced will replace the module again using the specified selector and hrefmode.

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

This command supports the command defaults system.

Replacing mcache Links

DesignSync can update an mcache link in the hierarchy with mcache links or replace the module in the hierarchy with mcache links.

You can replace a module with mcache links when the following conditions are met:

- The module is being fetched statically.
- The module being fetched is available in an mcache; meaning it is contained in a known mcache directory and is linkable by the client.
- The base directory of the existing module is not shared with a different, overlapping module.
- The base directory of the existing module version hierarchy could be successfully removed with the rmmod command.

Important: When replacing a module that is not already populated with mcache links, you must specify the -force option.

Understanding the Output

The swap replace command provides the option to specify the level of information the command outputs during processing. The report mode is passed along to the underlying populate command. Most of the output detail below is from the populate command documentation, and is included here for easy reference. The only output specific to the swap replace command pertain to workspace checks prior to calling the populate command.

The -report option allows you to specify what type of information is displayed:

If you run the command with the -report brief option, the swap replace command outputs the following information:

```
  o Failure messages
  o Warning messages
  o Version of each module processed as a result of a recursive
     populate
  o Creation message for any new hierarchical reference populated as a
    result of a recursive module 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 specify the default -report normal
option, the swap replace 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 Information about all objects that are fetched

If you run the command with the -report verbose option, the swap
replace command outputs all the information presented with -report
normal and the following additional information:
  o Status messages for workspace checks prior to calling the populate
    command
  o Informational message for every object examined but not updated
  o Information about all objects that are filtered

If you run the command with the -report error option, the swap replace
command outputs the following information:
  o Failure messages
  o Warning messages
  o Success/failure/skip status messages
```

SYNOPSIS

```
swap replace [-force] [-hrefmode {static|normal|dynamic}]
             [-mcachemode {link|server}] [-mcachepaths <path>[<path>...]]
             [-trigarg <arg>] [-report {error|brief|normal|verbose}]
             [-xtras <>] [--] <selector> <argument>
```

SELECTOR

- Selector

Selector

```
<selector>   A <selector> is required, to identify the module version
             to fetch into the workspace. This value, which can be a
             selector list, will be preserved as the persistent
             selector of the replaced module for use in future
```

```
populate operations. For more information on selectors
and selector lists, see the selectors topic.
```

ARGUMENTS

- Module Instance

Module Instance

```
<modinst>    A <modinst> is required, to identify the workspace module
             instance to replace. Note that the instance name remains
             the same for the new module version.

             The edit-in-place work flow is intended to operate on
             sub-modules. The swap restore command cannot "unreplace"
             a top-level module because there is no parent from which
             to restore. Consequently, top-level module instance
             arguments will be rejected and the swap replace command
             will throw an error if a top-level module instance is
             specified.
```

OPTIONS

- -force
- -hrefmode
- -mcachemode
- -mcachepaths
- -report
- -trigarg
- -xtras
- --

-force

```
-force              Specifies whether to force a replace that
                    removes the old module hierarchy from the
                    workspace if a new one can be created.

                    -noforce does not remove the old module
                   hierarchy if there are modified files in the
                   workspace or if the users is replacing file
                   copies in the workspace with mcache
                   links. (Default)

                    -force calls rmmod to remove the previous module
                  so it can be replaced with the specified
                  version.  This option is required if the user
                  wants to replace local copies of module files
                  with an mcache link, or if local changes in the
```

workspace are being overwritten with the restored module.

-hrefmode

-hrefmode

Specifies the href mode to use when populating the new module version.

Valid values are:
- dynamic - Expands the href at the time of the populate operation to identify the version of the submodule to be populated.
- static - Populates the submodule version referenced by the href when the module version was initially created.
- normal - Expands the hrefs at the time of the 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.

If no value is specified, the persistent href mode does not change unless the persistent href mode is static and the specified selector is dynamic, in which case the persistent href mode is set to dynamic to match the selector.

The default value is consistent with what the user of an edit-in-place methodology wants to accomplish. If a non-static selector is specified the user is most likely swapping in a new version for development. If a static version is specified the user is probably replacing another static version with the intent of testing the second version within a module framework. They therefore want the new version fetched in the same manner as the original module version.

-mcachemode

-mcachemode link|server

Specifies how the populate fetches the module from the module cache.
- 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. (Default)

                    Note: You cannot create mcache links to
                    dynamically fetched modules.

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

                  Note: The -mcachemode option overrides the
                  default module cache mode registry setting. If
                  -mcachemode is not specified, the swap replace
                  command uses the mode specified in the registry
                  setting. If no registry setting is specified,
                  the command uses the default value.
```

-mcachepaths

```
 -mcachepaths          Identifies one or more module caches to be
  <path[ path...]>     searched for replacing swapped modules from the
                     module cache.

                      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 command
                      fetches modules from the server.
```

-report

```
 -report brief|       Specifies the amount and type of information
  normal|verbose|     displayed by the swap replace command, and by its
  error               call to populate. 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. Workspace checks prior to calling populate are also output.

-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.

-xtras

-xtras <string> 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 swap replace or rmmod commands, 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 the object you specify begins with a hyphen (-).

RETURN VALUE

The swap replace command returns the populate result, which is

```
described below. However, if any step of the swap replace operation
fails prior to the populate call, then an error is thrown with an
appropriate message.

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.
```

SEE ALSO

```
command defaults, ci, populate, selectors, swap, swap restore,
swap show
```

EXAMPLES

```
This example replaces the current sub-module version of Alu with the
Latest version on the Trunk branch, in verbose mode.

stcl> swap replace Trunk:Latest Alu%0 -report verbose

/home/tbarbg7/top/design/Alu/Alu%0: Checking for modified objects in module
hierarchy ...

/home/tbarbg7/top/design/Alu/Alu%0: Checking for locked objects in module
hierarchy ...

Beginning populate operation at Wed Aug 03 02:28:55 PM EDT 2011...

Populating objects in Module          Alu%0
                      Base Directory  /home/tbarbg7/top/design/Alu
                      With href recursion

Fetching contents from selector 'Trunk:Latest', module version '1.3'

Alu%0/mult8.v : Already Fetched and Unmodified Version 1.1
Alu%0/mult8.gv : Already Fetched and Unmodified Version 1.1
```

```
 Alu%0/alu.gv : Already Fetched and Unmodified Version 1.1
 Total data to transfer: 6 Kbytes (estimate), 1 file(s), 0 collection(s)
 Progress - from local cache: 0 Kbytes, 0 file(s), 0 collection(s)
 Progress - from server: 6 Kbytes, 1 file(s), 0 collection(s), 100.0%
complete

 Alu%0/alu.v : Success - Checked out version: 1.2

 Alu%0 : Version of module in workspace updated to 1.3

 Finished populate of Module Alu%0 with base directory
/home/tbarbg7/top/design/Alu

 Time spent: 0.1 seconds, transferred 6 Kbytes, copied from local cache 0
Kbytes, average data rate 42.0 Kb/sec

 Finished populate operation.

 {Objects succeeded (4)} {}
 stcl>
```

swap restore

swap restore Command

NAME

```
 swap restore        - Restores a previously swapped module version
```

DESCRIPTION

- Restoring mcache Links
- Understanding the Output

```
 This command restores a previously swapped sub-module version to the
 version identified by a parent module in the workspace. The restore
 operation calls populate recursively using all persistent populate
 controls (such as filters).

 The sub-module version to restore is determined as follows:
 - If the -parent option is specified the sub-module selector to use is
   obtained from the hierarchical reference of the identified parent.
 - Otherwise, if the sub-module being restored has only one parent the
   selector to use is obtained from the hierarchical reference of the
   parent to the sub-module.
 - Otherwise, if the sub-module has multiple parents and the
   hierarchical reference of every parent has the same selector value,
   then the selector is used to identify the sub-module version to
   restore.
 - Otherwise, if there is an overriding href defined for the parent
   module, the overriding href is used to determine the appropriate
   submodule.
```

```
- If the sub-module is a mcache link or the replace operation
  requests mcache links, the sub-module version replaces the mcache
  link or the file copies with the mcache link.

If the sub-module version to restore cannot be determined, an
error is reported.

The href mode used to restore the sub-module is determined via the
parent module exactly as if the sub-module were being fetched
recursively via the parent:
- If the parent href mode is normal or dynamic the sub-module
  selector to use is obtained from the selector field of the
  hierarchical reference to the sub-module.
- If the parent module was fetched in static mode the sub-module
  selector to use is obtained from the version field of the
  hierarchical reference to the sub-module.
- The href mode is propagated to the sub-module following the existing
  populate rules.
This process ensures that the sub-module is restored as if it were
recursively populated through its parent.

Note: When using href mode "normal," the href mode behavior respects
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.

The swap restore command fails if the module hierarchy being restored
contains any local modifications (locally modified, added, moved or
removed objects) and the -force option is not specified. The command
also cancels locks on locally locked, unmodified objects, failing if
it could not successfully remove all locks.

Note: This command does not provide a -dryrun option.  To simulate a
list of the objects that would be replaced, you can run "ls
-unmanaged -recursive <basedir>' and -ls -modified -recursive
<modinstance>.  These command operates in folder recursively and does
not show any changes that are part of the hierarchy, but outside the
module base directory.

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

This command supports the command defaults system.
```

Restoring mcache Links

```
DesignSync can update a mcache link in the hierarchy with mcache links or
replace the module in the hierarchy with mcache links.

You can restore a module with mcache links when the following
conditions are met:
```

- The module is being fetched statically
- The module being fetched is available in an mcache; meaning it is contained in a known mcache directory and is linkable by the client.
- The base directory of the existing module is not shared with a different, overlapping module.
- The base directory of the existing module version hierarchy is successfully removed.

Important: When restoring a module that is not already populated with mcache links with an mcache link, you must specify the -force option.

Understanding the Output

The swap restore command provides the option to specify the level of information the command outputs during processing. The report mode is passed along to the underlying populate command. Most of the output detail below is from the populate command documentation, and is included here for easy reference. The only output specific to the swap restore command pertain to workspace checks prior to calling the populate command.

The -report option allows you to specify what type of information is displayed:

If you run the command with the -report brief option, the swap restore command outputs the following information:

- Failure messages
- Warning messages
- Version of each module processed as a result of a recursive populate
- Creation message for any new hierarchical reference populated as a result of a recursive module populate
- 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 specify the default -report normal option, the swap restore 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)
- Information about all objects that are fetched

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

- Status messages for workspace checks prior to calling the populate command
- Informational message for every object examined but not updated
- Information about all objects that are filtered

```
If you run the command with the -report error option, the swap restore
command outputs the following information:
 o Failure messages
 o Warning messages
 o Success/failure/skip status messages
```

SYNOPSIS

```
swap restore [-[no]force] [-mcachemode {link|server}]
             [-mcachepaths <path>[ <path>...]]
             [-report {error|brief|normal|verbose}]
             [-parent <parent_module_inst>] [-trigarg <arg>]
             [-xtras <string>] [--] <argument>
```

ARGUMENTS

- Workspace Module

Workspace Module

```
<workspace    A workspace module instance is required, to identify
 module>      the workspace module instance to restore.
```

OPTIONS

- -force
- -mcachemode
- -mcachepaths
- -parent
- -report
- -trigarg
- -xtras
- --

-force

```
-[no]force             Specifies whether to force a restore that
                       removes the old module hierarchy from the
                       workspace if a new one can be created.

                       -noforce does not remove the old module
                     hierarchy if there are modified files in the
                     workspace or if the users is replacing file
                     copies in the workspace with mcache
                     links. (Default)
```

-force calls rmmod to remove the existing module in the workspace and replace it with the module version indicated by persistent populate settings on the parent module.

This option is required if the user wants to replace local copies of module files with an mcache link, or if local changes in the workspace are being overwritten with the restored module.

-mcachemode

-mcachemode link| server	Specifies how the populate called by the swap restore fetches the module from the module cache. 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. (Default) Note: You cannot create mcache links to dynamically fetched modules. o server - Causes the populate command to fetch modules as physical copies from the server, not the module cache. Note: The -mcachemode option overrides the default module cache mode registry setting. If -mcachemode is not specified, the swap restore command uses the mode specified in the registry setting. If no registry setting is specified, the command uses the default value.

-mcachepaths

-mcachepaths <path[path...]>	Identifies one or more module caches to be searched for restoring swapped modules from the module cache. 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 command
                        fetches modules from the server.
```

-parent

```
 -parent                  Specifies the instance name of the parent
 <parent_module_inst>     module from which to determine the version of
                          the module to restore. This option is only
                          required if all parents do not have the same
                          selector for <modinst> among their hierarchical
                          references.
```

-report

```
 -report error|       Specifies the amount and type of information
  brief|normal|       displayed by the swap restore command, and by its
  verbose             call to populate. 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. Workspace checks prior to calling
                      populate are also output.
```

-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.

-xtras

-xtras <string> 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 swap restore or rmmod commands, 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 the object you specify begins with a hyphen (-).

RETURN VALUE

The swap restore command returns the populate result, which is described below. However, if any step of the swap restore operation fails prior to the populate call, then an error is thrown with an appropriate message.

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.

SEE ALSO

command defaults, ci, populate, swap, swap replace, swap show

EXAMPLES

This example restores the Gold version of the Alu sub-module, in verbose mode. Alu has only one parent, so the selector to use is obtained from the hierarchical reference of the parent to the Alu sub-module.

```
stcl> swap restore -report verbose Alu%0

 /home/tbarbg7/sitar/top/Alu/Alu%0: Checking for modified objects in module
hierarchy ...

 /home/tbarbg7/sitar/top/Alu/Alu%0: Checking for locked objects in module
hierarchy ...

 Beginning populate operation at Wed Aug 03 02:39:55 PM EDT 2011...

 Populating objects in Module          Alu%0
                       Base Directory  /home/tbarbg7/top/design/Alu
                       With href recursion

 Fetching contents from selector 'Gold', module version '1.2'

 Alu%0/mult8.v : Already Fetched and Unmodified Version 1.1
 Alu%0/mult8.gv : Already Fetched and Unmodified Version 1.1
 Alu%0/alu.gv : Already Fetched and Unmodified Version 1.1
 Total data to transfer: 6 Kbytes (estimate), 1 file(s), 0 collection(s)
 Progress - from local cache: 0 Kbytes, 0 file(s), 0 collection(s)
 Progress - from server: 6 Kbytes, 1 file(s), 0 collection(s), 100.0%
complete

 Alu%0/alu.v : Success - Checked out version: 1.1

 Alu%0 : Version of module in workspace updated to 1.2

 Finished populate of Module Alu%0 with base directory
/home/tbarbg7/top/design/Alu

 Time spent: 0.1 seconds, transferred 6 Kbytes, copied from local cache 0
Kbytes, average data rate 40.3 Kb/sec

 Finished populate operation.

 {Objects succeeded (4)} {}
 stcl>
```

swap show

swap show Command

NAME

```
swap show           - Shows the workspace's currently swapped modules
```

DESCRIPTION

- Understanding the Output

```
This command shows all of the modules in a workspace that have had
their versions swapped. This information is useful to an end user who
needs to know what modules have been updated for development and test.

This command supports the command defaults system.
```

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.

By default, or if you run the swap show command with the '-report
normal' option, the following information is output:

Column       Property
Titles       Names          Description
------       -----          ------------
Name         name           Name of the module.
                            Note: Modules are displayed
                            alphabetically by name.

Instance     modinstname    Unique instance name of the module in
                            the workspace.

Base                        Workspace directory containing the
Directory    basedir        contents of the module.

Url          url            Location of the module on the server.

Selector     selector       Selector used to determine which version
                            to fetch into a workspace.  For more
                            information, see the selectors help
                            topic.

"Beginning" and "Finished" progress messages are also output in
normal mode.

If you run the swap show command with the '-report brief' option, it
displays the following information:
```

```
Column       Property
Titles       Names          Description
------       -----          ------------
Unique                      Full module address for the module version
Address      address        in the workspace.

If you run the swap show command with the '-report verbose' option, it
displays the information shown with '-report normal' option and the
following additional information:

Column       Property
Titles       Names          Description
------       -----          ------------
Unique                      Full module address for the module version
Address      address        in the workspace.

Version      version        The version of the module in the
                            workspace.
```

SYNOPSIS

```
swap show [-format {text|list}] [-report {brief|normal|verbose}] [--]
          [<argument>]
```

ARGUMENTS

- Workspace Folder

Workspace Folder

```
<workspace folder>     Specifies the workspace folder for which you
                       want to view the swapped module versions.
```

OPTIONS

- -format
- -report
- --

-format

```
-format text|list      Determines the format of the output. For
                       information about the information displayed,
                       and the output sort order, see the
                       "Understanding the Output" section.

                       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.

-report

-report brief|normal|verbose	Determines what the output of the command is. For more information on the output of the -report option, see the "Understanding the Output" section.

Valid values are:

- o brief - Displays the full workspace path for each workspace module reported.
- o normal - Displays basic module information for each workspace module reported. Also outputs progress messages for the command. (Default)
- o verbose - Displays extended module information for each workspace module reported.

--

--	Indicates that the command should stop looking for command options. Use this option when the object you specify begins with a hyphen (-).

RETURN VALUE

If you run the swap show command with the '-format list' option, it

```
 returns a Tcl list. If you run the swap show command with the '-format
 text' option, it returns an empty string. For a description of the
 output, see the "Understanding the Output" section.

 If the command fails, it throws a Tcl error.
```

SEE ALSO

```
 command defaults, ci, populate, showmods, showstatus, swap,
 swap replace, swap restore
```

EXAMPLES

```
 This example shows the modules in the current workspace that have been
 swapped. The default "-report normal" mode is used.

 stcl> swap show

 Beginning swap show operation ...

 Name       Instance     Base Directory                          Url
Selector
 ------------------------------------------------------------------------------
----------------------------------------
 Addr_calc  Addr_calc%0  /home/tbarbg7/top/design/Addr_calc
sync://qelwsun14:30148/Modules/Addr_calc  Trunk:Latest
 Decoder    Decoder%0    /home/tbarbg7/top/design/Decoder
sync://qelwsun14:30148/Modules/Decoder    Trunk:Latest
 Instr_reg  Instr_reg%0  /home/tbarbg7/top/design/Instr_reg
sync://qelwsun14:30148/Modules/Instr_reg  Trunk:Latest

 Finished swap show operation.
 stcl>
```

Whereused Information

whereused

whereused Command

NAME

```
 whereused           - Traces the use of hierarchical references
```

DESCRIPTION

```
 This command is being deprecated in favor of 'whereused module' and
```

'whereused vault'. In future releases, this command will behave as other superset commands do, returning a list of of the available subcommands. A third command, "whereused member," has been added to identify where a module member is used.

Tip: Update any scripts, processes, or proceedures that use or recommend the whereused command to the appropriate whereused subcommand.

This command identifies the module versions in which any of the following targets (the "toargument" for the addhref command) are used: sub-module version, legacy module configuration, or DesignSync vault, identified by a selector. This allows users to trace the usage of any desired target.

When a hierarchical reference is created, DesignSync creates a back reference on the toargument indicating that it is referenced by the fromargument module. This provides the basic mechanism for the whereused functionality. If a back reference does not exist, for example, if the hierarchical reference was created before whereused was implemented, you can add back references independently using the addbackref command. The whereused command performs no validation on specified legacy module and vault selectors. Verify that the selector specified is correct before executing the command.

Notes:

- When referencing legacy modules or DesignSync vaults using a configuration name or a selector as the specified version, the version must be an exact match for the selector used when the hierarchical reference was added.
- When the hierarchical reference target (toargument) was specified with SSL protocol, the whereused command must also be specified with SSL.

This command is subject to access controls on the server. The whereused command requires browse access to the objects in the module hierarchy. See the ENOVIA Synchronicity Access Control Guide for details.

SYNOPSIS

```
whereused [-format list|text] [-[no]recursive] [-report <mode>]
          [-showtags all|none|immutable|version]
          [-versions <selector>,...] <argument>
```

ARGUMENTS

- DesignSync Vault
- Server Module
- Legacy Module URL

DesignSync Vault

```
<DesignSync vault> URL of the referenced DesignSync vault.
                   Specify the URL as follows:
                   sync[s]://<host>[:<port>]<vaultpath>;[<selector>]
                   where <host> is the SyncServer on which the
                   object resides, <port> is the SyncServer
                   port number, <vaultpath> identifies the
                   DesignSync object, and <selector> is the optional
                   selector.

                   If a selector is not specified, the command
                   uses versions specified with the -versions option
                   or the default value of Trunk:Latest.

                   Notes:
                     o If you specify a selector in the argument, you
                       cannot specify additional selectors with the
                       -versions option. To locate more than one
                       selector at a time, use the -versions option.

                     o When the hierarchical reference target
                       (toargument) was specified with SSL protocol,
                       the whereused command must also be specified
                       with SSL.

                     o The vault selector is not checked for validity
                       before the command is executed. Verify that
                       the vault URL and selector is typed correctly
                       before executing the command.
```

Server Module

```
<server module>    URL of the referenced module.
                   Specify the URL as follows:
                   sync[s]://<host>[:<port>]/Modules/[<Category>...]
                    /<module>[;<selector>,...]
                   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 <selector> is the optional
                   selector.

                   If a selector is not specified, the command
                   uses versions specified with the -versions option
                   or the default value of Trunk:Latest.

                   Notes:
                     o If you specify a selector in the argument, you
                       cannot specify additional selectors with the
                       -versions option. To locate more than one
                       selector at a time, use the -versions option.
```

```
                    o When the hierarchical reference target
                      (toargument) was specified with SSL protocol,
                      the whereused command must also be specified
                      with SSL.
```

Legacy Module URL

```
<Legacy module      URL of the referenced legacy module.
 URL>               Specify the URL as follows:
                    sync[s]://<host>[:<port>]/[Projects]/<vaultpath>;
                     [<selector>,...]
                    where <host> is the SyncServer on which the
                    module resides, <port> is the SyncServer
                    port number, vaultpath is the path to the
                    module, and <selector> is the optional selector,
                    or configuration name.

                    If a selector is not specified, the command
                    uses versions specified with the -versions option
                    or the default value of Trunk:Latest.

                    Notes:
                      o If you specify a selector in the argument, you
                        cannot specify additional selectors with the
                        -versions option. To locate more than one
                        selector at a time, use the -versions option.

                      o When the hierarchical reference target
                        (toargument) was specified with SSL protocol,
                        the whereused command must also be specified
                        with SSL.

                      o The vault selector is not checked for validity
                        before the command is executed. Verify that
                        the vault URL and selector is typed correctly
                        before executing the command.
```

OPTIONS

- -format
- -[no]recursive
- -report
- -showtags
- -versions

-format

```
-format list|text   Determines the format of the output.  For
                    details about the information returned
```

```
see the -report option.
Valid values are:

o text - Display text output formatted to show the
hierarchical reference tree. (Default)

o list - Displays a list with the following
         format:
          {
            object <object>...whereused
            <hierarchical_whereused_list>
          }
```

-[no]recursive

```
-[no]recursive    Determines whether to show the locations in
                  which the version is explicitly referenced, or
                 show all modules in which the version is implicitly
                 or explicitly referenced. An explicit reference
                 exists when there is a direct reference link
                 between the module and the
                 target.  An implicit reference exists when the
                 module and target are not directly connected, but
                 within the module's hierarchy exists a reference
                 to the target. For example: if the Chip module
                 references the Gold version of the ALU module, and
                 the Gold version of the ALU module references the
                 Gold version of the ROM module, the Chip module
                 contains an implicit reference to the ROM module
                 and an explicit reference to the ALU module.

                  -norecursive expands the reference and returns a
                  list of modules that contain an explicit reference
                  to the referenced module.  (Default)

                  -recursive traverses the reference and returns a
                  list of all modules that explicitly and implicitly
                  reference the specified version.
```

-report

```
-report brief|    Determines what the information is returned in the
 normal|verbose   output of the whereused command.

                  Valid values are:

                  o brief - reports only the module names for the
                    specified object ordered by specified version.

                    When displayed in text format, the output is
                    indented to show the reverse hierarchical
```

reference path. When displayed in list format, the hierarchical reference depth is indicated with the depth property.

Note: The -showtags option is ignored for this report mode.

- normal - reports the version of the module containing the reference to the specified object. The text format is indented to show hierarchical reference depth, and the list format uses the depth property. The version property in list mode contains the version number.

  Note: The version property may be a selector rather than a version number if the target is a DesignSync vault or legacy module.

- verbose - provides the information available in report mode -normal as well as the processing status of the command.

-showtags

-showtags all|none|immutable|version	Specifies whether tag information is displayed and optionally restricts the output to immutable tags or tagged versions.

- all - Displays all tags and all reference locations, including references that are not tagged. (Default)
- none - Displays all references locations, but does not display tag information
- immutable - Displays only reference locations tagged with an immutable tag and the name of the immutable tag.

  Note: Using the -showtags immutable option may not display all versions in which an immutable tag is used. The whereused command queries for all the whereused information but filters the display from the starting point until it reaches the last immutable tag in a reference tree.

- version - Displays any reference location that has a version tag and the name of the tag.

Note: The -showtags option is ignored for -report brief mode.

-versions

```
-versions          Comma separated set of versions to locate. The
 <selector>        versions can be numerics or selectors, If the
                   argument specified is a module, the selector
                   resolves to the numeric value of the module
                   version and is compared against the static href
                   value.

                   If the object is a DesignSync vault or legacy
                   module, then the version selector (branch, version
                   release, or alias selector) is validated by string
                   comparison against the dynamic href value.

                   Note: To specify a comma separated selector list,
                   use the version extended naming format, not the
                   -versions option.
```

RETURN VALUE

```
This command does not return a Tcl value in text mode.  If the command
is unable to run,  DesignSync throws an error explaining the failure.

If whereused is unable to process a reference, you will see the
reason for failure during command processing and the command adds
the sync URL of the failed module to the wuFail list.

In list mode, the list of modules is returned in an array of
name/value pairs. If a reference cannot be processed, the command
adds an error property containing the reason for the failure and the
module is added to the wuFail list.
```

SEE ALSO

```
selectors, addhref, addbackref, edithrefs, rmhref, showhrefs
```

EXAMPLES

- Example of Using whereused to find direct references to a version
- Example of Using whereused to find all references to version
- Example of Using whereused to find immutable tagged versions
- Example of Using whereused to find tagged versions
- Example of Displaying whereused Output in Tcl list

```
Many products use the common code available in LIB module.  To assure
the all the development teams are using a tested and approved version
of the LIB module, the authorized release versions are tagged with
```

```
immutable release tags after they are qualified. The version of LIB
in our example is v1.2.  A software defect is discovered in the LIB
module. Product management wants to quickly locate all the products
built with references to that version so that the new version can be
shipped to all customers who have that version.
```

Example of Using whereused to find direct references to a version

```
This whereused example shows all the modules that refer
explicitly to the defective LIB version.
Note: The (*) after a tag indicates that the tag is immutable.

stcl> whereused -version v1.2 -showtags all
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running non-recursively...

 ============================================================

 sync://svr2.ABCo.com:2647/Modules/LIB;1.32 - v1.2(*), Trunk:
      sync://svr2.ABCo.com:2647/Modules/Chip;1.7 - Trunk

 ============================================================


 Finished whereused operation.
```

Example of Using whereused to find all references to version

```
This example shows all the modules that refer to the defective LIB
module explicitly and implicitly.
Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused -recursive sync://svr2.ABCo.com:2647/Modules/LIB;1.32

 Beginning whereused operation ...

 Running recursively...

 ======================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB;1.32 - v1.2(*), Trunk:
      sync://svr2.ABCo.com:2647/Modules/Chip;1.7 - Trunk
         sync://svr2.ABCo.com:2647/Modules/top;1.9 - Trunk:
 ======================================================================


 Finished whereused operation.
```

Example of Using whereused to find immutable tagged versions

```
In the scenario above, the LIB module is tagged with an immutable
tag.  This usually indicates a significant release such as a released
product version, or, in our case, a released library reused
throughout the code base.  You can use the whereused command to
search for all objects tagged with an immutable tag.

Note: This may not display all the versions on which an immutable
tag is used.  Whereused queries for the information but
displays from the starting point until it reaches the last immutable
tag in a reference tree.

For instance, if you have a hierarchy such this:
top - not tagged with an immutable tag
 -> Chip tagged with an immutable tag
     -> LIB, tagged with an immutable tag,

the output of whereused -showtags immutable includes Chip and LIB.
If instead of Chip, top is tagged with an immutable tag, the output
shows top, Chip, and LIB, even though Chip is not tagged with an
immutable tag.

To perform a query to return all the whereused information for a
specific immutable tag, you can write a filtering script around
whereused using -showtags all and filtering for the desired immutable
tag.

In this example, the top module and LIB module both contain immutable
tags.
Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused -version v1.2 -showtags immutable -rec
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running recursively...

 ============================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB - v1.2(*)
      sync://svr2.ABCo.com:2647/Modules/Chip;1.8
         sync://svr2.ABCo.com:2647/Modules/top;1.10 - v1.2(*)
 ============================================================================


 Finished whereused operation.
```

Example of Using whereused to find tagged versions

```
This example shows all the tagged versions of modules that refer to
```

```
the defective LIB module.
 Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused -version v1.2 -showtags version -rec
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running recursively...

 ========================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB - v1.2(*)
      sync://svr2.ABCo.com:2647/Modules/Chip;1.8 - Gold, Latest
         sync://svr2.ABCo.com:2647/Modules/top;1.10 - Latest, v1.2(*)
 ========================================================================
```

Example of Displaying whereused Output in Tcl list

```
This example shows all the modules that refer to the defective ROM
module explicitly or implicitly in Tcl list format.

stcl> whereused -recursive -version v1.2 -format list
sync://svr2.ABCo.com:2647/Modules/ROM

{{object sync://svr2.ABCo.com:2647/Modules/LIB version 1.2 depth 0
{v1.2(*), Trunk:} whereused {{object
sync://svr2.ABCo.com:2647/Modules/Chip/ version 1.8  depth
1 {Gold, Trunk:} whereused {{object
sync://svr2.ABCo.com:2647/Modules/top version 1.10 depth 2  {v1.2(*),
Trunk:} whereused {}}}}}}}
```

whereused member

whereused member Command

NAME

```
whereused member    - Lists module versions containing a module member
```

DESCRIPTION

```
This command identifies where a module version is used.
This information includes:
 o Module version(s) in which the module member version exists.
 o Tags, if any, for the module version in which the module member
    version exists.
 o Module Member Tags in which the module member version exists,
   (also called "snapshot tags")
```

```
This allows users to trace the usage of any desired target.

This command is subject to access controls on the server. The
whereused command requires browse access to the objects in the
module hierarchy. See the ENOVIA Synchronicity Access Control Guide
for details.
```

SYNOPSIS

```
whereused member [-format list|text] [-modulecontext <moduleVersionURL>]
                 [-report brief|normal|verbose]
                 [-showtags all|none|immutable|version|member]
                 [-versions <selector>,...] <argument>
```

ARGUMENTS

- Module Member

Module Member

```
<module_member>   Specify the module member. It can be specified as a
                  workspace module member with an absolute or
                  relative path, or, when using the -modulecontext
                  option, a natural path.
```

OPTIONS

- -format
- -modulecontext
- -report
- -showtags
- -versions

-format

```
-format list|text  Determines the format of the output.  For
                   details about the information returned
                   see the -report option.
                   Valid values are:

                   o text - Display text output. (Default)

                   o list - Returns a TCL list.  See the Examples
                            section for examples of the TLC list
                            formatting.
```

-modulecontext

-modulecontext <moduleVersionURL>

Identifies a module version for the module member argument. The value must be a module version URL. When the -modulecontext option is used, the argument must be specified as a natural path.

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.

-report

-report brief|normal|verbose

Determines what the information is returned in the output of the whereused command.

Valid values are:

- brief - displays only information for tagged module versions.
- normal - displays information for all versions of the module containing the specified object.
- verbose - provides the information available in report mode normal as well as the processing status of the command and the natural path of the object.

-showtags

-showtags all|none|immutable|version|member

Specifies whether tag information is displayed and optionally restricts the output to immutable tags tagged versions, or module members.

- all - Displays module versions, module version tags and member tags. (Default)
- none - Displays module versions, but does not display tag information.
- immutable - Displays only module versions tagged with an immutable tag, and the name of the tag.
- version - Displays any module versions that have a version tag, and the name of the tag.
- member - Displays only the member tags applied

```
                    to the member version.
```

-versions

```
-versions          Comma separated set of member versions to
 <selector>        locate. The versions can be numerics or all
                   selectors. By default, this option defaults to the
                  version in the workspace. When using
                  -modulecontext, this option defaults to the member
                  version associated with the specified module
                  context.
```

RETURN VALUE

```
This command does not return a Tcl value in text mode.  If the command
is unable to run,  DesignSync throws an error explaining the failure.

In list mode, the list of module versions is returned in an array of
name/value pairs.
```

SEE ALSO

```
selectors, whereused module
```

EXAMPLES

- Example Showing the whereused member Commmand
- Example Showing the whereused member Commmand in List Mode

Example Showing the whereused member Commmand

```
This example shows locating where the mem.cpp module member is used.
The output is text in normal mode.

stcl> whereused member -versions 1.1,1.2 mem.cpp
    mem.cpp;1.1

    Module Version(s)    Branch  Tags
    ----------------------------------------
    1.2                  Trunk   alpha, pass1
    1.3 - 74             Trunk
    1.75                 Trunk   beta, pass2
    1.76 - 124           Trunk
    1.125                Trunk   Bronze
    1.126 - 268          Trunk
    1.4.1.1              Dev
```

```
    1.4.1.2              Dev     hack1
    1.4.1.3 - 66         Dev
    1.4.2.1 - 2          Test

       Member Version Tags
       -----------------------
       Rel_1
       Rel_2

  ====================================

  mem.cpp;1.2

    Module Version(s)  Branch  Tags
       ----------------------------------------
       1.269              Trunk   Checkpoint

  ====================================
```

Example Showing the whereused member Commmand in List Mode

```
This example shows locating where the mem.cpp module member is used.
The output is a list in verbose mode.

stcl> whereused member -format list -report verbose mem.cpp
{name mem.cpp versions {
  {member_version 1.1
     module_versions {
      {version_range {1.2 1.2} branch Trunk version_tags {alpha
pass1} natural_path /mem.cpp}
      {version_range {1.3 1.45} branch Trunk version_tags {}
natural_path /subdir/mem.cpp}
      {version_range {1.46 1.74} branch Trunk version_tags {}
natural_path /mem.cpp}
      {version_range {1.75 1.75} branch Trunk version_tags {beta
pass2} natural_path /mem.cpp}
      {version_range {1.76 1.124} branch Trunk version_tags {}
natural_path /mem.cpp}
      {version_range {1.126 1.268} branch Trunk version_tags {}
natural_path /mem.cpp}
      {version_range {1.4.1.1 1.4.1.1} branch Dev version_tags {}
natural_path /mem.cpp}
      {version_range {1.4.1.2 1.4.1.2}  branch Dev version_tags hack1
natural_path /mem.cpp}
      {version_range {1.4.1.3 1.4.1.66} branch Dev version_tags {}
natural_path /mem.cpp}
      {version_range {1.4.2.1 1.4.2.2} branch Test version_tags {}
natural_path /mem.cpp}
    }
    member_tags {
       {member_tag Rel_1 natural_path /mem.cpp}
       {member_tag Rel_2 natural_path /subdir/mem.cpp}
    }
 }
```

```
 {member_version 1.2
    {module_versions {
      {version_range {1.269 1.269} branch Trunk version_tags
Checkpoint natural_path /mem.cpp}
    }
    member_tags {}
 }
}
```

whereused module

whereused module Command

NAME

```
whereused module    - Traces the use of hierarchical references
```

DESCRIPTION

```
This command identifies the module versions in which a sub-module
version is used, identified by a selector. This allows users to trace
the usage of any desired target.

When a hierarchical reference is created, DesignSync creates a back
reference on the toargument indicating that it is referenced by the
fromargument module. This provides the basic mechanism for the
whereused functionality. If a back reference does not exist, for
example, if the hierarchical reference was created before whereused
was implemented,  you can add back references independently using the
addbackref command.

Note: When the hierarchical reference target (toargument) was
specified with SSL protocol, the whereused module command must also
be specified with SSL.

This command is subject to access controls on the server. The
whereused module command requires browse access to the objects in the
module hierarchy. See the ENOVIA Synchronicity Access Control Guide
for details.
```

SYNOPSIS

```
whereused module [-format list|text] [-[no]recursive] [-report <mode>]
                 [-showtags all|none|immutable|version]
                 [-versions <selector>,...] <argument>
```

ARGUMENTS

- Server Module

Server Module

```
<server module>    URL of the referenced module.
                   Specify the URL as follows:
                   sync[s]://<host>[:<port>]/Modules/[<Category>...]
                    /<module>[;<selector>,...]
                   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 <selector> is the optional
                   selector.

                   If a selector is not specified, the command
                   uses versions specified with the -versions option
                   or the default value of Trunk:Latest.

                   Notes:
                     o If you specify a selector in the argument, you
                       cannot specify additional selectors with the
                       -versions option. To locate more than one
                       selector at a time, use the -versions option.

                     o When the hierarchical reference target
                       (toargument) was specified with SSL protocol,
                       the whereused module command must also be
                       specified with SSL.
```

OPTIONS

- -format
- -[no]recursive
- -report
- -showtags
- -versions

-format

```
-format list|text  Determines the format of the output.  For
                   details about the information returned
                   see the -report option.
                   Valid values are:

                   o text - Display text output formatted to show the
                   hierarchical reference tree. (Default)

                   o list - Displays a list with the following
                            format:
                             {
```

```
    object <object>...whereused
    <hierarchical_whereused_list>
}
```

-[no]recursive

-[no]recursive

Determines whether to show the locations in which the version is explicitly referenced, or show all modules in which the version is implicitly or explicitly referenced. An explicit reference exists when there is a direct reference link between the module and the target. An implicit reference exists when the module and target are not directly connected, but within the module's hierarchy exists a reference to the target. For example: if the Chip module references the Gold version of the ALU module, and the Gold version of the ALU module references the Gold version of the ROM module, the Chip module contains an implicit reference to the ROM module and an explicit reference to the ALU module.

-norecursive expands the reference and returns a list of modules that contain an explicit reference to the referenced module. (Default)

-recursive traverses the reference and returns a list of all modules that explicitly and implicitly reference the specified version.

-report

-report brief|normal|verbose

Determines what the information is returned in the output of the whereused module command.

Valid values are:

- brief - reports only the module names for the specified object ordered by specified version.

  When displayed in text format, the output is indented to show the reverse hierarchical reference path. When displayed in list format, the hierarchical reference depth is indicated with the depth property.

  Note: The -showtags option is ignored for this report mode.

- normal - reports the version of the module containing the reference to the specified

object. The text format is indented to show hierarchical reference depth, and the list format uses the depth property. The version property in list mode contains the version number.

- verbose - provides the information available in report mode -normal as well as the processing status of the command.

-showtags

-showtags all|none|immutable|version	Specifies whether tag information is displayed and optionally restricts the output to immutable tags or tagged versions. - all - Displays all tags and all reference locations, including references that are not tagged. (Default) - none - Displays all references locations, but does not display tag information - immutable - Displays only reference locations tagged with an immutable tag and the name of the immutable tag. Note: Using the -showtags immutable option may not display all versions in which an immutable tag is used. The whereused module command queries for all the whereused information but filters the display from the starting point until it reaches the last immutable tag in a reference tree. - version - Displays any reference location that has a version tag and the name of the tag. Note: The -showtags option is ignored for -report brief mode.

-versions

-versions <selector>	Comma separated set of versions to locate. The versions can be numerics or selectors. The selector resolves to the numeric value of the module version and is compared against the static href value. Note: To specify a comma separated selector list, use the version extended naming format, not the

```
                    -versions option.
```

RETURN VALUE

```
This command does not return a Tcl value in text mode.  If the command
is unable to run,  DesignSync throws an error explaining the failure.

If whereused module is unable to process a reference, you will see
the reason for failure during command processing and the command adds
the sync URL of the failed module to the wuFail list.

In list mode, the list of modules is returned in an array of
name/value pairs. If a reference cannot be processed, the command
adds an error property containing the reason for the failure and the
module is added to the wuFail list.
```

SEE ALSO

```
selectors, addhref, addbackref, edithrefs, rmhref, showhrefs
```

EXAMPLES

- Example of Using whereused module to find direct references to a version
- Example of Using whereused module to find all references to version
- Example of Using whereused module to find immutable tagged versions
- Example of Using whereused module to find tagged versions
- Example of Displaying whereused module Output in Tcl list

```
Many products use the common code available in LIB module.  To assure
the all the development teams are using a tested and approved version
of the LIB module, the authorized release versions are tagged with
immutable release tags after they are qualified. The version of LIB
in our example is v1.2.  A software defect is discovered in the LIB
module. Product management wants to quickly locate all the products
built with references to that version so that the new version can be
shipped to all customers who have that version.
```

Example of Using whereused module to find direct references to a version

```
This whereused module example shows all the modules that refer
explicitly to the defective LIB version.
Note: The (*) after a tag indicates that the tag is immutable.

stcl> whereused module -version v1.2 -showtags all
sync://svr2.ABCo.com:2647/Modules/LIB
```

```
Beginning whereused operation ...

Running non-recursively...

============================================================

sync://svr2.ABCo.com:2647/Modules/LIB;1.32 - v1.2(*), Trunk:
     sync://svr2.ABCo.com:2647/Modules/Chip;1.7 - Trunk

============================================================


Finished whereused operation.
```

Example of Using whereused module to find all references to version

```
This example shows all the modules that refer to the defective LIB
module explicitly and implicitly.
Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused module -recursive
  sync://svr2.ABCo.com:2647/Modules/LIB;1.32


 Beginning whereused operation ...

 Running recursively...

 ====================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB;1.32 - v1.2(*), Trunk:
      sync://svr2.ABCo.com:2647/Modules/Chip;1.7 - Trunk
         sync://svr2.ABCo.com:2647/Modules/top;1.9 - Trunk:
 ====================================================================


 Finished whereused operation.
```

Example of Using whereused module to find immutable tagged versions

```
In the scenario above, the LIB module is tagged with an immutable
tag.  This usually indicates a significant release such as a released
product version, or, in our case, a released library reused
throughout the code base.  You can use the whereused command to
search for all objects tagged with an immutable tag.

Note: This may not display all the versions on which an immutable
tag is used.  Whereused module queries for the information but
displays from the starting point until it reaches the last immutable
tag in a reference tree.
```

```
For instance, if you have a hierarchy such this:
top - not tagged with an immutable tag
 -> Chip tagged with an immutable tag
     -> LIB, tagged with an immutable tag,

the output of whereused module -showtags immutable includes Chip and
LIB. If instead of Chip, top is tagged with an immutable tag, the
output shows top, Chip, and LIB, even though Chip is not tagged with
an immutable tag.

To perform a query to return all the whereused information for a
specific immutable tag, you can write a filtering script around
whereused using -showtags all and filtering for the desired immutable
tag.

In this example, the top module and LIB module both contain immutable
tags.
Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused module -version v1.2 -showtags immutable -rec
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running recursively...

 ========================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB - v1.2(*)
      sync://svr2.ABCo.com:2647/Modules/Chip;1.8
         sync://svr2.ABCo.com:2647/Modules/top;1.10 - v1.2(*)
 ========================================================================


 Finished whereused operation.
```

Example of Using whereused module to find tagged versions

```
This example shows all the tagged versions of modules that refer to
the defective LIB module.
 Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused module -version v1.2 -showtags version -rec
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running recursively...

 ========================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB - v1.2(*)
      sync://svr2.ABCo.com:2647/Modules/Chip;1.8 - Gold, Latest
```

```
        sync://svr2.ABCo.com:2647/Modules/top;1.10 - Latest, v1.2(*)
 ================================================================
```

Example of Displaying whereused module Output in Tcl list

```
This example shows all the modules that refer to the defective ROM
module explicitly or implicitly in Tcl list format.

stcl> whereused module -recursive -version v1.2 -format list
sync://svr2.ABCo.com:2647/Modules/ROM

{{object sync://svr2.ABCo.com:2647/Modules/LIB version 1.2 depth 0
{v1.2(*), Trunk:} whereused {{object
sync://svr2.ABCo.com:2647/Modules/Chip/ version 1.8  depth
1 {Gold, Trunk:} whereused {{object
sync://svr2.ABCo.com:2647/Modules/top version 1.10 depth 2  {v1.2(*),
Trunk:} whereused {}}}}}}}
```

whereused vault

whereused vault Command

NAME

```
whereused vault     - Traces the use of hierarchical references
```

DESCRIPTION

```
This command identifies the module versions in which either of the
following targets (the "toargument" for the addhref command) are
used: legacy module configuration, or DesignSync vault, identified by
a selector. This allows users to trace the usage of any desired
target.

When a hierarchical reference is created, DesignSync creates a back
reference on the toargument indicating that it is referenced by the
fromargument module. This provides the basic mechanism for the
whereused functionality. If a back reference does not exist, for
example, if the hierarchical reference was created before whereused
was implemented,  you can add back references independently using the
addbackref command. The whereused vault command performs no
validation on specified legacy module and vault selectors. Verify
that the selector specified is correct before executing the command.

IMPORTANT: When referencing either legacy modules or DesignSync
vaults using a configuration name or a selector as the specified
version, the version must be an exact match for the selector used
when the hierarchical reference was added.
```

```
Note: When the hierarchical reference target (toargument) was
specified with SSL protocol, the whereused command must also be
specified with SSL.

This command is subject to access controls on the server. The
whereused vault command requires browse access to the objects in the
module hierarchy. See the ENOVIA Synchronicity Access Control Guide
for details.
```

SYNOPSIS

```
whereused vault [-format list|text] [-[no]recursive] [-report <mode>]
                [-showtags all|none|immutable|version]
                [-versions <selector>,...] <argument>
```

ARGUMENTS

- DesignSync Vault
- Legacy Module URL

DesignSync Vault

```
<DesignSync vault> URL of the referenced DesignSync vault.
                   Specify the URL as follows:
                   sync[s]://<host>[:<port>]<vaultpath>;[<selector>]
                    where <host> is the SyncServer on which the
                   object resides, <port> is the SyncServer
                   port number, <vaultpath> identifies the
                   DesignSync object, and <selector> is the optional
                   selector.

                   If a selector is not specified, the command
                   uses versions specified with the -versions option
                   or the default value of Trunk:Latest.

                   Notes:
                     o If you specify a selector in the argument, you
                       cannot specify additional selectors with the
                       -versions option. To locate more than one
                       selector at a time, use the -versions option.

                     o When the hierarchical reference target
                       (toargument) was specified with SSL protocol,
                       the whereused vault command must also be
                       specified with SSL.

                     o The vault selector is not checked for validity
                       before the command is executed. Verify that
                       the vault URL and selector is typed correctly
                       before executing the command.
```

Legacy Module URL

```
<Legacy module      URL of the referenced legacy module.
 URL>               Specify the URL as follows:
                    sync[s]://<host>[:<port>]/[Projects]/<vaultpath>;
                     [<selector>,...]
                    where <host> is the SyncServer on which the
                    module resides, <port> is the SyncServer
                    port number, vaultpath is the path to the
                    module, and <selector> is the optional selector,
                    or configuration name.

                    If a selector is not specified, the command
                    uses versions specified with the -versions option
                    or the default value of Trunk:Latest.

                    Notes:
                      o If you specify a selector in the argument, you
                        cannot specify additional selectors with the
                        -versions option. To locate more than one
                        selector at a time, use the -versions option.

                      o When the hierarchical reference target
                        (toargument) was specified with SSL protocol,
                        the whereused vault command must also be
                        specified with SSL.

                      o The vault selector is not checked for validity
                        before the command is executed. Verify that
                        the vault URL and selector is typed correctly
                        before executing the command.
```

OPTIONS

- -format
- -[no]recursive
- -report
- -showtags
- -versions

-format

```
-format list|text   Determines the format of the output.  For
                    details about the information returned
                    see the -report option.
                    Valid values are:

                    o text - Display text output formatted to show the
                    hierarchical reference tree. (Default)

                    o list - Displays a list with the following
```

```
format:
 {
   object <object>...whereused
   <hierarchical_whereused_list>
 }
```

-[no]recursive

-[no]recursive

Determines whether to show the locations in which the version is explicitly referenced, or show all modules in which the version is implicitly or explicitly referenced. An explicit reference exists when there is a direct reference link between the module and the target. An implicit reference exists when the module and target are not directly connected, but within the module's hierarchy exists a reference to the target. For example: if the Chip module references the Gold version of the ALU module, and the Gold version of the ALU module references the Gold version of the ROM module, the Chip module contains an implicit reference to the ROM module and an explicit reference to the ALU module.

-norecursive expands the reference and returns a list of modules that contain an explicit reference to the referenced module. (Default)

-recursive traverses the reference and returns a list of all modules that explicitly and implicitly reference the specified version.

-report

-report brief|normal|verbose

Determines what the information is returned in the output of the whereused vault command.

Valid values are:

o brief - reports only the module names for the specified object ordered by specified version.

  When displayed in text format, the output is indented to show the reverse hierarchical reference path. When displayed in list format, the hierarchical reference depth is indicated with the depth property.

  Note: The -showtags option is ignored for this report mode.

o normal - reports the version of the module containing the reference to the specified object. The text format is indented to show hierarchical reference depth, and the list format uses the depth property. The version property in list mode contains the version number.

Note: The version property may be a selector rather than a version number if the target is a DesignSync vault or legacy module.

o verbose - provides the information available in report mode -normal as well as the processing status of the command.

-showtags

-showtags all|none|immutable|version	Specifies whether tag information is displayed and optionally restricts the output to immutable tags or tagged versions.

o all - Displays all tags and all reference locations, including references that are not tagged. (Default)

o none - Displays all references locations, but does not display tag information

o immutable - Displays only reference locations tagged with an immutable tag and the name of the immutable tag.

Note: Using the -showtags immutable option may not display all versions in which an immutable tag is used. The whereused vault command queries for all the whereused information but filters the display from the starting point until it reaches the last immutable tag in a reference tree.

o version - Displays any reference location that has a version tag and the name of the tag.

Note: The -showtags option is ignored for -report brief mode.

-versions

-versions <selector>	Comma separated set of versions to locate. The versions can be numerics or selectors. The version

```
 selector (branch, version release, or alias
selector) is validated by string comparison
against the dynamic href value.

 Note: To specify a comma separated selector list,
 use the version extended naming format, not the
 -versions option.
```

RETURN VALUE

```
This command does not return a Tcl value in text mode.  If the command
is unable to run,  DesignSync throws an error explaining the failure.

If whereused vault is unable to process a reference, you will see the
reason for failure during command processing and the command adds
the sync URL of the failed module to the wuFail list.

In list mode, the list of modules is returned in an array of
name/value pairs. If a reference cannot be processed, the command
adds an error property containing the reason for the failure and the
module is added to the wuFail list.
```

SEE ALSO

```
selectors, addhref, addbackref, edithrefs, rmhref, showhrefs
```

EXAMPLES

addbackref

addbackref Command

NAME

```
addbackref          - Adds whereused support to targets of
                      hierarchical references
```

DESCRIPTION

```
This command traverses the hierarchical references from a module to
any of the following targets (toargument): sub-module version; legacy
module configuration; or DesignSync vault, identified by a selector;
and creates a back reference. The addbackref command follows the
```

hierarchy forward. This is complementary functionality to the whereused command which follows this hierarchy backwards to provide the functionality for locating where a module is used.

When hierarchical references are added, back references are automatically created to support the whereused command which displays a list of references to a particular target.

You can manually create back references in cases where the references were not created automatically, for example:

- The reference was created before whereused was implemented.
- The user creating the reference did not have permission to modify the referenced module.
- The referenced module was unreachable when the reference was created.

When you create back references, the server creates a property that stores the name of the referencing module on the target. When whereused is run, those references are resolved to specific module versions.

Back references are always explicit references created between a module and a target. An explicit reference exists when there is a direct reference link between the module and the target. An implicit reference exists when the module and target are not directly connected, but a reference exists within the module's hierarchy.

When -recursive is used with a module version or branch argument, all references from the module version, even references within sub-modules, are traversed until any of the following targets are reached:

- legacy modules
- DesignSync vaults
- modules located on pre-V6R2009 SyncServer

When one of these targets is reached, the back reference is created on the target, but any references within the target are not traversed.

If a back reference cannot be created, the command displays the failure reason and continues processing.

Important: The addbackref command can only be run on modules that exist on V6R2009 or later SyncServers.

This command is subject to access controls on the server. The addbackref command requires browse access to the objects in the module hierarchy. See the ENOVIA Synchronicity Access Control Guide for details.

SYNOPSIS

```
addbackref [-[no]recursive] <argument>
```

ARGUMENTS

- Branch/Version URL
- Category URL
- Module URL

Branch/Version URL

```
<branch/
 version URL>     URL of the module branch or version on which to
                  create the back references.
                  Specify the URL as follows:
                  sync[s]://<host>[:<port>]/Modules/[<Category>...]
                   /<module>;[<branch>:][<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, <branch> identifies the optional
                  branch, <version> identifies the optional
                  version selector.

                  Note:  You may select a branch, version
                 selector or both within the URL. If you
                 specify a branch selector, it resolved to a module
                 version.
```

Category URL

```
<category URL>    URL of the category containing the modules to
                  process. Back references are created for all
                  modules within the specified category.
                  Specify the URL as follows:
                  sync[s]://<host>[:<port>]/Modules/[<Category>...]
                  where <host> is the SyncServer on which the
                  module resides, <port> is the SyncServer
                  port number, and <Category> identifies the
                  category path.

                  Notes:
                  o If sync[s]://<host>[:<port>]/Modules/ is
                    specified with the -recursive option, it is
                    equivalent to selecting all the categories on
                    the server as well as any uncategorized modules.

                  o When the -recursive option is specified with a
                    category argument, the addbackref command
                    recursively traverses the categories, but does
                    not recursively traverse the modules.
```

Module URL

```
<module URL>        URL of the module on which to create the
                    back references. 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.
```

OPTIONS

- -[no]recursive

-[no]recursive

```
-[no]recursive      Determines whether the back references are created
                   for explicit references or both explicit and
                   implicit references. This option is ignored for
                   <Module URL> arguments.

                    -norecursive processes a single reference layer.
                   This means that only explicit references are
                   resolved, implicit or nested references are not
                   resolved. (Default)
                      o Category URL - All explicit references within
                        the modules of the category are processed, but
                        modules within sub-categories are not.

                      o Module URL - All explicit references for all
                        versions of all  branches of the module are
                        processed and back references are created.

                      o Branch or version URL - All explicit references
                        are processed and back references are created.

                    -recursive processes all explicit and implicit
                     references, depending on the specified argument.
                      o Category URL - All explicit references of the
                        modules in the category and any sub-categories
                        are processed and back references are created.
                        Sub-modules of the processed modules are not
                        processed. Implicit references within modules
                        are not processed.

                      o Module URL - The -recursive option is not
                        valid for a Module URL with no branch or
                        version qualifier.

                      o Branch or version URL - All explicit and
                        implicit references are processed and back
```

references are created.

RETURN VALUE

If the addbackref command is successful, DesignSync returns an empty string (""). If the command cannot run, DesignSync throws an error message explaining the failure.

If a back reference cannot be created, the command displays the failure reason and continues processing.

SEE ALSO

whereused, addhref, edithrefs, showhrefs

EXAMPLES

- Adding Back References to a Specific Module
- Adding Back References to All Modules Within a Category

Adding Back References to a Specific Module

This example shows creating a back reference on a specific module branch.

```
dss> addbackref sync://srv2.ABCo.com:2647/Modules/Chip;Alpha:

 Beginning addbackref operation ...

 Resolved sync://srv2.ABCo.com:2647/Modules/Chip;Alpha: to module
 version 1.5.1.3


 Processing modules on server "sync://srv2.ABCo.com:2647"...


 Processing module "sync://srv2.ABCo.com:2647/Modules/Chip" at
 version(s) 1.5.1.3...


 ADDED: back reference module sync:///Modules/Chip added to
 sync://srv2.ABCo.com:2647/Modules/top

 Finished addbackref operation.
```

Adding Back References to All Modules Within a Category

```
This example shows adding back references to all the modules within a
category.

dss> addbackref sync://srv2.ABCo.com:2647/Modules/Cadence

 Beginning addbackref operation ...

 Processing modules on server "sync://srv2.ABCo.com:2647"...

 Processing modules in category
 "sync://srv2.ABCo.com:2647/Modules/Cadence"...

 Processing module
 "sync://srv2.ABCo.com:2647/Modules/Cadence/DSDIIModtest" at all
 versions...

 ADDED: back reference module sync:///Modules/Cadence/DSDIIModtest
 added to sync://srv2.ABCo.com:2647/Modules/Cadence/Groups/LibGroup1

 ADDED: back reference module sync:///Modules/Cadence/DSDIIModtest
 added to sync://srv3.ABCo.com:2647/Modules/Cadence/Modtest

 Processing module "sync://srv2.ABCo.com:2647/Modules/Cadence/LibSec"
 at all versions...

 Processing module "sync://srv2.ABCo.com:2647/Modules/Cadence/LibInit"
 at all versions...

 ADDED: back reference module sync:///Modules/Cadence/LibInit added
 to sync://srv2.ABCo.com:2647/Modules/Cadence/LibSec

 Finished addbackref operation.
```

addhref

addhref Command

NAME

```
addhref             - Creates a hierarchical reference between
                      modules
```

DESCRIPTION

- Adding Multiple Hrefs Within a Single Operation
- Adding Overriding Hrefs

```
This command creates a new module version containing the new
hierarchical references, or connections, from an upper-level module
```

(fromargument) to any of the following (toargument): submodule branch or version, external module, legacy submodule configuration, DesignSync vault, or IP Gear deliverable. DesignSync stores the hierarchical reference on the server on which the upper-level module resides.

An href can only be created when the following conditions are met:
- The items you want to connect must already exist.
- The fromargument must be a current module version populated with a dynamic selector.
- The fromargument module must be the latest version on the module branch or the auto-merge must be enabled.
- The fromargument module branch cannot be locked by another user.
- The servers that hosts the modules being connected must be available.
- If the toargument is an external module, legacy module, IP Gear deliverable, or DesignSync vault, the relative path must be unique within the scope of the upper-level module and cannot be nested within another hierarchically referenced external module, legacy module, IP Gear deliverable, or DesignSync vault. They cannot be contained within another relative path of the same module.
- The toargument must use asingle selector. It cannot use a selector list.

Notes:
- The addhref command does not update individual workspaces. It only updates the module on the server. For example, if you fetch a module into your work area and later a hierarchical reference is added to the module, the newly added hierarchical reference is not added to the local definition. To identify hierarchical references added on the server, run the showstatus command. To synchronize your local work area with the server, run 'populate' on your workspace.

- When specifying Synchronicity URLs, you should enter host names consistently; always use the same host/domain name for a particular host, regardless of whether the name is the actual machine name or a DNS alias. For access outside of the origin server's LAN, you should use fully qualified domain names.

- You can add or modify existing hrefs in a module snapshot using the addhref command. This creates a new module version on the snapshot branch.

- If the hierarchical reference toargument is a static selector, modifications to the submodule cannot be checked in for workspaces that use the populated href.

- For external modules, the sync URL does not include the host/port information. Instead it includes the external module type. The type is an administrator-defined string that DesignSync operations use to locate the external module interface provided by the administrator. The module interface defines the operation performed on the external module to which the hierarchical reference refers.

- When specifying external modules, you must supply the -name value. When creating other references, if the -name value is not

supplied the addhref command uses the tail of the path portion of the module URL. External module URLs do not provide such information. Instead, the contents of the URL string contains data meaningful to the external module interface that defines operations using a different configuration management system. As such the addhref command cannot derive a meaningful default name.

- To take advantage of a performance enhancement for references on the same server, specify the same domain name for the fromargument and toargument. (A domain name is needed when the referenced submodule is on the same server so that the reference can be followed by users outside the server's LAN.) If you specify a different domain name for fromargument and toargument, the addhref operation succeeds, but the connection isn't optimized for the local server connection.

- Adding hrefs can affect mirrors containing the upper-level module. For more information on mirrors, see the mirror commands.

- To perform a collection of hierarchical reference changes, including adding hrefs, removing hrefs and modifying hrefs, see the edithrefs command.

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 Project Administration User's Guide.

This command supports the command defaults system.

Adding Multiple Hrefs Within a Single Operation

The add href command can be used to add multiple hrefs in a single operation to a module. You can add multiple hrefs in one of three different ways.

Important: When adding multiple hrefs, the toarguments must be either modules or external modules.

- Populate the modules to reference in your workspace and specify the toarguments as module instances.
- Create a file containing a list of toarguments.
- Specify the list of toarguments as a Tcl list.

When you specify the toarguments as module instances, you can specify the module instance by the full path to the module instance, the module instance itself, or, if the module name is unique in the workspace, the module name. When hrefs are created this way, all the reference information comes from the workspace. This includes vault URLs, selectors, relative path information, and static version numbers. This allows you to model the module hierarchy in your workspace and translate that concept to the server.

```
When you specify the toarguments as a file or as Tcl list, you
specify each entry in the following format:
  url <toargument> [name <name>] [relpath<rel_path>]
  [selector<selector>]

  Where:
  url is a valid toargument as described in the toarguments section.
  name is optionally specified using the same guidelines as the -name
    option.
  relpath is optionally specified using the same guidelines as the
    -relpath option.
  selector is a valid selector.
```

Adding Overriding Hrefs

```
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.

To add an overriding hierarchical reference, you add the reference to
a parent module within the module hierarchy.

Notes:

o The relpath of the hierarchical reference is used to
  determine which sub-module to replace.

o In order for the overriding href to be used by the system, you must
  populate recursively from the module containing the override
  href or higher within the module hierarchy.
```

SYNOPSIS

```
addhref <fromargument> <toargument>  [-name <name>]
[-relpath <rel_path>] [-selector] [-rootpath <path>]

addhref <fromargument> [-file <filename> |-tcllist
<toargument_specification_list>] [-relpath <rel_path>] [-rootpath <path>]

addhref [-selector] [--] <fromargument> <toargument> [<toargument>...]
```

ARGUMENTS

- From Arguments
- From Argument: Server Module Version
- From Argument: Workspace Module
- To Arguments

- To Argument: DesignSync Vault
- To Argument: External Module
- To Argument: IP Gear Deliverable
- To Argument: Legacy Module Configuration
- To Argument: Server Module Version
- To Argument: Workspace Module(s)

```
This command has two type of arguments: "fromarguments" and
"toarguments."
```

From Arguments

```
The fromargument specifies the URL of the upper-level module version
from which you want to create the connection.  You may specify one of
the following arguments:
```

From Argument: Server Module Version

<server module version>	Specify the URL as follows: sync[s]://<host>[:<port>]/<vaultPath>[;<selector>] where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, <vaultPath> identifies the module, and <selector> is the branch and version information. You may use this format to specify a module, module branch or module version. The default branch is "Trunk." The default version is "Latest".

From Argument: Workspace Module

<workspace module>	Specify the workspace module. The module must be populated in your workspace, contain the latest server version, and use a dynamic workspace selector to add an href. If you do not have the latest version, you can create an href only if you have auto-merge selected as the default checkin setup with SyncAdmin. For more information on selecting auto-merge as the default checkin option, see The ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide: Site Options. Note: When you specify a workspace module, the selector of the workspace is used by the command, therefore the -selector switch is not applicable.

To Arguments

The toargument is a server module version, external module, legacy module configuration, IP Gear deliverable, DesignSync vault, or one or more module instances (workspace module) to which you want to create a connection.

Important: When creating an href to a server that uses SSL redirection, you must use the DesignSync server SSL port (syncs://) in the URL of the toargument.

Note: You must specify at least one toargument. The toargument can be any of the following arguments or a filename or tcllist (specified with -file or tcllist).

If your toargument contains a static selector, then any changes made to the objects contained within a workspace containing the populated reference cannot be checked in with a module recursive checkin of the parent module.

The toargument must be a single selector. You cannot use a selector list.

You may specify one of the following arguments:

To Argument: DesignSync Vault

<DesignSync vault> Specifies the DesignSync vault. Specify the URL as follows:
sync[s]://<host>[:<port>]/<vaultPath>[;<selector>]
where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, and <vaultPath> identifies the module. This is not valid argument for adding multiple toarguments within a single addhref command.

To Argument: External Module

<external module> Specifies 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 change management system but available for viewing and integration through DesignSync. Specify the 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 change management system or to DesignSync. Note: When specifying an external module, you must specify the -name option.

To Argument: IP Gear Deliverable

<IP Gear deliverable>	Specifies an IP Gear deliverable. The format for specifying an IP Gear deliverable is: sync[s]://<host>[:<port>]/Deliverable/<ID> where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, and <ID> is the deliverable ID number; for example, sync://publisher.ipgsrvr1.com:2647/Deliverable/142. This is not valid argument for adding multiple toarguments within a single addhref command.

To Argument: Legacy Module Configuration

<legacy module configuration>	Specifies the URL or path of the submodule to which you wish to create the connection. If this is a workspace, the reference uses the same selector as the workspace. This is not valid argument for adding multiple toarguments within a single addhref command.

To Argument: Server Module Version

<server module version>	Specifies the URL of the module version to which you want to create the connection. Specify the URL as follows: sync[s]://<host>[:<port>]/<vaultPath>[;<selector>] where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, <vaultPath> identifies the module, and <selector> is the branch and version information. You may use this format to specify a module, module branch or module version. The default branch is Trunk. The default version is Latest.

To Argument: Workspace Module(s)

<workspace module(s)>	Specifies one or more module instance names or submodule paths to which you wish to create the

```
connection.  The reference uses the same selector
as the workspace. The modules must be fetched in
your workspace to add an href.

Note: If you specify a relpath, The path can only
be the base directory for a single module. If you
specify more than one workspace module, you
cannot specify a relpath.
```

OPTIONS

- -file|-tcllist
- -name
- -relpath
- -rootpath
- -selector

-file|-tcllist

```
 -file <filename> |  This option allows you to specify a set of
  -tcllist <list>    toarguments.  You can specify this set either as
                     a Tcl list or a flat file containing a Tcl
                     formatted list.

                     When specifying a file, each line must be either
                     a comment or specify the details of a single
                     href to be added to the module identified in the
                     fromargument.

                     When specifying a TCLlist, each entry specifies
                     the details of a single href to be added to the
                     module identified in the fromargument.

                     The format for both a file and a Tcl list is the
                     same, a Tcl list of Tcl lists:
                     {{url <toargument>} {name <name>}
                     {relpath<rel_path>} {selector <selector>}}

                     Where: url is a valid toargument as described in
                     the toarguments section.
                     name is optionally specified using the same
                     guidelines as the -name option.
                     relpath is optionally specified using the same
                     guidelines as the -repath option.
                     selector is a valid selector.

                     You can not specify the -file option when you
                     have a explicitly specified a toargument at the
                     command line.

                     The -selector option is mutually exclusive with
                     both the -file and -tcllist options.
```

Note: When specifying the -tcllist option, you must use the stcl or stclc command line interface. dss/dssc will not process the list correctly and return an error.

-name

-name

Specifies the href name. This is the name by which this href is known relative to the parent module. This name is required for external module references. For any other type of reference, if the 'name' option is not specified, the href name defaults to the name of the target module, legacy module configuration, IP Gear deliverable (ID value), or DesignSync vault.

Note: The name value must be unique within the scope of the fromargument. If an href already exists with the name specified, this command returns an error.

Note: Hierarchical reference names may only contain printable characters and cannot contain a space or any of these restricted characters:

~ ! ? @ # $ % ^ & * ( ) , ; : | ` ' " = [ ] / \ < >

When multiple workspace module instances are specified, the name value is automatically set to the module instance name.

-relpath

-relpath <rel_path>

Indicates the path from the upper-level module to the module, legacy module configuration, IP Gear deliverable, or DesignSync vault. The populate command uses this path when recursively fetching items into your work area to create additional levels of hierarchy.

Paths:
- Cannot be absolute.
- Should not contain at signs (@), number signs (#), back slashes (\), question marks (?), asterisks (*), semicolons (;), ampersands (&), pipes (|), or square brackets ([ ]).
- Can be empty (""). Specify an empty path if you do not want to fetch the contents of the submodule into your local work area. This behavior may be desirable if you want to reference a ProjectSync project that tracks defects against a CAD tool.

Legacy modules paths have these additional restrictions:
o Cannot be './'.
o Must be unique within the scope of the upper-level module and cannot be nested nested within another hierarchically referenced legacy module, IP Gear deliverable, or DesignSync vault. They cannot be contained within another relative path of the same configuration.

If the relative path is not supplied, it is defined from the toargument and fromargument. If both the toargument and the fromargument are workspace modules, the relative path is the current relative path between the two modules. If either the toargument or the fromargument is a workspace module, the relpath uses the href name.

The -relpath argument is not applicable when fromargument is a workspace module and the toargument is one or more workspace modules because the relpath is automatically determined.

-rootpath

-rootpath <path>	The relative path prefix for all added hierarchical references. The rootpath when specified with a toargument, is the relative path for the toargument. When rootpath is specified without a toargument, it becomes the default relative path for all added hrefs. Note: You cannot specify a rootpath when adding a reference to an external module or when specifying multiple module workspace instances.

-selector

-selector <selector>	Specifies the selector, or selector list. The -selector should only be used with a module instance argument when the user needs to override the persistent selector of the workspace. For server arguments, the selector should be specified in the object URL. For more information about selectors, see the selectors topic. The -selectors option is mutually exclusive with the -file and -tcllist options.

RETURN VALUE

```
If the addhref command is successful, DesignSync returns an empty
string ("").  If the command fails, DesignSync returns an error
explaining the failure. When adding multiple hrefs, if any of the
hrefs are not added, the entire command fails.
```

SEE ALSO

```
rmhref, edithrefs, showhrefs, showstatus, command defaults, mirror
,
```

EXAMPLES

- Creating a Module Hierarchy from a Workspace
- Creating a Module Hierarchy without Staging in the Workspace
- Using Empty Relative Paths
- Adding Multiple Hrefs from a Tcl List

Creating a Module Hierarchy from a Workspace

```
This example creates this hierarchy:
Chip           <= uses the default configuration
  ALU          <= uses the default configuration
  CPU          <= uses the Gold branch
    ROM        <= uses the Gold version

In this example Chip has hrefs to ALU and CPU.  CPU has an href to
ROM.

The modules are individually pre-populated in the workspace in the
following directory structure:

Directory        Module instance
---------        ---------------
chip             Chip0
chip/alu         ALU%0
chip/cpu         CPU%0
chip/cpu/rom     ROM%0

Notes:
 o After you have created the hierarchy on the server with the
   addhref command, you must recursively populate Chip%0 in the
   workspace to recreate the hierarchical design.

 o Because there is only one instance of each module in the workspace
   root, you do not need to specify the %0 module instance indicator
```

```
   to the addhref command. The hierarchy is constructed from the
   bottom up.

 o The href to the ROM module is to the version tag Gold.  Regardless
   of the mutability of this tag, the selector is static. Thus if you
   were to make any changes to the ROM module, they could not be
   checked in when an upper-level module was checked in recursively.

dss> addhref CPU ROM

 Beginning addhref operation ...

 sync://srvr2.ABCo.com:2647/Modules/CPU: Added hierarchical reference(s):
    Name:           ROM
    Object:         sync://srv2.ABCo.com:2647/Modules/ROM
    Type:           Module
    Selector:       Gold
    Version:        1.2.1.1
    Relative Path:  rom

sync://qelwsun14:30125/Modules/CPU: Created new module version 1.3.

dss> addhref Chip ALU CPU

 Beginning addhref operation ...

 sync://srvr2.ABCo.com:2647/Modules/Chip: Added hierarchical reference(s):
    Name:           ALU
    Object:         sync://srvr2.ABCo.com:2647/Modules/ALU
    Type:           Module
    Selector:       Trunk:
    Version:        1.2
    Relative Path:  alu

    Name:           CPU
    Object:         sync://srv2.ABCo.com:2647/Modules/CPU
    Type:           Module
    Selector:       Gold:
    Version:        1.2
    Relative Path:  cpu


sync://qelwsun14:30125/Modules/ChipDesign/Chip: Created new module
version 1.3.

Notice: A populate of module
'sync://srvr2.ABCo.com:2647/Modules/Chip' is necessary to update the
workspace with the new hrefs.

Finished addhref operation.
```

Creating a Module Hierarchy without Staging in the Workspace

```
This example creates this hierarchy:
Chip           <= uses the default configuration
  ALU          <= uses the default configuration
  CPU          <= uses the Gold branch
  ROM          <= uses the Gold version

It guides you through the creation of the connections between the
Latest version of the Chip module and Latest version of the ALU
module, between the Latest version of the Chip module and the Gold
branch version of the CPU module, and the Gold version of the ROM
module.

Note: The latest version on the Gold branch of CPU is used, meaning
this is a dynamic module version which can be updated in the
workspace. The Gold version of ROM, however, is fixed.  Modifications
in the workspace for this module cannot be checked in, but if the tag
is moved to a different version, the version will be updated to the
new module version during the next populate operation that updates
ROM.

This addhref command, creating the hierarchy between Chip and ALU
is created using the Chip%0 instance in a workspace.

dss> addhref -name ALU -relpath ALU Chip%0 \
  sync://srvr2.ABCo.com:2647/Modules/ALU;Trunk:Latest

 Beginning addhref operation ...

sync://srvr2.ABCo.com:2647/Modules/Chip: Added hierarchical reference:
     Name:           ALU
     Object:         sync://srvr2.ABCo.com:2647/Modules/ALU
     Type:           Module
     Selector:       Trunk:
     Version:        1.1
     Relative Path:  ALU

sync://srvr2.ABCo.com:2647/Modules/Chip: Created new module version 1.3.

 Finished addhref operation.

These addhref commands, creating the hierarchy between Chip and CPU
and ROM specify the ROM module version Gold and the CPU branch
version Gold.

dss> addhref -name ROM -relpath ROM \
   sync://srv2.ABCo.com:2647/Modules/CPU;Gold: \
   sync://srv2.ABCo.com:2647/Modules/ROM;Gold \

 Beginning addhref operation ...

  sync://srv2.ABCo.com:2647/Modules/CPU: Added hierarchical reference:
     Name:           ROM
     Object:         sync://srv2.ABCo.com:2647/Modules/ROM
     Type:           Module
     Selector:       Gold
     Version:        1.4
     Relative Path:  /ROM
```

```
 sync://srv2.ABCo.com:2647/Modules/CPU: Created new module version 1.3.

 Finished addhref operation.

dss> addhref -name CPU -relpath CPU
 sync://srv2.ABCo.com:2647/Modules/Chip;1.3 \
 sync://srv2.ABCo.com:2647/Modules/CPU;Gold: \

 Beginning addhref operation ...

 sync://srv2.ABCo.com:2647/Modules/Chip: Added hierarchical reference:
     Name:           CPU
     Object:         sync://srv2.ABCo.com:2647/Modules/CPU
     Type:           Module
     Selector:       Gold:
     Version:        1.3
     Relative Path:  CPU

 sync://srv2.ABCo.com:2647/Modules/Chip: Created new module version 1.4.

 Finished addhref operation.

 This command shows the hierarchy of the modules.

dss>  showhrefs -hrefmode dynamic -recursive \
  sync://srv.2.ABCo.com/Modules/Chip;Trunk:Latest

 Beginning showhrefs operation ...

 Showing hrefs of module
 sync://srv2.ABCo.com:2647/Modules/Chip;Trunk:Latest (1.4) ...

sync://srv2.ABCo.com:2647/Modules/Chip;1.1:Latest: Href mode is
  dynamic.

Name  Url                                        Version  Type  Relative Path
-------------------------------------------------------------------------
ALU   sync://srv2.ABCo.com:2647/Modules/ALU  Trunk:   Module  ALU
CPU   sync://srv2.ABCo.com:2647/Modules/CPU  Gold:     Module  CPU


sync://srv2.ABCo.com:2647/Modules/ALU;Trunk: Module has no
 hierarchical references.

=========================================================================

Showing hrefs of module sync://srv2.ABCo.com:2647/Modules/CPU;Gold
 (1.3)...

sync://srv2.ABCo.com:2647/Modules/CPU;Gold: Href mode is dynamic.

Name  Url                                 Version  Type    Relative Path
-------------------------------------------------------------------------
ROM   sync://srv2.ABCo.com:2647/Modules/ROM  Gold       Module  /ROM
```

```
sync://srv2.ABCo.com:2647/Modules/ROM;Gold:: Module has no
   hierarchical references.

Finished showhrefs operation.


Once you have created the hierarchical references, recursively fetch
the updated version of the Chip module to update the references in
your workspace.

dss> populate -get -recursive -hrefmode dynamic Chip%0

Beginning populate operation...


Populating objects in Module Chip%0 with base directory
  /home/rsmith/MyModules/chip...


Fetching contents from selector 'Trunk:', module version '1.4'
...
Creating Sub Module Instance 'ALU%1' with base directory
 '/home/rsmith/MyModules/chip/ALU'

Creating Sub Module Instance 'CPU%1' with base directory
'/home/rsmith/MyModules/chip/CPU'
Finished populate of Module Chip%0 with base directory
/home/rsmith/MyModules/chip

=====================================================

Populating objects in Module ALU%1 with base directory
  /home/rsmith/MyModules/chip/ALU...


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

=====================================================

 Creating Sub Module Instance with
   Base Directory  = /home/rsmith/MyModules/chip/ALU
   Name            = ALU
   URL             = sync://srv2.ABCo.com:2647/Modules/ALU
   Selector        = Trunk:
   Instance Name   = ALU%1
   Metadata Root   = /home/rsmith/MyModules
   Parent Instance = Chip%0


 Populating objects in Module ALU%1 with base directory
  /home/rsmith/MyModules/chip/ALU...


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

 /alu.h: Success - Checked Out  version: 1.1
 /alu.c: Success - Checked Out  version: 1.2
```

```
/alu.doc: Success - Checked Out  version: 1.2

ALU%1: Version of module in workspace updated to 1.4

=====================================================

Creating Sub Module Instance with
   Base Directory  = /home/rsmith/MyModules/chip/CPU
   Name            = CPU
   URL             = sync://srv2.ABCo.com:2647/Modules/CPU
   Selector        = Gold:
   Instance Name   = CPU%1
   Metadata Root   = /home/rsmith/MyModules
   Parent Instance = Chip%0


Populating objects in Module CPU%1 with base directory
  /home/rsmith/MyModules/chip/CPU...

Fetching contents from selector '1.2', module version '1.2'

/cpu.h: Success - Checked Out  version: 1.1
/cpu.c: Success - Checked Out  version: 1.1
/cpu.doc: Success - Checked Out  version: 1.1

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

Creating Sub Module Instance 'ROM%1' with base directory
  '/home/rsmith/MyModules/chip/CPU/ROM'

Finished populate of Module CPU%1 with base directory
  /home/rsmith/MyModules/chip/CPU


=====================================================

Creating Sub Module Instance with
   Base Directory  = /home/rsmith/MyModules/chip/CPU/ROM
   Name            = ROM
   URL             = sync://srv2.ABCo.com:2647/Modules/ROM
   Selector        = Gold
   Instance Name   = ROM%1
   Metadata Root   = /home/rsmith/MyModules
   Parent Instance = CPU%1


Populating objects in Module ROM%1 with base directory
 /home/rsmith/MyModules/chip/CPU/ROM...


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

/rom.doc: Success - Checked Out  version: 1.2
/rom.c: Success - Checked Out  version: 1.2
/rom.h: Success - Checked Out  version: 1.1

ROM%1: Version of module in workspace updated to 1.4
```

```
  Finished populate of Module ROM%1 with base directory
   /home/rsmith/MyModules/chip/CPU/ROM


  Finished populate operation...

   {Objects succeeded (12)} {}
```

Using Empty Relative Paths

```
 This example illustrates how you might track defects in your
 design tools that affect your design. The ALU team relies on some
 critical design tools, for example, DesignCompiler, that are not part
 of their project. They use ProjectSync to track defects in the current
 versions of these tools. When team members do a ProjectSync hierarchical
 query on ALU, they want all DesignCompiler defects to be reported.

 Notes:
 - This example assumes that the ALU team has set up the DesignCompiler
   project in ProjectSync. The DesignCompiler project exists only to
   have notes attached to it; it does not have any data files checked
   into it.
 - This example uses an empty relative path. The empty relative path
   ensures that when the ALU team recursively fetches the ALU module
   with the hcm get command that it does not fetch the contents of the
   DesignCompiler module into the local work area.

   dss> addhref -name DesignCompiler -relpath "" \
   sync://srv2.ABCo.com:2647/Modules/ALU
   sync://tools.ABCo.com:2647/Tools/DesignCompiler;Trunk:Latest

  Beginning addhref operation ...

  sync://srv2.ABCo.com:2647/Modules/ALU: Added hierarchical reference:
     Name:           DesignCompiler
     Object:         sync://tools.ABCo.com:2647/Modules/Tools/ \
                   DesignCompiler
     Type:           Module
     Selector:       Trunk:
     Version:        1.1
     Relative Path:

  sync://srv2.ABCo.com:2647/Modules/ALU: Created new module version
   1.3.

  Finished addhref operation.
```

Adding Multiple Hrefs from a Tcl List

```
 This example shows an operation that adds multiple hrefs within a
```

```
 single operation by using the Tcl list capability.  This example
 shows entering the Tcl list on the command line with the -tcllist,
 but you could enter the same information, formatted the same way,
 into a file and specify it with the -file option.

 Note:  When specifying the -tcllist option, you must use the stcl or stclc
 command shell.

 This example creates this hierarchy:
 Chip           <= uses the default configuration
   ALU          <= uses the default configuration
   CPU          <= uses the Gold branch
     ROM        <= uses the Gold version

 stcl> addhref Chip%0 -tcllist {{url
 sync://srv2.ABCo.com:2647/Modules/ALU name ALU relpath alu}
 {url sync://srv2.ABCo.com:2647/Modules/CPU name CPU relpath
 cpu selector Gold:} {url sync://srv2.ABCo.com:2647/Modules/ROM
 name ROM relpath cpu/rom selector Gold}}

Beginning addhref operation ...

sync://srv2.ABCo.com:2647/Modules/Chip: Added hierarchical reference(s):
    Name:           ALU
    Object:         sync://srv2.ABCo.com:2647/Modules/ALU
    Type:           Module
    Selector:       Trunk:
    Version:        1.2
    Relative Path:  alu

    Name:           CPU
    Object:         sync://srv2.ABCo.com:2647/Modules/CPU
    Type:           Module
    Selector:       Gold:
    Version:        1.2
    Relative Path:  cpu

    Name:           ROM
    Object:         sync://srv2.ABCo.com:2647/Modules/ROM
    Type:           Module
    Selector:       Gold
    Version:        1.2.1.1
    Relative Path:  cpu/rom


 sync://srv2.ABCo.com:2647/Modules/Chip: Created new module version 1.7.

 Finished addhref operation.
```

edithrefs

edithrefs Command

NAME

```
edithrefs           - Updates the hierachical references for a module
```

DESCRIPTION

- Adding a Hierarchical Reference
- Removing a Hierarchical Reference
- Changing a Hierarchical Reference
- File Format for Editing Hierarchical References
- Running in Interactive Mode
- Understanding the Output

The command modifies; adds, removes, and changes, the hrefs associated with a module in a single batch operation or outputs a list of the current hierarchical references which can be edited to fed back into the command to modify the hierarchical references for a module.

There are four types of actions that can be processed within the file, including no action.

- Adding a hierarchical reference (add)
- Changing a hierarchical reference (change)
- Removing a hierarchical reference (remove)
- No change to this hierarchical reference (none)

When batch editing a list of hierarchical references, you can either start from a blank file and add the hierarchical references to update or you can generate a list of hierarchical references and modify the hrefs to update using the edithrefs command.

The edithrefs command can be run using a saved file containing the modified list of hierarchical references or interactively, which locks the module while you edit the references.

When you generate a file, DesignSync provides comments within the file to help you modify it appropriately. The amount of help provided varies with the report mode specified for the command. For more information on report modes, see Understanding the Output.

Interactive mode uses the default editor. The default editor can be specified with SyncAdmin.

Important: When working in interactive mode, the module is locked for the duration of the edithref command. This may make interactive mode a less attractive choice for a module that is already actively being used.

Tip: While edithref does not automatically lock the module when not run in interactive mode, DesignSync recommends the best practice of locking the module before making structural changes to minimize the risk of error. For more information on locking a module, see the lock command.

The file format is discussed in detail in "File Format for Editing Hierarchical References."

Note: All lines are examined. If any line is unchanged, it is implicitly processed by DesignSync as "None," even if the None action is not explicitly specified. If a new or changed line is discovered, even if not explicitly marked, it will still be processed accordingly.

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 Adminstration User's Guide.

This command supports the command defaults system.

Adding a Hierarchical Reference

Specifying action "add" (a) on a hierarchical reference line in the input file adds the hierarchical references to the module.

When you add a hierarchical reference, DesignSync requires you to specify the URL. All other fields have default values which can be used.

Note: For external modules you must specify both URL and name values.

- action - If not specified, DesignSync examines the line. If no href exists with the name specified, DesignSync treats the line as an implicit add.
- name - If not specified, DesignSync uses the target module name, legacy module configuration, IP Gear deliverable, or DesignSync vault. For external modules, there is no default, so the name must be specified.
- relpath - If not specified, DesignSync uses the default value as explained in the addhref command, relpath argument description.
- selector - If not specified, this remains an empty string, for which DesignSync substitutes the default selector, Trunk: at processing time.

Important: When adding a new hierarchical reference, you cannot use the name of an existing href, either implicitly, or explicitly, even if you are deleting the href using that name within the same input file.

Adding a hierarchical reference adds the parent module to the whereused information of the referenced module.

Removing a Hierarchical Reference

Specifying the action "remove" (r) on a hierarchical reference line in the input file removes the hierarchical reference from the module.

When you remove a hierarchical reference, DesignSync requires you to specify two properties, the action property with the value "remove" and the name property.

Changing a Hierarchical Reference

Specifying action "change" (c) on a hierarchical reference line in the input file changes an existing hierarchical reference.

When you change the hierarchical reference, you must provide the name, even if you are changing the href name. All other properties can be optional.

- action - If not specified, DesignSync examines the line. If an href exists with the name specified, DesignSync treats the line as an implicit change.
- relpath - When not specified, DesignSync keeps the existing relpath.
- selector - When not specified, DesignSync keeps the existing selector unless a different selector is specified as part of the url property. If the url property includes a selector and the selector property is set, the value of the selector property is the one used for the changed href and the URL selector is ignored.
- newname - If not specified, DesignSync keeps the existing name. If specified, DesignSync changes the name of the hierarchical reference to the name specified with the newname value. The newname key/value pair is only valid for changing a hierarchical reference.

Note: If no properties are changed in the change line, the hierarchical reference is not modified.

Changing a hierarchical reference adds the parent module to the whereused information of the referenced module.

File Format for Editing Hierarchical References

The file format created by the edithrefs command as an output file (-output) or read and processed by the command as an input file (-input) contains a single line for every href being modified. Optionally, it can contain a single line for hrefs that remain unchanged. This allows the user to work from a generated list of hierarchical references, changing only the ones that need to be changed without needing to remove the unchanged lines.

Generally, the file consists of one line per hierarchical reference

containing the following key/value pairs:

Key	Description
o action -	The type of action allowed. The allowed values are: o add/a o remove/r o change/c o none/n
o name -	Name of the hierarchical reference. This must conform to DesignSync naming conventions. For more details, see the addhref command.
o url -	URL to the referenced object, which can be a submodule branch or version, external module, legacy submodule configuration, DesignSync vault, or IP Gear deliverable.
o relpath -	The path from the upper-level module to the referenced object used when recursively fetching the upper-level module into a workspace.
o selector-	A valid selector. For more details, see the selector topic. Note: If existing hrefs use a selector list, they will continue to work, matching the first valid selector, but you cannot add or change an href to use a selector list.

You can use the hash mark (#) in the first column of the line to place a comment in the file. Blank lines are not processed. If any value requires a space, surround the value with double quotes ("").

This sample shows the syntax of the input file.

```
#This is a sample input file structure

#This line adds a hierarchical reference
action a [name <name>] url <moduleURL> [selector <selector> \
[relpath <relpath>]

#This line removes a hierarchical reference
action r name <name>

#This line changes an existing hierarchical reference
action c name <name> [url <moduleURL>] [newname <newname>]\
[selector <selector>] [relpath <relpath>]

#This line takes no action.
action n [name <name>] [url <moduleURL>] [selector <selector>] \
[relpath <relpath>]
```

Tip: Store your input files in a single directory with a descriptive filename so you can use them to quickly look up the name, relapth, or selector of the created hiearachical references, or so you can modify them for reuse if you change the module hierarchy.

Running in Interactive Mode

The edithrefs command provides the ability to run interactively. In

interative mode, DesignSync generates a current list of hrefs for the module which the user can then edit as described in the "File Format for Editing Hierarchical References" section.

Running interactively may be desirable during the initial creation phase, before the module is generally available or if the changes are quick to make. While interactive mode is in process, the module is locked so users can make neither structural nor content changes while the locker is making structural changes using the edithrefs command.

To run in interactive mode, do not specify the -input or -output file options to the edithrefs command.

Understanding the Output

The report option allows you to specify both the level of information the command outputs during processing and the amount of commenting syntax help provided in the generated output.

Note: Regardless of the report mode selected, if the edithrefs command is run in interactive mode, DesignSync will ask whether the changes made in the editor should be committed. For more information, see Example 2: Modifying Hierarchical References in Interactive Mode.

report brief mode
==========

If you run the command with the -report brief option, the edithrefs command outputs the following information:
- Failure messages.
- Warning messages.
- Success/failure/skip status.

Additionally, if the -report brief option is used to create an output file with the -outfile option, that file will contain only a list of hrefs. It will contain no comments on how to format the line for processing as an input file to edithrefs.

report normal mode
==========
By default, or if you run the add command with the -report normal option, the command displays all the information contained in -report brief, and the following additional information:
- URL of the module branch being locked
- operation status messages

Additionally, if the -report normal option is used to create an output file with the -outfile option, that file will contain a syntax line showing general syntax of a defined href. Additional lines in the file, show, in brief, the possible values for action property and the syntax for the newname property.

report verbose mode
==========

```
If you run the command with the -report verbose option, it provides
the same command output as -report normal, but provides a longer,
more detailed explanation for modifying the hrefs for processing. For
an example of the verbose output, see Example 4: Generating a Verbose list
of Hierarchical References.
```

SYNOPSIS

```
edithrefs [-infile <file-containing-href-edits> |
           -outfile <file-to-write-hrefs-to>]
          [-report brief|normal|verbose] [--] <argument>
```

ARGUMENTS

- Server URL
- Workspace Module

Server URL

```
<serverURL>         Specifies the URL of the upper-level module
                    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. The selector
                     provided must be a dynamic selector, otherwise a
                     new module version cannot be created.  If a
                     static selector is provided, the command exits
                     with an appropriate error.
```

Workspace Module

```
<workspace Module>  Specifies the module identifier for the module.
                    The module must have already been populated in
                    the workspace. The workspace selector must be a
                    dynamic selector.  If the persistent selector for
                    the workspace is a static selector, a new module
                    version cannot be created and the command exits
```

```
                    with an appropriate error.
```

OPTIONS

- -infile
- -outfile
- -report
- --

-infile

```
 -infile <file>      Name of the file containing the list of
                     hierarchical references to be modified.  For
                     information on the format of the infile, see
                     File Format for Editing Hierarchical References.

                     This option is mutually exclusive with the
                     -outfile option.

                     Note: If neither -input or -output is specified,
                    the command runs in interactive mode.
```

-outfile

```
 -outfile <file>     Name of the file in which the command will create
                     the list of hierarchical references that exist
                    for the specified module version. This file can
                    be modified and used by this command (with the
                    -infile option) to edit the hierarchical
                    references.

                     This option is mutually exclusive with the
                    -infile option.

                     Note: If neither -input or -output is specified,
                    the command runs in interactive mode.
```

-report

```
 -report brief |    Controls the level of additional information
  normal | verbose  reported as the command progresses. For more
                    information, see Understanding the Output.

                    Note: When used with the -outfile option, the
                    report option also controls the verbiage of the
                    help available within the outfile.
```

--

```
--                  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 runs succesfully, 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 fails, DesignSync returns an error explaining
the failure.

Note: The Command failure is not the same as a failure to modify an href.
If there is a value in the second list, you should review the log to
determine why the href failed to update.
```

SEE ALSO

```
addhref, rmhref, showhrefs
```

EXAMPLES

- Example of Generating a list of Hierarchical References
- Example of Modifying Hierarchical References in Interactive Mode
- Example of Modifying Hierarchical References From an Href List
- Example of Generating a verbose list of Hierarchical References

Example of Generating a list of Hierarchical References

```
This example shows the normal output of creating a list of
hierarchical references to edit.

This is the command that creates the output file.

stcl> edithrefs -outfile hreflist.txt -report normal Chip%0

 ## Hrefs generated from: sync://serv.ABCo.com:2647/Modules/Chip;Trunk:
 ##              Revision: 1.7
 ##     Workspace module: Chip%0
 ## Output to: /home/rsmith/MyMods/chip
 ## NOTE: it is recommended to lock the branch prior to changing hrefs.
 ##
 {} {}

This is the outfile file contents generated from the command above.
```

```
Note: The comments within the file show the general syntax of the
href lines, but do not contain the detailed information that would be
generated by -report verbose.

# Href properties in each row:
#   name <href-name> url <to-url> selector <to-selector> relpath
#   <href-relpath>
# Optional user added instructions on each row:
#   action add|change|remove|none
#   newname <value>
name ROM url sync://serv1.ABCo.com:2647/Modules/Components/ROM
  selector Gold: relpath rom
name RAM url sync://serv1.ABCo.com:2647/Modules/Components/RAM selector {}
 relpath ram
name ALU url sync://serv2.ABCo.com:2647/Modules/Components/ALU
 Trunk:Beta relpath alu
```

Example of Modifying Hierarchical References in Interactive Mode

```
This example shows using the hierarchical reference interactive mode
using the default editor on UNIX (vi) from the output file generated
in Example 1.

dss> edithrefs Chip%0

 Beginning edithrefs operation ...

 ## Locked branch for: sync://serv.ABCo.com:2647/Modules/Chip;Trunk:(1.7)

 ## Hrefs generated from: sync://serv.ABCo.com:2647/Modules/Chip;Trunk:
 ##             Revision: 1.7
 ##     Workspace module: Chip%0

 ## Invoking the editor for making href changes. After exiting the editor,
 ## you will be able to decide if you wish to submit your changes.

The editor opens containing a list of hierarchical references for
version 1.7 of module Chip. For readability, a comment before each
line will indicate what modifications have been made to the list, for
example, if the action is a, the comment will indicate that by
including the text "action: a."

# Href properties in each row:
#   name <href-name> url <to-url> selector <to-selector> relpath
#   <href-relpath>
# Optional user added instructions on each row:
#   action add|change|remove|none
#   newname <value>

#This href is being removed. action: r
action r name ROM url sync://serv1.ABCo.com:2647/Modules/Components/ROM\
  selector Gold: relpath rom
```

```
#This href is being changed.  action: c AND newname: RAMFinal
action c name RAM newname RAMFinal url\
sync://serv1.ABCo.com:2647/Modules/Components/RAM selector {}
 relpath ram

#This href is unchanged. action: n (not required)
action n name ALU url sync://serv2.ABCo.com:2647/Modules/Components/ALU\
 Trunk:Beta relpath alu

#This href is being added action: a
action a name NewROM url sync://serv2.ABCo.com:2647/Modules/Components/ROM\
 selector BuildReady: relpath rom

When the changes are completed, save the file as normal.  DesignSync
will then prompt you to submit the changes.

 Submit href changes (Y/N):
 y
 sync://serv2.ABCo.com:2647/Modules/Components/Chip: Added hierarchical
 reference(s):
    Name:           NewROM
    Object:         sync://serv2.ABCo.com:2647/Modules/Components/ROM
    Type:           Module
    Selector:       BuildReady
    Version:        1.2
    Relative Path:  rom

 sync://serv2.ABCo.com:2647/Modules/Components/Chip: Changed hierarchical
 reference(s):
    Name:           RAM => RAMFinal

 sync://serv2.ABCo.com:2647/Modules/Components/Chip: Removed hierarchical
 reference(s):
    Name:           ROM

 sync://serv2.ABCo.com:2647/Modules/Components/Chip: Created new module
 version 1.10.

 Finished edithrefs operation.
 {Objects succeeded (3)} {}
```

Example of Modifying Hierarchical References From an Href List

```
This example shows using the hierarchical reference list generated in
Example 1 making the same changes described in Example 2.

dss> edithrefs -infile hreflist.txt Chip%0
Beginning edithrefs operation ...

  sync://serv2.ABCo.com:2647/Modules/Components/Chip: Added hierarchical
 reference(s):
    Name:           NewROM
    Object:         sync://serv2.ABCo.com:2647//Modules/Components/ROM
    Type:           Module
```

```
    Selector:       BuildReady
    Version:        1.2
    Relative Path:  rom

 sync://serv2.ABCo.com:2647/Modules/Components/Chip: Changed hierarchical
 reference(s):
    Name:           RAM => RAMFinal

 sync://serv2.ABCo.com:2647/Modules/Components/Chip: Removed hierarchical
 reference(s):
    Name:           ROM

 sync://serv2.ABCo.com:2647/Modules/Components/Chip: Created new module
 version 1.10.

 Finished edithrefs operation.
 {Objects succeeded (3)} {}
```

Example of Generating a verbose list of Hierarchical References

```
This example shows the verbose output of creating a list of
hierarchical references to edit.

stcl> edithrefs -outfile hreflist.txt -report verbose hrefs main%0
 Beginning edithrefs operation ...

##
## Hrefs generated from:
   sync://sync://serv2.ABCo.com:2647/Modules/Chip;Trunk:
##             Revision: 1.11
##     Workspace module: Chip%0
## Output to: C:/Workspaces/chipdesign/chip/hreflist2.txt
## Note: To avoid conflicts, you should lock the module branch before
##       changing hrefs.
##

Finished edithrefs operation.
{} {}

 The generated file created from our hierarchy looks something like
this.
#
# Hrefs generated from: sync://serv2.ABCo.com:2647/Modules/Chip;Trunk:
#             Revision: 1.11
#     Workspace module: Chip%0
#
# The generated file using the -outfile option contains all the
# latest hrefs from the provided module branch.
# There will be one href per line with name/value pairs for the
# existing href
# properties that can be changed.
nnn #
# The following properties are output:
#   name, url, selector, relpath
```

```
#
# The following properties can be provided but are not saved with the
# href:
#   action, allowed values: add | change | remove | none
#   newname, used to change the href name (name property required
#             to identify existing href having name changed)
#
# The action property defines an explicit action.
# - If no action is provided and an href with the provided name
# exists, the
# entry is considered a change action and each provided property is
# checked
# for a change. Any entries with changes will be reflected on the
# server in
# the next module version.
# - For a change entry, only the provided properties can be changed on the
# server, except for the selector property that can be changed as part of
# a provided url. When both the url and the selector property provide the
# selector, the selector property is used.
# - If the url or selector of a module is changed, a new static version
# number is gotten and saved with href.
# - If no action and an href with the provided name does not exist, this
# entry is considered an add action.
# - An add entry does not need a name provided (except external modules)
# so an entry without the name provided is considered an add action and
# a default name based on the target url will be used.
# - To remove an href the remove action must be provided and any provided
# properties are not used.
# - Names cannot be swapped with one run of edithrefs nor can the
#same name
# be used for two entries even if one is removing it and the other
# adding it.
# - If an existing href is not provided in the file, it will not be
# affected.
# - Comments indicated by '#' used in first column of line.
#

name ROM url sync://serv1.ABCo.com:2647/Modules/Components/ROM
  selector Gold: relpath rom
name RAM url sync://serv1.ABCo.com:2647/Modules/Components/RAM selector {}
 relpath ram
name ALU url sync://serv2.ABCo.com:2647/Modules/Components/ALU
 Trunk:Beta relpath alu
```

reconnectmod

reconnectmod Command

NAME

```
reconnectmod        - Updates the hrefs from a module to a new module
```

DESCRIPTION

After you have moved a module (with the exportmod/importmod commands), you can update hierarchical references pointing to the old module to the new module. This operation does not create a new module version of the referencing modules, it modifies the reference within the module version to point at the new location. This maintains the integrity of both static and dynamic hierarchical references after a module has changed location.

You can update the references all modules or a single module, or a list of modules.

Note: When doing a move module with the mvmod command, the reconnect command is called automatically as part of that operation.

This command supports the command defaults system.

This command is subject to access controls on the server.

SYNOPSIS

```
reconnectmod  [-[no]force] [(-from <oldServerURL>) |
              (-parents <TCLlist>)] <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>     Specifies the URL of the new 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

- -force
- -from

- -parents

-force

```
 -[no]force            Determines whether to force the modulereconnect
                        to module specified with the -from option, even
                        if url/uid for the modules doesn't match the
                        information contained in the imported
                        transportable module.

                        -noforce does not recreate the href is the
                        url and uid information does not match the
                        expected module information.  (Default)

                        -force recreates the href even if the url and
                        uid information contained in the imported
                        transportable module does not match the information
                        for the target module specified with the -from
                        option.
```

-from

```
 -from <OldServerURL>  Specifies the URL of the old 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

                       Note: This option is mutually exclusive with -parents.
```

-parents

```
 -parents <TCLlist>    Specify the list of urls identifying parent
                       modules to update. By default, the list of
                       parents is retrieved from the whereused
                       information included with the module when the
                       module is imported.

                       Note: This option is mutually exclusive with
```

```
                        -from.
```

RETURN VALUE

SEE ALSO

```
mvmod, addbackref, addhref, rmhref, edithrefs
```

EXAMPLES

```
This example shows how to use reconnectmod after moving a module.
dss>  reconnectmod sync://serv2.ABCo.com:2647/Modules/ChipDesign/Chip-NX2
 Beginning module reconnect ...

 The following hierarchical reconnection will be made to each parent
 module:
    From: sync://serv1.ABCo.com:2647/Modules/Chips/chip-nx1
    To:   sync://serv2.ABCo.com:2647/Modules/ChipDesign/Chip-NX2

sync://serv1.ABCo.com:2647/Modules/Components/ROM : Updating
hierarchical references ...
sync://serv1.ABCo.com:2647/Modules/Components/ROM : Updating
reconnect history ...
sync://serv3.ABCo.com:2647/Modules/Components/CPU : Updating
hierarchical references ...
sync://serv3.ABCo.com:2647/Modules/Components/CPU : Updating
reconnect history ...

sync://serv2.ABCo.com:2647/Modules/ChipDesign/Chip-NX2 : Adding back
references ...
sync://serv1.ABCo.com:2647/Modules/Chips/chip-nx1 : Removing back
references ...
{Objects succeeded (4)} {}
```

rmhref

rmhref Command

NAME

```
rmhref              - Removes a hierarchical reference between modules
```

DESCRIPTION

```
This command removes hierarchical references (connections) from an
```

```
upper-level module to a module, external module, legacy module,
DesignSync vault, or IP Gear Deliverable.

The hierarchical reference is removed by creating a new module version
that does not contain the reference(s).

Note: You can remove existing hrefs in a module snapshot using the
      rmhref command. This creates a new module version on the
      snapshot branch.

The rmhref command does not automatically update individual
workspaces.  It only updates the module on the server.  After the
reference has been removed, all users using the module should
repopulate their workspaces to update the module. To identify
hierarchical references removed from the server, run the showstatus
command.

An href can only be removed when the following conditions are met:
o The href being removed must already exist.
o The fromargument must be a current module version.
o The fromargument module must be the latest version on the module
  branch or the auto-merge must be enabled.
o The fromargument module branch cannot be locked by another user.
o The server(s) that hosts both the fromargument and the toargument
  must be available.  Note: The availability of the toargument is not
  explicitly checked by the command.

Note: Removing hierarchical references can affect mirrored data
      differently.  For more information see the mirror commands.


To perform a collection of hierarchical reference updates, including
adding, removing or changing existing hrefs, see the edithrefs
command.

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.
```

SYNOPSIS

```
rmhref <fromargument> <toargument> [<toargument>..]
```

FROMARGUMENTS

- Server Module Version
- Workspace Module

Specifies the URL of the upper-level module version from which you want to remove the connection.

Note: If you are specifying the non-latest version, your system must have auto-merge enabled.

Specify one of the following arguments:

Server Module Version

<server module version>	Specify the URL as follows: 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. You may use this format to specify a module, module version or module branch. Note: If you specify a module, the remove is performed on the trunk branch.

Workspace Module

<workspace module>	Specify the workspace module. The module must be loaded in your workspace to remove an href and use a dynamic selector. If the version in the workspace is not the same as the current server version, you must have auto-merge enabled as the default checkin setup in order to remove the href. For more information on selecting auto-merge as the default checkin option, see The ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide: Site Options. Note: If the workspace uses as a static selectors, modifications, such as removing an href cannot be checked in. You must update the workspace to use a dynamic selector or remove the href from the server version.

TOARGUMENTS

- Hierarchical Reference Name

Hierarchical Reference Name

<href name>	Specifies the name of the href or TCL glob pattern for multiple hrefs to be removed.

```
Do not use a URL.  Use the name
value specified with the -name option in the
addhref command that created the reference.
To find the name of the href you want to
remove, you can run the showhrefs command on
the module.

Note: The rmhref command accepts multiple
toarguments to remove. The command then
creates a new module version with all of the
specified hrefs removed.
```

RETURN VALUE

```
If the rmhref command is successful, DesignSync returns an empty
string ("").  If the command fails, DesignSync returns an error
explaining the failure.
```

SEE ALSO

```
addhref, edithrefs, populate, showhrefs, showstatus, mirror,
command defaults
,
```

EXAMPLES

- Example of Removing an Href from a Module Workspace
- Example of Removing a Hierarchical Reference from a Server Module Version

```
The following examples use the hierarchy created in the addhref
example to show removing a hierarchical reference.
Note: you can only remove one href at a time. You can remove the
hrefs in any order.

Chip           <= uses the default configuration
  ALU          <= uses the default configuration
  CPU          <= uses the Gold branch version
    ROM        <= uses the Gold version
```

Example of Removing an Href from a Module Workspace

```
This example removes the href between the Chip module and the ALU
module, using the Chip%0 workspace instance to specify the top-level
module.

The following commands shows the module hierarchy before the href
removal.
```

```
dss> showhrefs Chip%0

 Beginning showhrefs operation ...

 Showing hrefs of module /home/tmarci2/MyModules/chip/Chip%0 ...

/home/rsmith/MyModules/chip/Chip%0: Workspace version - 1.8
/home/rsmith/MyModules/chip/Chip%0: Href mode - normal

Name  Url                                        Selector    Version
     Type         Relative Path
-----------------------------------------------------------------
ALU   sync://srvr2.ABCo.com:2647/Modules/ALU  Trunk:      1.3
     Module       ALU
CPU   sync://srvr2.ABCo.com:2647/Modules/CPU  Gold:       1.3
     Module       CPU

Finished showhrefs operation.

The following command removes the hierarchical reference from the
workspace and the server.

dss> rmhref Chip%0 ALU

 Beginning rmhref operation ...

 sync://srvr2.ABCo.com:2647/Modules/Chip: Created new module version 1.9.

 Finished rmhref operation.

After populate is run to update the workspace with the new module
version, you can run the showhrefs command to shows that the href
was removed.

dss> showhrefs Chip%0

 Beginning showhrefs operation ...

 Showing hrefs of module /home/rsmith/MyModules/chip/Chip%0 ...

 /home/tmarci2/MyModules/chip/Chip%0: Workspace version - 1.9
 /home/tmarci2/MyModules/chip/Chip%0: Href mode - normal

 Name  Url                                        Selector    Version
     Type    Relative Path
 -----------------------------------------------------------------
 CPU   sync://srvr2.ABCo.com:2647/Modules/CPU  Gold:      1.3
     Module  CPU

 Finished showhrefs operation.
```

Example of Removing a Hierarchical Reference from a Server Module Version

```
This example removes the href between the Chip module and the CPU
```

```
module, using the url to specify the top-level module.
Note: Because no version or branch is specified, the href is removed
from the latest version on the Trunk branch.

dss> rmhref sync://srvr2.ABCo.com:2647/Modules/Chip CPU

 Beginning rmhref operation ...

 sync://srvr2.ABCo.com:2647/Modules/Chip: Created new module version 1.6.

 Finished rmhref operation.

An showhrefs command on the module shows that the href was
removed and there are no hierarchical references left in the module.

dss> showhref sync://srv2.ABCo.com:2647/Modules/Chip

 Beginning showhrefs operation ...

 sync://srv2.ABCo.com:2647/Modules/Chip: Module has no hierarchical
 references.

 Finished showhrefs operation.
```

showhrefs

showhrefs Command

NAME

```
showhrefs           - Displays the hierarchy of a module
```

DESCRIPTION

- External Module Support
- Understanding the Output

```
This command displays the hierarchical references for a module or a
legacy module configuration.  When run recursively, this command
also displays the hierarchical references for all modules and legacy
module configurations in the hierarchy.

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.
```

External Module Support

DesignSync supports showing the hierarchical reference status of an external module to determine state of the hierarchical references. After an external module has been populated, the showhref command can be available to query the status of the external module hierarchical reference and return the results.

Hierarchical reference conflicts are reported in the final list output when all of the following conditions are true;
- o conflict property is set to yes during command processing
- o conflicts option is provided

These properties are not, however; returned in the final list output or included in the conflict summary.

Over-ridden hierarchical references can also be reported in the final output when the -overridden option is used with the -report verbose.

For information on populating an external module, see the populate command. For information on configuring showstatus for external modules, see the DesignSync Administrator's Guide.

Understanding the Output

The output of the showhrefs 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 showhrefs command, by default, displays the following information:

Column Titles	Property Names	Description
Name	name	The name of the href. For example, CPU.
URL	url	The URL of the referenced module, external module, legacy module configuration, DesignSync vault, or IP Gear deliverable. The URL for all references except external modules, includes host, port, and vaultPath. The external modules URL identifies the reference as an external module href and contains the <external-type> and the <external-data> strings.
Selector	selector	The selector for the href as supplied to the add href command. This varies depending on the type of href. o modules - the selector list used to identify the referenced module version.

		o legacy modules - the name of the referenced legacy module configuration. o DesignSync vaults - the selector list used to identify the referenced vault versions. For all other object types, the selector field is empty.
Version	version	The version of the referenced module the selector resolved at the time the href was created. If the href is not a module, the version field is empty.
Type	type	The type of object referenced. o Module - href to a module. o Alias - href to a legacy module alias. o Branch - href to a legacy module branch configuration. o External - href to an external module. o Release - href to a legacy module release configuration.. o Selector - href to a legacy module selector configuration. o Vault - href to a DesignSync vault. o Deliverable - href to an IP Gear deliverable. o Unknown - indicates that the object type could not be determined at the time the href was created.
	conflict	When the -conflict option is specified, the report reports conflicting hierarchical references. In text mode, a *CONFLICT* identifier is appended to the lines showing the conflicting URLs. In list mode, the command uses the conflict key. The value of the key is "yes," if the href is in conflict, or "no," if the href is not in conflict. In text mode, a conflict line is written beneath the table for each href in conflict during processing and, at the end of the output, a summary table shows all the conflicting hrefs. When populating a workspace module, with a V6R2015x client or later, showhrefs marks the conflicts at fetch time. And this option displays the conflicting references along with information about which modules are fetched into the workspace during populate operations. When populating a server-side module, conflict detection occurs on the fly while processing the hierarchy on all

		associated servers. The first conflicting href that is processed may not be known until another conflicting href is found and therefore the first conflicting href may not be marked as such. All subsequent conflicting hrefs will be marked with a line indicating the href is in-conflict with another href. The first conflicting href may not show as in conflict in the initial href report, but does in the summary table.
Relative Path	relpath	The relative path from the base directory of the upper-level module (Parent) to the base directory of the submodule. This path is used by the populate command when you fetch the module into your work area.
	instance_name	The instance identifier for the workspace module version. This is not applicable if the specified argument is a server module.
	basedir	The workspace base directory of the module. This is not applicable if the specified argument is a server module.

If you run the showhrefs command with '-report brief' it displays:

Column Titles	Property Names	Description
Name	name	The module name of the top level module and the href name for all referenced submodules, legacy module configurations, IP Gear deliverables, and DesignSync vaults.
URL/Base Directory	url/ basedir	Path to the module. If the command is performed on the server, it includes the full sync URL for the module, including selector information. If the command is performed on the workspace, it includes the full directory path. For external modules, it always displays the URL as: sync:///ExternalModule/<external-type>/<external-data>
	conflict	When the -conflict option is specified, the report reports conflicting hierarchical references. In text mode, a *CONFLICT* identifier is appended to the lines showing the conflicting URLs. In list mode, the command uses the conflict key. The value of the key is "yes," if the href is in conflict, or "no," if the href is not in conflict. The first

```
                              conflicting href may not show as in
                              conflict. For more information, see
                              the "conflict" description above in the
                              report -normal output section.

Note: The -brief output indents the module name to graphically
represent the  hierarchical.

If you run the report in -verbose mode with the -overridden option,
it adds the following column:

 Column       Property
 Titles       Names          Description
 ------       -----          ------------
 Overridden   overridden     Whether this particular href is
                             overridden by a higher level href.
                             yes - this URL is ignored by system
                                   because it is overridden by a
                                   higher level href.
                             no - this URL is active; not overridden.

If you specify the -overridden option, all submodules, even the
overridden submodules, display with their own showhrefs table.

As showhrefs recurses a hierarchy, it counts any errors and displays
suitable error messages. At the end of the operation, showhrefs
displays an error message which contains the number of errors that
occurred.

Using the -format list option formats the output into a Tcl string
that can be processed in scripts, however the order and the property
names differ slightly from the -format text (default) option.  For
more information, see Example 3 which shows using the -format list
option with report -normal and Example 4 which shows report -brief.
```

SYNOPSIS

```
showhrefs [-[no]conflicts] [-format text|list]
          [-hrefmode {dynamic | static | normal}]
          [no]overridden] [-[no]recursive]
          [-report {brief | normal | verbose | script}]
          [-[no]stopatconflict] [-xtras <xtras>] <argument>
```

ARGUMENTS

- Server Module
- Workspace Module
- External Module Instance
- Server Legacy Module
- Workspace Legacy Module

Server Module

<server module> Shows hierarchical references for a module version. Specify the URL as follows:
sync[s]://<host>[:<port>]/Modules/[<category>...]<module>[;<selector>]

where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, <category> identifies the path to the module, <module> is the name of the module, and <selector> identifies a particular branch or version. You may use this format to specify a module, module branch, or module version. The default branch is "Trunk." The default version is "Latest.

Workspace Module

<workspace module> Shows hierarchical references for a workspace module. You can specify the workspace module by using the module name, if it's unique within the workspace (For example: Chip), or the workspace module instance name. (For example: Chip%0).

External Module Instance

<external_mod> Specifies the external module instance. The external module must be populated into the workspace.

Server Legacy Module

<server module> Shows hierarchical references for a legacy module. Specify the URL as follows:
sync[s]://<host>[:<port>]/<vaultPath>[;<selector>]
where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, <vaultPath> identifies the module, and <selector> identifies a legacy module configuration. The default branch is "Trunk." The default version is "Latest.

Note: If no selector information is provided for legacy modules, DesignSync uses the default configuration.

Workspace Legacy Module

<workspace module> Shows hierarchical references for a workspace

```
                    module. You can specify the legacy module using
                    the path of the workspace. (Legacy Modules Only)
```

OPTIONS

- -[no]conflict
- -format
- -hrefmode
- overridden_option
- -[no]recursive
- -report
- -[no]stopatconflict
- -xtras

-[no]conflict

```
 -[no]conflict          Determines whether to show conflicting
                        hierarchical references; multiple references
                        to different versions of the same module; in
                        the output.

                        -[no]conflict does not show module
                        conflicts for server module arguments.  For
                        workspace module argument, if conflicting
                        hrefs are populated into the workspace, the
                        conflict is reported. (Default)

                        -conflicts shows, for each module with a
                         conflicting hierarchical reference,
                         information about which submodule version was
                         or was not populated because of a
                         conflict.

                        Notes:
                        o If the -noconflicts option is specified with the
                          -stopatconflicts option, the -stopatconflicts
                          option is silently ignored.

                        o This option is only applicable to modern
                          modules. If specified for a legacy module
                          argument,   the option is silently ignored.
```

-format

```
 -format <type>         Determines the format of the output.
                        Valid values are:
                        o list -  Displays a list with the following
                          format:
                               {
```

```
                name <name>

               }
```

o text - Display a text table with headers and columns. (Default)

-hrefmode

-hrefmode

Indicates how the hierarchy is traversed by the command.

Note: This option is ignored for workspace modules which are always traversed the way they were loaded into the workspace.

Valid values are:

o dynamic - Resolves the href selector to determine the referenced module version to expand.

o static - Expands the module version to which the href selector resolved when the href was created.

o normal - Resolves the href selector to determine the referenced module version to expand. If, while traversing the hierarchy, the showhrefs command reaches a static selector, for example a version tag or numeric version ID, the hrefmode switches to static for the remaining sub-hierarchy of the referenced module.(Default) This option respects 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.

-[no]overridden

Determines whether to display information about overridden hrefs.

-nooverridden - Ignores overridden hrefs and does not include them into the href traversal. (Default)

-overridden - Traverses the module hierarchy, including displaying overridden hrefs and showing a table for each href.

Note: In order to see overridden hrefs, you must specify BOTH -overridden AND -report

verbose mode.

-[no]recursive

-[no]recursive — Determines whether to display hierarchical references for the specified module, or the specified module and all submodules.

-norecursive displays hierarchical references for the specified module. (Default)

-recursive displays hierarchical references for the specified module and all submodules.

-report

-report — Specifies the information output. The information each option returns is discussed in detail in the "Understanding the Output" section above.

Valid values are:

- brief - Displays the Name of the hierarchical reference and the module path to the reference. If the -showconflicts option is selected, a *CONFLICT* identifier showing the conflicting hrefs. The brief mode does not provide information about which href would be populated.
- normal - Displays a list of hierarchical references and their properties. (Default)
- verbose - Displays the information available with normal.
- script - Returns a Tcl list of config_name/property_list pairs.

-[no]stopatconflict

-[no]stopatconflict — Specifies whether the command should continue processing a hierarchy recursively after reaching a conflict in the hierarchy or stop.

-nostopatconflict continues recursing through the directory hierarchy regardless of how many conflicts are revealed. (Default)

-stopatconflict stops processing recursively

```
                          through the directory hierarchy when the
                          first conflict within the hierarchy is
                          exposed.

                        Notes:
                          o When processing a workspace module, the
                            traversal only includes recursion through
                            the fetched submodules and it always stopped
                            at conflicting hrefs not in the workspace.

                          o If the -noconflict option is selected,
                            with the -stopatconflict option, the
                            -stopatconflict option is silently
                            ignored.
```

-xtras

```
 -xtras <xtras>           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

```
 If you run the showhrefs 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 ("").

 For a description of the output, see the "Understanding the Output"
 section.
```

SEE ALSO

```
 addhref, edithrefs, populate, showmcache, showstatus, rmhref,
 command defaults
,
```

EXAMPLES

- Example of Displaying the Hierarchical References on the Server
- Example of Showing Hrefs on the Server Vault in List Format
- Example of Displaying the Hierarchical References in a Workspace
- Example of Showing Hrefs on the Workspace in List Format
- Example of Showing Hrefs in Brief Report mode and List Format
- Example Showing Overridden Hrefs in the Workspace in Text Format

- Example Showing Overridden Hrefs in List Format
- Example Showing Conflicting Hierarchical References in the Workspace

```
The following examples all use this hierarchy, which is the example
created with the addhref command:

Chip           <= uses the default configuration
  Cpu          <= uses the Gold version
    ROM        <= uses the Gold version
```

Example of Displaying the Hierarchical References on the Server

```
This example displays the hierarchical references for the Chip
module in text mode, which is specified by the sync URL.

dss> showhrefs -recursive sync://srvr2.ABCo.com:2647/Modules/Chip

 Beginning showhrefs operation ...

 Showing hrefs of module sync://srvr2.ABCo.com:2647/Modules/Chip (1.7) ...
 sync://srvr2.ABCo.com:2647/Modules/Chip: Href mode is normal.

 Name  Url                                    Version  Type    Relative Path
 ------------------------------------------------------------------------
 CPU sync://srvr2.ABCo.com:2647/Modules/CPU  Gold     Module  CPU

 ========================================================================

 Showing hrefs of module sync://srvr2.ABCo.com:2647/Modules/CPU;Gold (1.3)
...

 sync://srvr2.ABCo.com:2647/Modules/CPU;Gold: Href mode is static.

 Name  Url                                Version  Type    Relative Path
 ---------------------------------------------------------------------
 ROM   sync://srvr2.ABCo.com:2647/Modules/ROM  1.4      Module  /ROM

 sync://srvr2.ABCo.com:2647/Modules/ROM;Trunk:: Module has no
 hierarchical references.

Finished showhrefs operation.
```

Example of Showing Hrefs on the Server Vault in List Format

```
This example displays hierarchical references specifying a server
module, sync://srvr2.ABCo.com/Modules/Chip,  with report -normal
output. This example uses the hierarchy described at the beginning of
the examples section.

dss> showhrefs -format list -recursive \
```

```
 sync://srvr2.ABCo.com:2647/Modules/Chip

{relpath CPU name CPU type Module version Gold url  \
 sync://srvr2.ABCo.com:2647/Modules/CPU hrefs  {{relpath /ROM name \
 ROM type Module version 1.4 url \
 sync://srvr2.ABCo.com:2647/Modules/ROM}}}

Note: The hierarchy of each submodule is contained within the hrefs
      property.
```

Example of Displaying the Hierarchical References in a Workspace

```
This example displays the hierarchical references for the Chip module
in text mode; specified by the module workspace instance name.

 dss> showhrefs -recursive Chip%0

 Beginning showhrefs operation ...
 Showing hrefs of module /home/rsmith/Modules/chip/Chip%0 ...

 /home/rsmith/MyModules/chip/Chip%0: Workspace version - 1.7
 /home/rsmith/MyModules/chip/Chip%0: Href mode - normal

 Name  Url                                Version  Type    Relative Path
 ------------------------------------------------------------------------
 CPU   sync://srvr2.ABCo.com:2647/Modules/CPU  1.2      Module  CPU

 ========================================================================

Showing hrefs of module /home/rsmith/MyModules/chip/CPU/CPU%1 ...

Name  Url                                Version  Type    Relative Path
------------------------------------------------------------------------
ROM   sync://srvr2.ABCo.com:2647/Modules/ROM  1.4      Module  /ROM

/home/rsmith/MyModules/chip/ROM/ROM%1: Workspace version - 1.2
/home/rsmith/MyModules/chip/CPU/ROM/ROM%1: Module has no hierarchical
references.

Finished showhrefs operation.
```

Example of Showing Hrefs on the Workspace in List Format

```
This example displays hierarchical references specifying a workspace
module, Chip%0,  with report -normal output. This example uses the
hierarchy described at the beginning of the examples section.

dss> showhrefs -format list  -recursive Chip%0

{hrefs {{relpath /ROM name ROM instance_name ROM%1 version 1.4 type \
Module basedir /home/rsmith/MyModules/chip/CPU/ROM url \
sync://srvr2.ABCo.com:2647/Modules/ROM}} relpath CPU name CPU \
```

```
 instance_name CPU%1 version 1.2 type Module basedir \
 /home/rsmith/MyModules/chip/CPU url \
 sync://srvr2.ABCo.com:2647/Modules/CPU}
```

Example of Showing Hrefs in Brief Report mode and List Format

```
 This example shows the brief version of the report on the hrefs shown
 in Example 1 formatted for Tcl processing.

 dss> showhrefs -format list -report brief -recursive Chip%0

  name Chip url sync://srvr2.ABCo.com:2647/Modules/Chip hrefs {{name \
  CPU url {sync://srvr2.ABCo.com:2647/Modules/CPU;Gold} hrefs {{name \
  ROM url {sync://srvr2.ABCo.com:2647/Modules/ROM;1.4}}}}}
```

Example Showing Overridden Hrefs in the Workspace in Text Format

```
 This example shows overridden hierarchical references in the
 workspace.  It uses the following hierarchical structure:

 CHIP           <= uses the default configuration
   CPU          <= uses the Trunk:Latest version
     LIB        <= uses the Gold version
   ROM        <= uses the Trunk:Latest version
     LIB      <= uses the Trunk:Latest version (static version 1.2)

 dss> showhrefs -overridden -report verbose CHIP%0

 Beginning showhrefs operation ...

 Showing hrefs of module c:/Workspaces/chip/CHIP%0 ...

 c:/Workspaces/chip/CHIP%0: Workspace version - 1.4
 c:/Workspaces/chip/CHIP%0: Href mode - normal

 Name  Url                                                    Selector
 Static Version  Type    Relative Path  Overridden
 ------------------------------------------------------------------
 CPU   sync://serv1.ABCo.com:2647/Modules/ChipDesign/CPU        Trunk:    1.2
Module  ../cpu         no
 LIB   sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB  Gold      1.2
Module  ../lib         no
 ROM   sync://serv1.ABCo.com:2647/Modules/ChipDesign/ROM        Trunk:    1.2
Module  ../rom         no

  ========================================================================

 Showing hrefs of module c:/Workspaces/cpu/CPU%1 ...

 c:/Workspaces/cpu/CPU%1: Workspace version - 1.3
 c:/Workspaces/cpu/CPU%1: Href mode - normal
```

```
Name  Url                                                    Selector
Static Version  Type    Relative Path  Overridden
-------------------------------------------------------------------
LIB   sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB Gold
1.2             Module  ../lib         yes

c:/Workspaces/lib/LIB%0: Workspace version - 1.2
c:/Workspaces/lib/LIB%0: Module has no hierarchical references.

 ============================================================================

Showing hrefs of module c:/Workspaces/rom/ROM%1 ...

c:/Workspaces/rom/ROM%1: Workspace version - 1.3
c:/Workspaces/rom/ROM%1: Href mode - normal

Name  Url                                                    Selector
Static Version  Type    Relative Path  Overridden
-------------------------------------------------------------------------
LIB   sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB Trunk:
1.2            Module  ../lib         yes


Finished showhrefs operation.
```

Example Showing Overridden Hrefs in List Format

```
 This example shows overridden hierarchical references in the
workspace.  It uses the same hierarchical structure as the previous
example.

dss> showhrefs -overridden -report verbose -format list CHIP%0
{hrefs {{relpath ../LIB resolved_selector 1.2 name LIB selector Gold
type Module overridden no version 1.2 url
sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB}} relpath ../COM
resolved_selector 1.3 name COM selector Trunk: type Module overridden
no version 1.2 url sync://serv1.ABCo.com:2647/Modules/ChipDesign/COM}
{relpath ../LIB resolved_selector 1.2 name LIB selector Gold type
Module overridden no version 1.2 url
sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB} {hrefs {{relpath
../LIB resolved_selector 1.2 name LIB selector Trunk: type Module
overridden no version 1.2 url
sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB}} relpath ../ROM
resolved_selector 1.3 name ROM selector Trunk: type Module overridden
no version 1.2 url sync://serv1.ABCo.com:2647/Modules/ChipDesign/ROM} #
```

Example Showing Conflicting Hierarchical References in the Workspace

```
This example shows hierarchical conflicts in the workspace in normal
reporting mode.
```

```
dss> showhrefs -conflict TOP%0

 Beginning showhrefs operation ...

 Showing hrefs of module c:/chip/top/TOP%0 ...

 c:/chip/top/TOP%0: Workspace version - 1.4 : Selector - Trunk:
 c:/chip/top/TOP%0: Href mode - normal

 Name  Url                                                    Selector
 Static Version  Type    Relative Path
 ------------------------------------------------------------------------
 COM   sync://serv1.ABCo.com:2647/Modules/Chip-P21z/COM        Trunk:
 1.2             Module  ../COM
 LIB   sync://serv1.ABCo.com:2647/Modules/Chip-P21z/Tools/LIB  Gold
 1.2             Module  ../LIB
 ROM   sync://serv1.ABCo.com:2647/Modules/Chip-P21z/ROM        Trunk:
 1.2             Module  ../ROM

 LIB: Not present in workspace due to hierarchical conflict.
 Finished showhrefs operation.
```

hcm showhrefs Command

NAME

```
hcm showhrefs       - Displays the hierarchy of a module
                      configuration (legacy)
```

DESCRIPTION

- Understanding the Output

```
This command displays the hierarchical references for a module
configuration. When run recursively, this command displays the
hierarchical references for the upper-level module configuration and
all referenced submodule configurations, IP Gear deliverables, or
DesignSync vaults.

This command should not be used for non-legacy modules.  For
non-legacy modules, use the showhrefs command with no hcm prefix. The
command syntax for hcm showhrefs is different than the showhref
command syntax.

Note:
- This command obeys DesignSync's BrowseServer access control; if you
  do not have permission to browse a server, this command returns an
  error.

For this command to be successful, the following must be true:
- If you specify -target, the server on which the configuration resides
```

must be available and the configuration specified must exist.
- If you specify -path, the path specified must exist.
- If you do not specify -target the directory specified by -path must contain an HCM configuration. If either -path is not specified or it does not contain a configuration, a parent directory in the file system hierarchy must contain an HCM configuration. The hcm showhrefs command looks first for a configuration in the directory identified by -path. If that directory does not contain a configuration, the hcm showhrefs command searches up the file system hierarchy until it finds a parent directory that does contain a configuration or it reaches the root of the file system. If it reaches the root of the file system without finding a configuration, the command fails.

Understanding the Output

By default, or if you run the hcm showhrefs command with the '-report normal' option, the command displays the status of the hierarchical references for the configuration in a user-friendly format.

The hcm showhrefs command, by default, displays the following information:

o Reference	The leaf of the vault path. For example, if you specified sync://host2:port2/Projects/ALU for -totarget, the value for Reference would be ALU.
o URL	The complete URL of an object including host, port, vaultPath, and optionally '@<config>'. If the object is a configuration and '@<config>' is absent, the default configuration is assumed.
o Relative Path	The relative path from the base directory of the upper-level module configuration (Parent) to the submodule configuration (Target). This path is used by the hcm get command when you fetch the module into your work area.

If you run the hcm showhrefs command with '-report brief' it displays the following information:

o Target	The upper-level module configuration (first line of output) and the submodule configurations, DesignSync vaults, or IP Gear deliverables, to which the upper-level module is connected. If '@' is absent, the default configuration is assumed.
o URL	The complete URL of an object including host, port, vaultPath, and optionally '@<config>'. If the object is a configuration and '@<config>' is absent, the default configuration is assumed. URL information is displayed if you run the hcm showhrefs command with the -target option.
o Path	The local directory in which the module or or IP Gear deliverable reside. Path information is displayed unless you specify the -target option. Note: A path marked as '(mcached)' indicates that the local directory is a link to

```
                    a release in the module cache.

If you run the hcm showhrefs command with '-report script', it returns
a Tcl list in the following form:

   {relpath <relative_path1>  target <full_URL_object1>}
    {relpath <relative_path2>  target <full_URL_object2>}
    ...

Notes:
o The order of relpath and target might be reversed in your output.
o The output of hcm showhrefs -report script does not include
  information about module cache links.

If you run the hcm showhrefs command with '-report command', it
displays to the screen a list of hierarchical references in the
following form:

   -fromtarget <full_URL_uppermod@config>
     -totarget <full_URL_submod> -relpath <relative_path>

   - fromtarget is the URL of the upper-level module configuration for
     which the hierarchical references are shown. If the upper-level
     module configuration is a default configuration, the '@config'
     will be absent.

   - totarget is the URL of an IP Gear deliverable, DesignSync vault,
     or a submodule configuration to which the upper-level module
     configuration connects. If the totarget is a configuration, the
     URL will include '@config' for branch or selector configurations,
     aliases, and releases. If the totarget is a default configuration,
     '@config' will not be present.

   - relpath is the relative path from the base directory of the upper-
     level module configuration to the submodule configuration, IP Gear
     deliverable, or DesignSync vault. This path is used by the hcm get
     command when you fetch the module into your work area.

Note:
- relpath can be empty. An empty path is specified if you do not want
  to fetch the contents of the submodule into your work area.
  This behavior may be desirable if you want to reference a ProjectSync
  project that tracks defects against a CAD tool.

As hcm showhrefs recurses a hierarchy, it counts any errors and
displays suitable error messages. At the end of the operation,
hcm showhrefs displays an error message which contains the number of
errors that occurred.
```

SYNOPSIS

```
hcm showhrefs  [-recursive]
               [-report {brief | normal |verbose | command | script}]
               [-target <configuration_url> | -path <path>]
```

OPTIONS

- -path
- -recursive
- -report
- -target

-path

-path <path>	Identifies the directory containing the configuration for which the hierarchical references will be shown. If neither -path or -target is specified, the hierarchical references for the default configuration associated with the current directory will be shown.

-recursive

-recursive	Displays the hierarchical references for the upper-level module configuration and all submodule configurations.

-report

-report	Indicates the format in which the output appears. Valid values are: o brief - Displays the output in a user-friendly format such that the hierarchy of submodules is visible. 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 - Displays to the screen a list of hierarchical references in a format that closely matches the usage of the hcm rmhref command. With this format you can easily cut and paste to remove a hierarchical reference. o script - Returns a Tcl list.

Note:
- If you run the hcm showhrefs command with the -path option and the '-report brief' option, the command does not detect errors with referenced objects. When you run hcm showhrefs with the -path option, the server is not contacted.

-target

-target <configuration_url>	Specifies the URL of a configuration for which the hierarchical references will be shown. o To specify the default configuration of the upper-level module, use the following syntax: 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. o To specify a configuration other than the default configuration, use the following syntax: sync[s]://<host>[:<port>]/<vaultPath>@<config> where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, and <vaultPath> identifies the module and <config> identifies the specific configuration. Note: - If you do not have access to browse the server on which the module is located (set via DesignSync's BrowseServer access control), the hcm showhrefs command fails.

RETURN VALUE

If you run the hcm showhrefs 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

hcm addhref, hcm get, hcm release, hcm showmcache, hcm showstatus
,

EXAMPLES

- Displaying the hrefs Associated with a Configuration
- Understanding When Hierarchical References Are Updated

Displaying the hrefs Associated with a Configuration

```
The following examples assume the following hierarchy in your work
area:
Chip              <= uses the default configuration
   Cpu            <= uses the default configuration
   Alu@R1         <= R1 is already released
     Dpath@R2     <= R2 is already released

The "Creating, Fetching, and Releasing Module Hierarchies" example in
the hcm Example topic includes the steps for creating the modules,
configurations, and hierarchical references that are presented in
this example.

Displaying the Hierarchical References on the Server
----------------------------------------------------
This example displays to the screen the hierarchical references for
the default configuration of the Chip module.

dss> hcm showhrefs -target sync://chip.ABCo.com:2647/Projects/Chip \
         -report command

This command displays the following output:
-fromtarget sync://chip.ABCo.com:2647/Projects/Chip
 -totarget sync://alu.ABCo.com:2647/Projects/Alu@R1 -relpath Alu
-fromtarget sync://chip.ABCo.com:2647/Projects/Chip
 -totarget sync://cpu.ABCo.com:2647/Projects/Cpu -relpath Cpu

The output includes only the Cpu and Alu hierarchical references;
in this example the hcm showhrefs command was run without specifying
the -recursive option.

To show the entire directory structure, run the hcm showhrefs
command with the -recursive option as follows:

dss> hcm showhrefs -recursive \
                   -target sync://chip.ABCo.com:2647/Projects/Chip

This command displays the following output:

Target:         sync://chip.ABCo.com:2647/Projects/Chip

REFERENCE  URL                                          RELATIVE PATH
-------------------------------------------------------------------
Alu@R1     sync://alu.ABCo.com:2647/Projects/Alu@R1 Alu
Cpu        sync://cpu.ABCo.com:2647/Projects/Cpu    Cpu


===================================================================
```

```
Target:         sync://alu.ABCo.com:2647/Projects/Alu@R1
Parent:         sync://chip.ABCo.com:2647/Projects/Chip

REFERENCE  URL                                              RELATIVE PATH
-------------------------------------------------------------------------
Dpath@R2   sync://dpath.ABCo.com:2647/Projects/Dpath@R2 Dpath

Note: Target has no hierarchical references.
      Target: sync://dpath.ABCo.com:2647/Projects/Dpath@R2
Note: Target has no hierarchical references.
      Target: sync://cpu.ABCo.com:2647/Projects/Cpu
```

Alternately, to show the entire directory structure visually, run the hcm showhrefs command with both -recursive and the '-report brief' options as follows:

```
dss> hcm showhrefs -recursive -report brief \
                   -target sync://chip.ABCo.com:2647/Projects/Chip
```

This command displays the following output:

```
Hierarchy for sync://chip.ABCo.com:2647/Projects/Chip

TARGET            URL
-----------------------------------------------------------
Chip              sync://chip.ABCo.com:2647/Projects/Chip
    Alu@R1        sync://alu.ABCo.com:2647/Projects/Alu@R1
        Dpath@R2  sync://Dpath.ABCo.com:2647/Projects/Dpath@R2
    Cpu           sync://cpu.ABCo.com:2647/Projects/Cpu
```

Displaying the Hierarchical References in the Work Area

The previous example displayed the hierarchical references for the default configuration of the Chip module that resides on the server. To display the hierarchical references for the default configuration of the Chip module that was fetched into the work area substitute the -path option for the -target option as follows:

```
dss> hcm showhrefs -recursive -report brief \
                   -path /home/jsmith/Designs/Chip
```

This command displays the following output:

```
Hierarchy for sync://chip.ABCo.com:2647/Projects/Chip

TARGET            PATH
-----------------------------------------------------
Chip              /home/jsmith/Designs/Chip
    Alu@R1        /home/jsmith/Designs/Chip/Alu
        Dpath@R2  /home/jsmith/Designs/Chip/Alu/Dpath
    Cpu           /home/jsmith/Designs/Chip/Cpu
```

Displaying Hierarchical References in the Work Area When It Contains Releases Fetched from the Module Cache

The previous example showed using hcm showhrefs with the -path option to display hierarchical references for the default

```
 configuration of the Chip module that was fetched into the work
 area directory /home/jsmith/Designs/Chip.

 This example shows the use of the hcm showhrefs -path command when:
 o The Alu@R1 release resides in the module cache, /home/ChipDev/cache.
 o The Alu@R1 release in the module cache contains a hierarchical
   reference to the Dpath@R2 configuration, which is also a release.
 o The default configuration of the Chip module was fetched to the
   work area by using hcm get in module cache link mode. (For
   information, see the hcm get command.)
 o As a result of the fetch of the Chip module, the work area
   contains a link to the Alu@R1 release in the module cache.

 dss> hcm showhrefs -recursive -report brief \
                    -path /home/jsmith/Designs/Chip

 This command displays the following output:

 Hierarchy for sync://chip.ABCo.com:2647/Projects/Chip

 TARGET            PATH
 ------------------------------------------------------
 Chip              /home/jsmith/Designs/Chip
     Alu@R1        /home/jsmith/Designs/Chip/Alu (mcached)
         Dpath@R2  /home/ChipDev/cache/Alu/Dpath
     Cpu           /home/jsmith/Designs/Chip/Cpu

Notes:
o In this example, '(mcached)' indicates that the
  /home/jsmith/Designs/Chip/Alu directory is a link to the Alu@R1
  release in the module cache.
o Because Dpath@R2 is a submodule of the Alu@R1 release in the
  module cache, the path for Dpath@R2 shows its location in the
  module cache, not the work area.
```

Understanding When Hierarchical References Are Updated

```
 This example assumes the following hierarchy in your work area:
 Top               <= uses the default configuration
    Mem@DEV        <= DEV is a branch configuration
    IO@TEST        <= TEST is a selector configuration

 The "Creating and Removing Module Configurations" example in the hcm
 Example topic includes the steps for creating the modules,
 configurations, and hierarchical references that are presented in the
 following examples.

 This example displays the hierarchical references associated with
 the default configuration of the Top module and their status. It is
 assumed that the default configuration of Top has previously been
 fetched recursively to the work area directory,
 /home/jsmith/Designs/Top, using the -recursive option to the hcm get
 command.
```

```
After the default configuration of Top was fetched, the hierarchical
reference between Top and IO@TEST was removed on the server.

dss> hcm rmhref \
            -fromtarget sync://srvr1.ABCo.com:2647/Projects/Top \
            -totarget sync://srvr1.ABCo.com:2647/Projects/IO@TEST

It is further assumed that after the hierarchical reference between
Top and IO@TEST was removed on the server the work area was not
fetched again.

To display the hierarchical references for the default Top
configuration, run the following command:

dss> hcm showhrefs -recursive -path /home/jsmith/Designs/Top

Note:
-This command displays the hierarchical references that are in your
 work area. Notice that even though the reference between the
 default configuration of Top and IO@TEST was removed on the server,
 it still exists in your work area.
-The specified path can be relative.

This command displays this following output:

Target:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top

REFERENCE  URL                                              RELATIVE PATH
-------------------------------------------------------------------------
IO@TEST    sync://srvr1.ABCo.com:2647/Projects/IO@TEST IO
Mem@DEV    sync://srvr1.ABCo.com:2647/Projects/Mem@DEV Mem

Note: Target has no hierarchical references.
      Target: sync://srvr1.ABCo.com:2647/Projects/IO@TEST
      Path:   /home/jsmith/Designs/Top/IO
Note: Target has no hierarchical references.
      Target: sync://srvr1.ABCo.com:2647/Projects/Mem@Dev
      Path:   /home/jsmith/Designs/Top/Mem

To determine if the hierarchical references for the default Top
configuration in the work area are current, run the hcm showstatus
command.
```

Informational

Whereused Information

whereused

whereused Command

NAME

```
whereused           - Traces the use of hierarchical references
```

DESCRIPTION

```
This command is being deprecated in favor of 'whereused module' and
'whereused vault'. In future releases, this command will behave as
other superset commands do, returning a list of of the available
subcommands.  A third command, "whereused member," has been added
to identify where a module member is used.

Tip: Update any scripts, processes, or proceedures that use or
recommend the whereused command to the appropriate whereused
subcommand.

This command identifies the module versions in which any of the
following targets (the "toargument" for the addhref command) are
used: sub-module version, legacy module configuration, or DesignSync
vault, identified by a selector. This allows users to trace the usage
of any desired target.

When a hierarchical reference is created, DesignSync creates a back
reference on the toargument indicating that it is referenced by the
fromargument module. This provides the basic mechanism for the
whereused functionality. If a back reference does not exist, for
example, if the hierarchical reference was created before whereused
was implemented,  you can add back references independently using the
addbackref command. The whereused command performs no validation on
specified legacy module and vault selectors. Verify that the selector
specified is correct before executing the command.

Notes:
  o When referencing legacy modules or DesignSync vaults using a
    configuration name or a selector as the specified version, the
    version must be an exact match for the selector used when the
    hierarchical reference was added.

  o When the hierarchical reference target (toargument) was specified
    with SSL protocol, the whereused command must also be specified
    with SSL.

This command is subject to access controls on the server. The
whereused command requires browse access to the objects in the
module hierarchy. See the ENOVIA Synchronicity Access Control Guide
for details.
```

SYNOPSIS

```
whereused [-format list|text] [-[no]recursive] [-report <mode>]
          [-showtags all|none|immutable|version]
          [-versions <selector>,...] <argument>
```

ARGUMENTS

- DesignSync Vault
- Server Module
- Legacy Module URL

DesignSync Vault

```
<DesignSync vault> URL of the referenced DesignSync vault.
                   Specify the URL as follows:
                   sync[s]://<host>[:<port>]<vaultpath>;[<selector>]
                   where <host> is the SyncServer on which the
                   object resides, <port> is the SyncServer
                   port number, <vaultpath> identifies the
                   DesignSync object, and <selector> is the optional
                   selector.

                   If a selector is not specified, the command
                   uses versions specified with the -versions option
                   or the default value of Trunk:Latest.

                   Notes:
                     o If you specify a selector in the argument, you
                       cannot specify additional selectors with the
                       -versions option. To locate more than one
                       selector at a time, use the -versions option.

                     o When the hierarchical reference target
                       (toargument) was specified with SSL protocol,
                       the whereused command must also be specified
                       with SSL.

                     o The vault selector is not checked for validity
                       before the command is executed. Verify that
                       the vault URL and selector is typed correctly
                       before executing the command.
```

Server Module

```
<server module>    URL of the referenced module.
                   Specify the URL as follows:
                   sync[s]://<host>[:<port>]/Modules/[<Category>...]
                    /<module>[;<selector>,...]
                   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 <selector> is the optional
                   selector.

                   If a selector is not specified, the command
                   uses versions specified with the -versions option
```

```
                    or the default value of Trunk:Latest.

                    Notes:
                      o If you specify a selector in the argument, you
                        cannot specify additional selectors with the
                        -versions option. To locate more than one
                        selector at a time, use the -versions option.

                      o When the hierarchical reference target
                        (toargument) was specified with SSL protocol,
                        the whereused command must also be specified
                        with SSL.
```

Legacy Module URL

```
 <Legacy module     URL of the referenced legacy module.
  URL>              Specify the URL as follows:
                    sync[s]://<host>[:<port>]/[Projects]/<vaultpath>;
                     [<selector>,...]
                    where <host> is the SyncServer on which the
                    module resides, <port> is the SyncServer
                    port number, vaultpath is the path to the
                    module, and <selector> is the optional selector,
                    or configuration name.

                    If a selector is not specified, the command
                    uses versions specified with the -versions option
                    or the default value of Trunk:Latest.

                    Notes:
                      o If you specify a selector in the argument, you
                        cannot specify additional selectors with the
                        -versions option. To locate more than one
                        selector at a time, use the -versions option.

                      o When the hierarchical reference target
                        (toargument) was specified with SSL protocol,
                        the whereused command must also be specified
                        with SSL.

                      o The vault selector is not checked for validity
                        before the command is executed. Verify that
                        the vault URL and selector is typed correctly
                        before executing the command.
```

OPTIONS

- -format
- -[no]recursive
- -report
- -showtags
- -versions

-format

```
-format list|text  Determines the format of the output.  For
                   details about the information returned
                   see the -report option.
                   Valid values are:

                   o text - Display text output formatted to show the
                   hierarchical reference tree. (Default)

                   o list - Displays a list with the following
                            format:
                             {
                               object <object>...whereused
                               <hierarchical_whereused_list>
                             }
```

-[no]recursive

```
-[no]recursive     Determines whether to show the locations in
                   which the version is explicitly referenced, or
                  show all modules in which the version is implicitly
                  or explicitly referenced. An explicit reference
                  exists when there is a direct reference link
                  between the module and the
                  target.  An implicit reference exists when the
                  module and target are not directly connected, but
                  within the module's hierarchy exists a reference
                  to the target. For example: if the Chip module
                  references the Gold version of the ALU module, and
                  the Gold version of the ALU module references the
                  Gold version of the ROM module, the Chip module
                  contains an implicit reference to the ROM module
                  and an explicit reference to the ALU module.

                   -norecursive expands the reference and returns a
                   list of modules that contain an explicit reference
                   to the referenced module.  (Default)

                   -recursive traverses the reference and returns a
                   list of all modules that explicitly and implicitly
                   reference the specified version.
```

-report

```
-report brief|     Determines what the information is returned in the
 normal|verbose    output of the whereused command.

                   Valid values are:
```

- brief - reports only the module names for the specified object ordered by specified version.

  When displayed in text format, the output is indented to show the reverse hierarchical reference path. When displayed in list format, the hierarchical reference depth is indicated with the depth property.

  Note: The -showtags option is ignored for this report mode.

- normal - reports the version of the module containing the reference to the specified object. The text format is indented to show hierarchical reference depth, and the list format uses the depth property. The version property in list mode contains the version number.

  Note: The version property may be a selector rather than a version number if the target is a DesignSync vault or legacy module.

- verbose - provides the information available in report mode -normal as well as the processing status of the command.

-showtags

-showtags all|none|immutable|version	Specifies whether tag information is displayed and optionally restricts the output to immutable tags or tagged versions.

- all - Displays all tags and all reference locations, including references that are not tagged. (Default)

- none - Displays all references locations, but does not display tag information

- immutable - Displays only reference locations tagged with an immutable tag and the name of the immutable tag.

  Note: Using the -showtags immutable option may not display all versions in which an immutable tag is used. The whereused command queries for all the whereused information but filters the display from the starting point until it reaches the last immutable tag in a reference tree.

- version - Displays any reference location that has a version tag and the name of the tag.

Note: The -showtags option is ignored for -report brief mode.

-versions

-versions <selector>	Comma separated set of versions to locate. The versions can be numerics or selectors, If the argument specified is a module, the selector resolves to the numeric value of the module version and is compared against the static href value. If the object is a DesignSync vault or legacy module, then the version selector (branch, version release, or alias selector) is validated by string comparison against the dynamic href value. Note: To specify a comma separated selector list, use the version extended naming format, not the -versions option.

RETURN VALUE

This command does not return a Tcl value in text mode. If the command is unable to run, DesignSync throws an error explaining the failure.

If whereused is unable to process a reference, you will see the reason for failure during command processing and the command adds the sync URL of the failed module to the wuFail list.

In list mode, the list of modules is returned in an array of name/value pairs. If a reference cannot be processed, the command adds an error property containing the reason for the failure and the module is added to the wuFail list.

SEE ALSO

selectors, addhref, addbackref, edithrefs, rmhref, showhrefs

EXAMPLES

- Example of Using whereused to find direct references to a version
- Example of Using whereused to find all references to version
- Example of Using whereused to find immutable tagged versions
- Example of Using whereused to find tagged versions
- Example of Displaying whereused Output in Tcl list

```
Many products use the common code available in LIB module.  To assure
the all the development teams are using a tested and approved version
of the LIB module, the authorized release versions are tagged with
immutable release tags after they are qualified. The version of LIB
in our example is v1.2.  A software defect is discovered in the LIB
module. Product management wants to quickly locate all the products
built with references to that version so that the new version can be
shipped to all customers who have that version.
```

Example of Using whereused to find direct references to a version

```
This whereused example shows all the modules that refer
explicitly to the defective LIB version.
Note: The (*) after a tag indicates that the tag is immutable.

stcl> whereused -version v1.2 -showtags all
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running non-recursively...

 ============================================================

 sync://svr2.ABCo.com:2647/Modules/LIB;1.32 - v1.2(*), Trunk:
      sync://svr2.ABCo.com:2647/Modules/Chip;1.7 - Trunk

 ============================================================


 Finished whereused operation.
```

Example of Using whereused to find all references to version

```
This example shows all the modules that refer to the defective LIB
module explicitly and implicitly.
Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused -recursive sync://svr2.ABCo.com:2647/Modules/LIB;1.32

 Beginning whereused operation ...

 Running recursively...

 ================================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB;1.32 - v1.2(*), Trunk:
      sync://svr2.ABCo.com:2647/Modules/Chip;1.7 - Trunk
         sync://svr2.ABCo.com:2647/Modules/top;1.9 - Trunk:
 ================================================================================
```

```
 Finished whereused operation.
```

Example of Using whereused to find immutable tagged versions

```
In the scenario above, the LIB module is tagged with an immutable
tag.  This usually indicates a significant release such as a released
product version, or, in our case, a released library reused
throughout the code base.  You can use the whereused command to
search for all objects tagged with an immutable tag.

Note: This may not display all the versions on which an immutable
tag is used.  Whereused queries for the information but
displays from the starting point until it reaches the last immutable
tag in a reference tree.

For instance, if you have a hierarchy such this:
top - not tagged with an immutable tag
 -> Chip tagged with an immutable tag
     -> LIB, tagged with an immutable tag,

the output of whereused -showtags immutable includes Chip and LIB.
If instead of Chip, top is tagged with an immutable tag, the output
shows top, Chip, and LIB, even though Chip is not tagged with an
immutable tag.

To perform a query to return all the whereused information for a
specific immutable tag, you can write a filtering script around
whereused using -showtags all and filtering for the desired immutable
tag.

In this example, the top module and LIB module both contain immutable
tags.
Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused -version v1.2 -showtags immutable -rec
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running recursively...

 ================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB - v1.2(*)
      sync://svr2.ABCo.com:2647/Modules/Chip;1.8
         sync://svr2.ABCo.com:2647/Modules/top;1.10 - v1.2(*)
 ================================================================


 Finished whereused operation.
```

Example of Using whereused to find tagged versions

```
This example shows all the tagged versions of modules that refer to
the defective LIB module.
 Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused -version v1.2 -showtags version -rec
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running recursively...

 ================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB - v1.2(*)
      sync://svr2.ABCo.com:2647/Modules/Chip;1.8 - Gold, Latest
         sync://svr2.ABCo.com:2647/Modules/top;1.10 - Latest, v1.2(*)
 ================================================================
```

Example of Displaying whereused Output in Tcl list

```
This example shows all the modules that refer to the defective ROM
module explicitly or implicitly in Tcl list format.

stcl> whereused -recursive -version v1.2 -format list
sync://svr2.ABCo.com:2647/Modules/ROM

{{object sync://svr2.ABCo.com:2647/Modules/LIB version 1.2 depth 0
{v1.2(*), Trunk:} whereused {{object
sync://svr2.ABCo.com:2647/Modules/Chip/ version 1.8  depth
1 {Gold, Trunk:} whereused {{object
sync://svr2.ABCo.com:2647/Modules/top version 1.10 depth 2  {v1.2(*),
Trunk:} whereused {}}}}}}}
```

whereused member

whereused member Command

NAME

```
whereused member    - Lists module versions containing a module member
```

DESCRIPTION

```
This command identifies where a module version is used.
This information includes:
```

```
  o Module version(s) in which the module member version exists.
  o Tags, if any, for the module version in which the module member
     version exists.
  o Module Member Tags in which the module member version exists,
    (also called "snapshot tags")

 This allows users to trace the usage of any desired target.

 This command is subject to access controls on the server. The
 whereused command requires browse access to the objects in the
 module hierarchy. See the ENOVIA Synchronicity Access Control Guide
 for details.
```

SYNOPSIS

```
 whereused member [-format list|text] [-modulecontext <moduleVersionURL>]
                  [-report brief|normal|verbose]
                  [-showtags all|none|immutable|version|member]
                  [-versions <selector>,...] <argument>
```

ARGUMENTS

- Module Member

Module Member

```
 <module_member>   Specify the module member. It can be specified as a
                   workspace module member with an absolute or
                   relative path, or, when using the -modulecontext
                   option, a natural path.
```

OPTIONS

- -format
- -modulecontext
- -report
- -showtags
- -versions

-format

```
 -format list|text  Determines the format of the output.  For
                    details about the information returned
                    see the -report option.
                    Valid values are:

                    o text - Display text output. (Default)
```

o list - Returns a TCL list. See the Examples section for examples of the TLC list formatting.

-modulecontext

-modulecontext <moduleVersionURL>	Identifies a module version for the module member argument. The value must be a module version URL. When the -modulecontext option is used, the argument must be specified as a natural path. 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.

-report

-report brief|normal|verbose	Determines what the information is returned in the output of the whereused command. Valid values are: o brief - displays only information for tagged module versions. o normal - displays information for all versions of the module containing the specified object. o verbose - provides the information available in report mode normal as well as the processing status of the command and the natural path of the object.

-showtags

-showtags all|none|immutable|version|member	Specifies whether tag information is displayed and optionally restricts the output to immutable tags tagged versions, or module members. o all - Displays module versions, module version tags and member tags. (Default) o none - Displays module versions, but does not display tag information. o immutable - Displays only module versions tagged

```
  with an immutable tag, and the name of the tag.

o version - Displays any module versions that have
a version tag, and the name of the tag.

o member -  Displays only the member tags applied
  to the member version.
```

-versions

```
-versions          Comma separated set of member versions to
 <selector>        locate. The versions can be numerics or all
                   selectors. By default, this option defaults to the
                  version in the workspace. When using
                  -modulecontext, this option defaults to the member
                  version associated with the specified module
                  context.
```

RETURN VALUE

```
This command does not return a Tcl value in text mode.  If the command
is unable to run,  DesignSync throws an error explaining the failure.

In list mode, the list of module versions is returned in an array of
name/value pairs.
```

SEE ALSO

```
selectors, whereused module
```

EXAMPLES

- Example Showing the whereused member Commmand
- Example Showing the whereused member Commmand in List Mode

Example Showing the whereused member Commmand

```
This example shows locating where the mem.cpp module member is used.
The output is text in normal mode.

stcl> whereused member -versions 1.1,1.2 mem.cpp
    mem.cpp;1.1

    Module Version(s)    Branch  Tags
    ----------------------------------------
    1.2                  Trunk   alpha, pass1
```

```
    1.3 - 74             Trunk
    1.75                 Trunk   beta, pass2
    1.76 - 124           Trunk
    1.125                Trunk   Bronze
    1.126 - 268          Trunk
    1.4.1.1              Dev
    1.4.1.2              Dev     hack1
    1.4.1.3 - 66         Dev
    1.4.2.1 - 2          Test

       Member Version Tags
       -----------------------
       Rel_1
       Rel_2

  ====================================

  mem.cpp;1.2

    Module Version(s)  Branch  Tags
       ----------------------------------------
       1.269              Trunk   Checkpoint

  ====================================
```

Example Showing the whereused member Commmand in List Mode

```
This example shows locating where the mem.cpp module member is used.
The output is a list in verbose mode.

stcl> whereused member -format list -report verbose mem.cpp
{name mem.cpp versions {
  {member_version 1.1
     module_versions {
      {version_range {1.2 1.2} branch Trunk version_tags {alpha
pass1} natural_path /mem.cpp}
      {version_range {1.3 1.45} branch Trunk version_tags {}
natural_path /subdir/mem.cpp}
      {version_range {1.46 1.74} branch Trunk version_tags {}
natural_path /mem.cpp}
      {version_range {1.75 1.75} branch Trunk version_tags {beta
pass2} natural_path /mem.cpp}
      {version_range {1.76 1.124} branch Trunk version_tags {}
natural_path /mem.cpp}
      {version_range {1.126 1.268} branch Trunk version_tags {}
natural_path /mem.cpp}
      {version_range {1.4.1.1 1.4.1.1} branch Dev version_tags {}
natural_path /mem.cpp}
      {version_range {1.4.1.2 1.4.1.2}  branch Dev version_tags hack1
natural_path /mem.cpp}
      {version_range {1.4.1.3 1.4.1.66} branch Dev version_tags {}
natural_path /mem.cpp}
      {version_range {1.4.2.1 1.4.2.2} branch Test version_tags {}
natural_path /mem.cpp}
```

```
    }
    member_tags {
       {member_tag Rel_1 natural_path /mem.cpp}
       {member_tag Rel_2 natural_path /subdir/mem.cpp}
    }
 }
 {member_version 1.2
    {module_versions {
      {version_range {1.269 1.269} branch Trunk version_tags
Checkpoint natural_path /mem.cpp}
    }
    member_tags {}
 }
}
```

whereused module

whereused module Command

NAME

```
whereused module    - Traces the use of hierarchical references
```

DESCRIPTION

```
This command identifies the module versions in which a sub-module
version is used, identified by a selector. This allows users to trace
the usage of any desired target.

When a hierarchical reference is created, DesignSync creates a back
reference on the toargument indicating that it is referenced by the
fromargument module. This provides the basic mechanism for the
whereused functionality. If a back reference does not exist, for
example, if the hierarchical reference was created before whereused
was implemented,  you can add back references independently using the
addbackref command.

Note: When the hierarchical reference target (toargument) was
specified with SSL protocol, the whereused module command must also
be specified with SSL.

This command is subject to access controls on the server. The
whereused module command requires browse access to the objects in the
module hierarchy. See the ENOVIA Synchronicity Access Control Guide
for details.
```

SYNOPSIS

```
whereused module [-format list|text] [-[no]recursive] [-report <mode>]
                 [-showtags all|none|immutable|version]
                 [-versions <selector>,...] <argument>
```

ARGUMENTS

- Server Module

Server Module

```
<server module>    URL of the referenced module.
                   Specify the URL as follows:
                   sync[s]://<host>[:<port>]/Modules/[<Category>...]
                    /<module>[;<selector>,...]
                   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 <selector> is the optional
                   selector.

                   If a selector is not specified, the command
                   uses versions specified with the -versions option
                   or the default value of Trunk:Latest.

                   Notes:
                     o If you specify a selector in the argument, you
                       cannot specify additional selectors with the
                       -versions option. To locate more than one
                       selector at a time, use the -versions option.

                     o When the hierarchical reference target
                       (toargument) was specified with SSL protocol,
                       the whereused module command must also be
                       specified with SSL.
```

OPTIONS

- -format
- -[no]recursive
- -report
- -showtags
- -versions

-format

```
-format list|text  Determines the format of the output.  For
                   details about the information returned
                   see the -report option.
                   Valid values are:
```

o text - Display text output formatted to show the hierarchical reference tree. (Default)

o list - Displays a list with the following format:

```
{
  object <object>...whereused
  <hierarchical_whereused_list>
}
```

-[no]recursive

-[no]recursive

Determines whether to show the locations in which the version is explicitly referenced, or show all modules in which the version is implicitly or explicitly referenced. An explicit reference exists when there is a direct reference link between the module and the target. An implicit reference exists when the module and target are not directly connected, but within the module's hierarchy exists a reference to the target. For example: if the Chip module references the Gold version of the ALU module, and the Gold version of the ALU module references the Gold version of the ROM module, the Chip module contains an implicit reference to the ROM module and an explicit reference to the ALU module.

-norecursive expands the reference and returns a list of modules that contain an explicit reference to the referenced module. (Default)

-recursive traverses the reference and returns a list of all modules that explicitly and implicitly reference the specified version.

-report

-report brief|normal|verbose

Determines what the information is returned in the output of the whereused module command.

Valid values are:

o brief - reports only the module names for the specified object ordered by specified version.

When displayed in text format, the output is indented to show the reverse hierarchical reference path. When displayed in list format, the hierarchical reference depth is indicated

with the depth property.

Note: The -showtags option is ignored for this report mode.

- normal - reports the version of the module containing the reference to the specified object. The text format is indented to show hierarchical reference depth, and the list format uses the depth property. The version property in list mode contains the version number.

- verbose - provides the information available in report mode -normal as well as the processing status of the command.

-showtags

-showtags all|none|immutable|version	Specifies whether tag information is displayed and optionally restricts the output to immutable tags or tagged versions.

- all - Displays all tags and all reference locations, including references that are not tagged. (Default)

- none - Displays all references locations, but does not display tag information

- immutable - Displays only reference locations tagged with an immutable tag and the name of the immutable tag.

  Note: Using the -showtags immutable option may not display all versions in which an immutable tag is used. The whereused module command queries for all the whereused information but filters the display from the starting point until it reaches the last immutable tag in a reference tree.

- version - Displays any reference location that has a version tag and the name of the tag.

Note: The -showtags option is ignored for -report brief mode.

-versions

-versions	Comma separated set of versions to locate. The

```
<selector>        versions can be numerics or selectors. The
                  selector resolves to the numeric value of the
                 module version and is compared against the static
                 href value.

                  Note: To specify a comma separated selector list,
                  use the version extended naming format, not the
                  -versions option.
```

RETURN VALUE

```
This command does not return a Tcl value in text mode.  If the command
is unable to run,  DesignSync throws an error explaining the failure.

If whereused module is unable to process a reference, you will see
the reason for failure during command processing and the command adds
the sync URL of the failed module to the wuFail list.

In list mode, the list of modules is returned in an array of
name/value pairs. If a reference cannot be processed, the command
adds an error property containing the reason for the failure and the
module is added to the wuFail list.
```

SEE ALSO

```
selectors, addhref, addbackref, edithrefs, rmhref, showhrefs
```

EXAMPLES

- Example of Using whereused module to find direct references to a version
- Example of Using whereused module to find all references to version
- Example of Using whereused module to find immutable tagged versions
- Example of Using whereused module to find tagged versions
- Example of Displaying whereused module Output in Tcl list

```
Many products use the common code available in LIB module.  To assure
the all the development teams are using a tested and approved version
of the LIB module, the authorized release versions are tagged with
immutable release tags after they are qualified. The version of LIB
in our example is v1.2.  A software defect is discovered in the LIB
module. Product management wants to quickly locate all the products
built with references to that version so that the new version can be
shipped to all customers who have that version.
```

Example of Using whereused module to find direct references to a version

```
This whereused module example shows all the modules that refer
explicitly to the defective LIB version.
Note: The (*) after a tag indicates that the tag is immutable.

stcl> whereused module -version v1.2 -showtags all
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running non-recursively...

 ============================================================

 sync://svr2.ABCo.com:2647/Modules/LIB;1.32 - v1.2(*), Trunk:
      sync://svr2.ABCo.com:2647/Modules/Chip;1.7 - Trunk

 ============================================================


 Finished whereused operation.
```

Example of Using whereused module to find all references to version

```
This example shows all the modules that refer to the defective LIB
module explicitly and implicitly.
Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused module -recursive
  sync://svr2.ABCo.com:2647/Modules/LIB;1.32


 Beginning whereused operation ...

 Running recursively...

 ======================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB;1.32 - v1.2(*), Trunk:
      sync://svr2.ABCo.com:2647/Modules/Chip;1.7 - Trunk
         sync://svr2.ABCo.com:2647/Modules/top;1.9 - Trunk:
 ======================================================================


 Finished whereused operation.
```

Example of Using whereused module to find immutable tagged versions

```
In the scenario above, the LIB module is tagged with an immutable
tag.  This usually indicates a significant release such as a released
product version, or, in our case, a released library reused
```

```
throughout the code base.  You can use the whereused command to
search for all objects tagged with an immutable tag.

Note: This may not display all the versions on which an immutable
tag is used.  Whereused module queries for the information but
displays from the starting point until it reaches the last immutable
tag in a reference tree.

For instance, if you have a hierarchy such this:
top - not tagged with an immutable tag
 -> Chip tagged with an immutable tag
     -> LIB, tagged with an immutable tag,

the output of whereused module -showtags immutable includes Chip and
LIB. If instead of Chip, top is tagged with an immutable tag, the
output shows top, Chip, and LIB, even though Chip is not tagged with
an immutable tag.

To perform a query to return all the whereused information for a
specific immutable tag, you can write a filtering script around
whereused using -showtags all and filtering for the desired immutable
tag.

In this example, the top module and LIB module both contain immutable
tags.
Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused module -version v1.2 -showtags immutable -rec
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...

 Running recursively...

 ======================================================================

 sync://svr2.ABCo.com:2647/Modules/LIB - v1.2(*)
      sync://svr2.ABCo.com:2647/Modules/Chip;1.8
         sync://svr2.ABCo.com:2647/Modules/top;1.10 - v1.2(*)
 ======================================================================


 Finished whereused operation.
```

Example of Using whereused module to find tagged versions

```
This example shows all the tagged versions of modules that refer to
the defective LIB module.
 Note: The (*) after a tag indicates that the tag is immutable.

dss> whereused module -version v1.2 -showtags version -rec
sync://svr2.ABCo.com:2647/Modules/LIB

 Beginning whereused operation ...
```

```
Running recursively...

================================================================

sync://svr2.ABCo.com:2647/Modules/LIB - v1.2(*)
     sync://svr2.ABCo.com:2647/Modules/Chip;1.8 - Gold, Latest
        sync://svr2.ABCo.com:2647/Modules/top;1.10 - Latest, v1.2(*)
================================================================
```

Example of Displaying whereused module Output in Tcl list

```
This example shows all the modules that refer to the defective ROM
module explicitly or implicitly in Tcl list format.

stcl> whereused module -recursive -version v1.2 -format list
sync://svr2.ABCo.com:2647/Modules/ROM

{{object sync://svr2.ABCo.com:2647/Modules/LIB version 1.2 depth 0
{v1.2(*), Trunk:} whereused {{object
sync://svr2.ABCo.com:2647/Modules/Chip/ version 1.8  depth
1 {Gold, Trunk:} whereused {{object
sync://svr2.ABCo.com:2647/Modules/top version 1.10 depth 2  {v1.2(*),
Trunk:} whereused {}}}}}}}
```

whereused vault

whereused vault Command

NAME

```
whereused vault     - Traces the use of hierarchical references
```

DESCRIPTION

```
This command identifies the module versions in which either of the
following targets (the "toargument" for the addhref command) are
used: legacy module configuration, or DesignSync vault, identified by
a selector. This allows users to trace the usage of any desired
target.

When a hierarchical reference is created, DesignSync creates a back
reference on the toargument indicating that it is referenced by the
fromargument module. This provides the basic mechanism for the
whereused functionality. If a back reference does not exist, for
example, if the hierarchical reference was created before whereused
was implemented,  you can add back references independently using the
addbackref command. The whereused vault command performs no
validation on specified legacy module and vault selectors. Verify
```

```
that the selector specified is correct before executing the command.

IMPORTANT: When referencing either legacy modules or DesignSync
vaults using a configuration name or a selector as the specified
version, the version must be an exact match for the selector used
when the hierarchical reference was added.

Note: When the hierarchical reference target (toargument) was
specified with SSL protocol, the whereused command must also be
specified with SSL.

This command is subject to access controls on the server. The
whereused vault command requires browse access to the objects in the
module hierarchy. See the ENOVIA Synchronicity Access Control Guide
for details.
```

SYNOPSIS

```
whereused vault [-format list|text] [-[no]recursive] [-report <mode>]
                [-showtags all|none|immutable|version]
                [-versions <selector>,...] <argument>
```

ARGUMENTS

- DesignSync Vault
- Legacy Module URL

DesignSync Vault

```
<DesignSync vault> URL of the referenced DesignSync vault.
                   Specify the URL as follows:
                   sync[s]://<host>[:<port>]<vaultpath>;[<selector>]
                    where <host> is the SyncServer on which the
                   object resides, <port> is the SyncServer
                   port number, <vaultpath> identifies the
                   DesignSync object, and <selector> is the optional
                   selector.

                   If a selector is not specified, the command
                   uses versions specified with the -versions option
                   or the default value of Trunk:Latest.

                   Notes:
                     o If you specify a selector in the argument, you
                       cannot specify additional selectors with the
                       -versions option. To locate more than one
                       selector at a time, use the -versions option.

                     o When the hierarchical reference target
                       (toargument) was specified with SSL protocol,
                       the whereused vault command must also be
```

```
                        specified with SSL.

                      o The vault selector is not checked for validity
                        before the command is executed. Verify that
                        the vault URL and selector is typed correctly
                        before executing the command.
```

Legacy Module URL

```
 <Legacy module      URL of the referenced legacy module.
  URL>               Specify the URL as follows:
                     sync[s]://<host>[:<port>]/[Projects]/<vaultpath>;
                      [<selector>,...]
                     where <host> is the SyncServer on which the
                     module resides, <port> is the SyncServer
                     port number, vaultpath is the path to the
                     module, and <selector> is the optional selector,
                     or configuration name.

                     If a selector is not specified, the command
                     uses versions specified with the -versions option
                     or the default value of Trunk:Latest.

                     Notes:
                       o If you specify a selector in the argument, you
                         cannot specify additional selectors with the
                         -versions option. To locate more than one
                         selector at a time, use the -versions option.

                       o When the hierarchical reference target
                         (toargument) was specified with SSL protocol,
                         the whereused vault command must also be
                         specified with SSL.

                       o The vault selector is not checked for validity
                         before the command is executed. Verify that
                         the vault URL and selector is typed correctly
                         before executing the command.
```

OPTIONS

- -format
- -[no]recursive
- -report
- -showtags
- -versions

-format

```
 -format list|text  Determines the format of the output.  For
```

```
details about the information returned
see the -report option.
Valid values are:

o text - Display text output formatted to show the
hierarchical reference tree. (Default)

o list - Displays a list with the following
         format:
          {
            object <object>...whereused
            <hierarchical_whereused_list>
          }
```

-[no]recursive

```
-[no]recursive     Determines whether to show the locations in
                   which the version is explicitly referenced, or
                  show all modules in which the version is implicitly
                  or explicitly referenced. An explicit reference
                  exists when there is a direct reference link
                  between the module and the
                  target.  An implicit reference exists when the
                  module and target are not directly connected, but
                  within the module's hierarchy exists a reference
                  to the target. For example: if the Chip module
                  references the Gold version of the ALU module, and
                  the Gold version of the ALU module references the
                  Gold version of the ROM module, the Chip module
                  contains an implicit reference to the ROM module
                  and an explicit reference to the ALU module.

                   -norecursive expands the reference and returns a
                   list of modules that contain an explicit reference
                   to the referenced module.  (Default)

                   -recursive traverses the reference and returns a
                   list of all modules that explicitly and implicitly
                   reference the specified version.
```

-report

```
-report brief|     Determines what the information is returned in the
 normal|verbose    output of the whereused vault command.

                   Valid values are:

                   o brief - reports only the module names for the
                     specified object ordered by specified version.

                     When displayed in text format, the output is
```

indented to show the reverse hierarchical reference path. When displayed in list format, the hierarchical reference depth is indicated with the depth property.

Note: The -showtags option is ignored for this report mode.

- normal - reports the version of the module containing the reference to the specified object. The text format is indented to show hierarchical reference depth, and the list format uses the depth property. The version property in list mode contains the version number.

  Note: The version property may be a selector rather than a version number if the target is a DesignSync vault or legacy module.

- verbose - provides the information available in report mode -normal as well as the processing status of the command.

-showtags

-showtags all|none|immutable|version	Specifies whether tag information is displayed and optionally restricts the output to immutable tags or tagged versions.

- all - Displays all tags and all reference locations, including references that are not tagged. (Default)
- none - Displays all references locations, but does not display tag information
- immutable - Displays only reference locations tagged with an immutable tag and the name of the immutable tag.

  Note: Using the -showtags immutable option may not display all versions in which an immutable tag is used. The whereused vault command queries for all the whereused information but filters the display from the starting point until it reaches the last immutable tag in a reference tree.

- version - Displays any reference location that has a version tag and the name of the tag.

Note: The -showtags option is ignored for -report brief mode.

-versions

```
-versions          Comma separated set of versions to locate. The
 <selector>        versions can be numerics or selectors. The version
                   selector (branch, version release, or alias
                  selector) is validated by string comparison
                  against the dynamic href value.

                   Note: To specify a comma separated selector list,
                   use the version extended naming format, not the
                   -versions option.
```

RETURN VALUE

```
This command does not return a Tcl value in text mode.  If the command
is unable to run,  DesignSync throws an error explaining the failure.

If whereused vault is unable to process a reference, you will see the
reason for failure during command processing and the command adds
the sync URL of the failed module to the wuFail list.

In list mode, the list of modules is returned in an array of
name/value pairs. If a reference cannot be processed, the command
adds an error property containing the reason for the failure and the
module is added to the wuFail list.
```

SEE ALSO

```
selectors, addhref, addbackref, edithrefs, rmhref, showhrefs
```

EXAMPLES

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          Specifies the version of a file to display.
 <selector>       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

- Understanding the Types of Possible Compare Operations
- Understanding the Output
- Understanding Status Values in the Ouput
- Running Compare on Modules (Module-based)
- Understanding Columns Returned When Comparing Module Objects (Module-based)
- Using Compare with Legacy Module Objects (Legacy-based)
- Using Compare with File-Based Objects (File-based)
- Understanding Columns Returned When Comparing File Objects (File-based)

```
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:

- This command is not intended to provide a way to compare two different modules, so there is no way to specify two module URLs.
- 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)
- DesignSync Folder (File-based)
- Server Folder (File-based)

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

- -exclude
- -filter (Module-based)
- -format
- -[no]history (File-based)
- -hreffilter (Module-based)
- -hrefmode (Module-based)
- -hrefmode2 (Module-based)
- -modulecontext (Module-based)
- -output
- -[no]path
- -[no]recursive (Module-based)
- -[no]recursive (Legacy-based)
- -[no]recursive (File-based)
- -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> |   Outputs the result to the specified file or
 -stream <stream>   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.

- o 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.

- o 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
- Example of Comparing Two Selectors with a URL
- Example of Comparing the Current Directory Against Another Directory
- Example of how to use '-format list' option
- Example Comparing a Workspace to a Server Module Version (Module-based)
- Example of Compare the Current Workspace Against A Module (Module-based)
- Example of Current Workspace Against Server Module Version (Module-based)
- Example of Comparing a Module with different Hrefmodes (Module-based)
- Example of Comparing a Tagging Module Version Against Latest (Module-based)
- Example of Comparing the Workspace Version to the Server Version (File-based)
- Example of Comparing Two Workspace Directories (File-based)

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
- -path
- -recursive (Module-based)
- -recursive (Legacy-based)
- -recursive (File-based)
- -report
- -selector (Module-based)
- -selector (File-based)
- -stream
- -version
- -view (Module-based)
- --

-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 | text>

Specifies the format of the output. The format can 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:
- 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 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:

o 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).

o brief - Displays the same information as 'normal'.

o normal - Include header information and progress lines where a long command might be performed, such as when scanning the vault. (Default)

o verbose - Include information on configuration mappings.

-selector (Module-based)

-selector <selector>

A valid selector or selector list.
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.

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

- Notes for Collection Objects
- Note for Modules (Module-based)

```
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 Object (File-based)

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 (File-based)
- -annotate
- -binary
- -case
- -embed
- -file1
- -file2
- -gui
- -kk
- -member
- -modulecontext (Module-based)
- -output
- -standard
- -syncdiff
- -unified
- -usemoduleversions (Module-based)
- -version (Module-based)
- -version (File-based)
- -white
- --

-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
- Examples of Comparing a File Against the Latest Server Version
- Example of Comparing a File Against A Specified Version
- Example of Comparing Original File Against Latest Server Version
- Example of Showing Conflicts in Your Local Version
- Examples of Comparing Collection Cell View Versions
- Example of Comparing Against the Local Cell View Version
- Example of Comparing Files Using the Module Version (Module-based)
- Example of Comparing Files Using the Member Version (Module-based)
- Example Comparing a Module Member to a Non-Local Module Member (Module-based)
- Example of Specifying the Module Version with the Ancestor * Option (Module-based)

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
- Example of Using -nothrow with locate

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

- Notes for Module Objects and Module Snapshots (Module-based)
- Notes on Legacy Modules (Legacy-based)
- Notes for Files-Based Objects (File-based)
- Report Options
- Report Data Keys Table (Module-based)
- Status Values for Modules and Modules Members (Module-based)
- Report Data Keys Table (File-based)

- Status Values for File-Based Objects (File-based)

```
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 Property     From   # of
 Key  Name         Vault? 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.

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 |     Provides information about the specified module
 module folder>      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 |  Provides information about the specified
 DesignSync folder |  DesignSync object or folder. If you specify a
 unmanaged object |    folder, you can use the -recursive option to
 unmanaged folder>     get information about all subfolders contained
                       in the specified folder.
```

OPTIONS

- -[no]addselect
- -branch
- -[un]changed
- -exclude
- -filter (Module-based)
- -format (Module-based)
- -format (File-based)
- -fullpath
- -[no]header
- -hreffilter (Module-based)
- -[un]locked (Module-based)
- -[un]locked (File-based)
- -[un]managed (Module-based)
- -[un]managed (File-based)
- -merged
- -[un]modified (Module-based)
- -[un]modified (File-based)
- -modulecontext (Module-based)
- -[no]needsmerge
- -output
- -path
- -[no]recursive (Module-based)
- -[no]recursive (Legacy-based)
- -[no]recursive (File-based)
- -report

- -[no]selected
- -stream
- -[non]versionable
- -workspace/-vault
- -writeableunlocked (Module-based)
- -writeableunlocked (File-based)
- -xtras (Module-based)
- --

-[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|      Determines whether to display only objects that
 rename|remove    have been modified as the result of a merge into
 all|""             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 [-branch <branch>]	Determines whether to show only objects that 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.
- o 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.
- o verbose Displays most fields (equivalent to '-report OXMSRUNCTA').
- o status Displays status fields (equivalent to '-report MSRUN').
- o K[K...] Displays the fields corresponding to the data keys, where K is a data key listed in the Report Options table above.
- o +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'.
- o -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

- 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.

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
- Example Showing the Contents of the Specified Folder
- Example Showing Objects that Need to be Merged
- Example Showing Objects that do not Need to be Merged
- Example Showing a Recursive Directory Listening
- Example Showing the ls Output in List Format
- Example Showing Locked Objects in the Workspace
- Example Showing All Locked Objects
- Example Showing All Locked Objects with Users
- Example Showing Locked Server Objects Using Status Report Mode
- Example Showing Locked Workspace Objects in Status Report Mode
- Example Showing Unmanaged Objects in Current Folder
- Example Showing Unlocked Writable Objects in the Workspace
- Example Showing Excluding Objects
- Example Showing a Variety of ls Commands To Display Object Vault
- Examples Showing Writing to an Output File or TCL stream
- Example Showing Locked References
- Example Showing Collection List
- Example Showing Module Structural Changes (Module-based)
- Example Showing the Contents of a Legacy Module Configuration (Legacy-based)

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
```

showhrefs

showhrefs Command

NAME

```
showhrefs           - Displays the hierarchy of a module
```

DESCRIPTION

- External Module Support
- Understanding the Output

```
This command displays the hierarchical references for a module or a
legacy module configuration.  When run recursively, this command
also displays the hierarchical references for all modules and legacy
module configurations in the hierarchy.

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.
```

External Module Support

DesignSync supports showing the hierarchical reference status of an external module to determine state of the hierarchical references. After an external module has been populated, the showhref command can be available to query the status of the external module hierarchical reference and return the results.

Hierarchical reference conflicts are reported in the final list output when all of the following conditions are true;
- conflict property is set to yes during command processing
- conflicts option is provided

These properties are not, however; returned in the final list output or included in the conflict summary.

Over-ridden hierarchical references can also be reported in the final output when the -overridden option is used with the -report verbose.

For information on populating an external module, see the populate command. For information on configuring showstatus for external modules, see the DesignSync Administrator's Guide.

Understanding the Output

The output of the showhrefs 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 showhrefs command, by default, displays the following information:

Column Titles	Property Names	Description
Name	name	The name of the href. For example, CPU.
URL	url	The URL of the referenced module, external module, legacy module configuration, DesignSync vault, or IP Gear deliverable. The URL for all references except external modules, includes host, port, and vaultPath. The external modules URL identifies the reference as an external module href and contains the <external-type> and the <external-data> strings.
Selector	selector	The selector for the href as supplied to the add href command. This varies depending on the type of href.

		o modules - the selector list used to identify the referenced module version. o legacy modules - the name of the referenced legacy module configuration. o DesignSync vaults - the selector list used to identify the referenced vault versions. For all other object types, the selector field is empty.
Version	version	The version of the referenced module the selector resolved at the time the href was created. If the href is not a module, the version field is empty.
Type	type	The type of object referenced. o Module - href to a module. o Alias - href to a legacy module alias. o Branch - href to a legacy module branch configuration. o External - href to an external module. o Release - href to a legacy module release configuration.. o Selector - href to a legacy module selector configuration. o Vault - href to a DesignSync vault. o Deliverable - href to an IP Gear deliverable. o Unknown - indicates that the object type could not be determined at the time the href was created.
	conflict	When the -conflict option is specified, the report reports conflicting hierarchical references. In text mode, a *CONFLICT* identifier is appended to the lines showing the conflicting URLs. In list mode, the command uses the conflict key. The value of the key is "yes," if the href is in conflict, or "no," if the href is not in conflict. In text mode, a conflict line is written beneath the table for each href in conflict during processing and, at the end of the output, a summary table shows all the conflicting hrefs. When populating a workspace module, with a V6R2015x client or later, showhrefs marks the conflicts at fetch time. And this option displays the conflicting references along with information about which modules are fetched into the workspace during populate operations. When populating a server-side module,

		conflict detection occurs on the fly while processing the hierarchy on all associated servers. The first conflicting href that is processed may not be known until another conflicting href is found and therefore the first conflicting href may not be marked as such. All subsequent conflicting hrefs will be marked with a line indicating the href is in-conflict with another href. The first conflicting href may not show as in conflict in the initial href report, but does in the summary table.
Relative Path	relpath	The relative path from the base directory of the upper-level module (Parent) to the base directory of the submodule. This path is used by the populate command when you fetch the module into your work area.
	instance_name	The instance identifier for the workspace module version. This is not applicable if the specified argument is a server module.
	basedir	The workspace base directory of the module. This is not applicable if the specified argument is a server module.

If you run the showhrefs command with '-report brief' it displays:

Column Titles	Property Names	Description
Name	name	The module name of the top level module and the href name for all referenced submodules, legacy module configurations, IP Gear deliverables, and DesignSync vaults.
URL/Base Directory	url/ basedir	Path to the module. If the command is performed on the server, it includes the full sync URL for the module, including selector information. If the command is performed on the workspace, it includes the full directory path. For external modules, it always displays the URL as: sync:///ExternalModule/<external-type>/<external-data>
	conflict	When the -conflict option is specified, the report reports conflicting hierarchical references. In text mode, a *CONFLICT* identifier is appended to the lines showing the conflicting URLs. In list mode, the command uses the conflict key. The value of the key is "yes," if

```
                              the href is in conflict, or "no," if the
                              href is not in conflict. The first
                              conflicting href may not show as in
                              conflict. For more information, see
                              the "conflict" description above in the
                              report -normal output section.

Note: The -brief output indents the module name to graphically
represent the  hierarchical.

If you run the report in -verbose mode with the -overridden option,
it adds the following column:

 Column       Property
 Titles       Names          Description
 ------       -----          ------------
 Overridden   overridden     Whether this particular href is
                             overridden by a higher level href.
                             yes - this URL is ignored by system
                                   because it is overridden by a
                                   higher level href.
                             no - this URL is active; not overridden.

If you specify the -overridden option, all submodules, even the
overridden submodules, display with their own showhrefs table.

As showhrefs recurses a hierarchy, it counts any errors and displays
suitable error messages. At the end of the operation, showhrefs
displays an error message which contains the number of errors that
occurred.

Using the -format list option formats the output into a Tcl string
that can be processed in scripts, however the order and the property
names differ slightly from the -format text (default) option.  For
more information, see Example 3 which shows using the -format list
option with report -normal and Example 4 which shows report -brief.
```

SYNOPSIS

```
showhrefs [-[no]conflicts] [-format text|list]
          [-hrefmode {dynamic | static | normal}]
          [no]overridden] [-[no]recursive]
          [-report {brief | normal | verbose | script}]
          [-[no]stopatconflict] [-xtras <xtras>] <argument>
```

ARGUMENTS

- Server Module
- Workspace Module
- External Module Instance
- Server Legacy Module
- Workspace Legacy Module

Server Module

`<server module>` Shows hierarchical references for a module version. Specify the URL as follows:
`sync[s]://<host>[:<port>]/Modules/[<category>...]<module>[;<selector>]`

where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, <category> identifies the path to the module, <module> is the name of the module, and <selector> identifies a particular branch or version. You may use this format to specify a module, module branch, or module version. The default branch is "Trunk." The default version is "Latest.

Workspace Module

`<workspace module>` Shows hierarchical references for a workspace module. You can specify the workspace module by using the module name, if it's unique within the workspace (For example: Chip), or the workspace module instance name. (For example: Chip%0).

External Module Instance

`<external_mod>` Specifies the external module instance. The external module must be populated into the workspace.

Server Legacy Module

`<server module>` Shows hierarchical references for a legacy module. Specify the URL as follows:
`sync[s]://<host>[:<port>]/<vaultPath>[;<selector>]`
where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, <vaultPath> identifies the module, and <selector> identifies a legacy module configuration. The default branch is "Trunk." The default version is "Latest.

Note: If no selector information is provided for legacy modules, DesignSync uses the default configuration.

Workspace Legacy Module

```
<workspace module>  Shows hierarchical references for a workspace
                    module. You can specify the legacy module using
                    the path of the workspace. (Legacy Modules Only)
```

OPTIONS

- -[no]conflict
- -format
- -hrefmode
- overridden_option
- -[no]recursive
- -report
- -[no]stopatconflict
- -xtras

-[no]conflict

```
-[no]conflict          Determines whether to show conflicting
                       hierarchical references; multiple references
                       to different versions of the same module; in
                       the output.

                       -[no]conflict does not show module
                       conflicts for server module arguments.  For
                       workspace module argument, if conflicting
                       hrefs are populated into the workspace, the
                       conflict is reported. (Default)

                       -conflicts shows, for each module with a
                        conflicting hierarchical reference,
                        information about which submodule version was
                        or was not populated because of a
                        conflict.

                       Notes:
                       o If the -noconflicts option is specified with the
                         -stopatconflicts option, the -stopatconflicts
                         option is silently ignored.

                       o This option is only applicable to modern
                         modules. If specified for a legacy module
                         argument,   the option is silently ignored.
```

-format

```
-format <type>         Determines the format of the output.
                       Valid values are:
                       o list -  Displays a list with the following
```

format:
{
name <name>

}

o text - Display a text table with headers and columns. (Default)

-hrefmode

-hrefmode	Indicates how the hierarchy is traversed by the command. Note: This option is ignored for workspace modules which are always traversed the way they were loaded into the workspace. Valid values are: o dynamic - Resolves the href selector to determine the referenced module version to expand. o static - Expands the module version to which the href selector resolved when the href was created. o normal - Resolves the href selector to determine the referenced module version to expand. If, while traversing the hierarchy, the showhrefs command reaches a static selector, for example a version tag or numeric version ID, the hrefmode switches to static for the remaining sub-hierarchy of the referenced module.(Default) This option respects 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.
-[no]overridden	Determines whether to display information about overridden hrefs. -nooverridden - Ignores overridden hrefs and does not include them into the href traversal. (Default) -overridden - Traverses the module hierarchy, including displaying overridden hrefs and showing a table for each href.

Note: In order to see overridden hrefs, you must specify BOTH -overridden AND -report verbose mode.

-[no]recursive

-[no]recursive	Determines whether to display hierarchical references for the specified module, or the specified module and all submodules. -norecursive displays hierarchical references for the specified module. (Default) -recursive displays hierarchical references for the specified module and all submodules.

-report

-report	Specifies the information output. The information each option returns is discussed in detail in the "Understanding the Output" section above. Valid values are: o brief - Displays the Name of the hierarchical reference and the module path to the reference. If the -showconflicts option is selected, a *CONFLICT* identifier showing the conflicting hrefs. The brief mode does not provide information about which href would be populated. o normal - Displays a list of hierarchical references and their properties. (Default) o verbose - Displays the information available with normal. o script - Returns a Tcl list of config_name/property_list pairs.

-[no]stopatconflict

-[no]stopatconflict	Specifies whether the command should continue processing a hierarchy recursively after reaching a conflict in the hierarchy or stop. -nostopatconflict continues recursing through the directory hierarchy regardless of how many conflicts are revealed. (Default)

```
                        -stopatconflict stops processing recursively
                         through the directory hierarchy when the
                         first conflict within the hierarchy is
                         exposed.

                        Notes:
                          o When processing a workspace module, the
                            traversal only includes recursion through
                            the fetched submodules and it always stopped
                            at conflicting hrefs not in the workspace.

                          o If the -noconflict option is selected,
                            with the -stopatconflict option, the
                            -stopatconflict option is silently
                            ignored.
```

-xtras

```
 -xtras <xtras>           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

```
 If you run the showhrefs 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 ("").

 For a description of the output, see the "Understanding the Output"
 section.
```

SEE ALSO

```
 addhref, edithrefs, populate, showmcache, showstatus, rmhref,
 command defaults
'
```

EXAMPLES

- Example of Displaying the Hierarchical References on the Server
- Example of Showing Hrefs on the Server Vault in List Format
- Example of Displaying the Hierarchical References in a Workspace
- Example of Showing Hrefs on the Workspace in List Format
- Example of Showing Hrefs in Brief Report mode and List Format

- Example Showing Overridden Hrefs in the Workspace in Text Format
- Example Showing Overridden Hrefs in List Format
- Example Showing Conflicting Hierarchical References in the Workspace

```
The following examples all use this hierarchy, which is the example
created with the addhref command:

Chip           <= uses the default configuration
  Cpu          <= uses the Gold version
    ROM        <= uses the Gold version
```

Example of Displaying the Hierarchical References on the Server

```
This example displays the hierarchical references for the Chip
module in text mode, which is specified by the sync URL.

dss> showhrefs -recursive sync://srvr2.ABCo.com:2647/Modules/Chip

 Beginning showhrefs operation ...

 Showing hrefs of module sync://srvr2.ABCo.com:2647/Modules/Chip (1.7) ...
 sync://srvr2.ABCo.com:2647/Modules/Chip: Href mode is normal.

 Name  Url                                  Version  Type    Relative Path
 ------------------------------------------------------------------------
 CPU sync://srvr2.ABCo.com:2647/Modules/CPU  Gold     Module  CPU

 ========================================================================

 Showing hrefs of module sync://srvr2.ABCo.com:2647/Modules/CPU;Gold (1.3)
...

 sync://srvr2.ABCo.com:2647/Modules/CPU;Gold: Href mode is static.

 Name  Url                                Version  Type    Relative Path
 ----------------------------------------------------------------------
 ROM   sync://srvr2.ABCo.com:2647/Modules/ROM  1.4      Module  /ROM

 sync://srvr2.ABCo.com:2647/Modules/ROM;Trunk:: Module has no
 hierarchical references.

Finished showhrefs operation.
```

Example of Showing Hrefs on the Server Vault in List Format

```
This example displays hierarchical references specifying a server
module, sync://srvr2.ABCo.com/Modules/Chip,  with report -normal
output. This example uses the hierarchy described at the beginning of
the examples section.
```

```
dss> showhrefs -format list -recursive \
 sync://srvr2.ABCo.com:2647/Modules/Chip

{relpath CPU name CPU type Module version Gold url  \
 sync://srvr2.ABCo.com:2647/Modules/CPU hrefs  {{relpath /ROM name \
 ROM type Module version 1.4 url \
 sync://srvr2.ABCo.com:2647/Modules/ROM}}}

Note: The hierarchy of each submodule is contained within the hrefs
      property.
```

Example of Displaying the Hierarchical References in a Workspace

```
This example displays the hierarchical references for the Chip module
in text mode; specified by the module workspace instance name.

 dss> showhrefs -recursive Chip%0

 Beginning showhrefs operation ...
 Showing hrefs of module /home/rsmith/Modules/chip/Chip%0 ...

 /home/rsmith/MyModules/chip/Chip%0: Workspace version - 1.7
 /home/rsmith/MyModules/chip/Chip%0: Href mode - normal

 Name  Url                                Version  Type    Relative Path
 ------------------------------------------------------------------------
 CPU   sync://srvr2.ABCo.com:2647/Modules/CPU  1.2      Module  CPU

 ========================================================================

 Showing hrefs of module /home/rsmith/MyModules/chip/CPU/CPU%1 ...

 Name  Url                               Version  Type    Relative Path
 ---------------------------------------------------------------------
 ROM   sync://srvr2.ABCo.com:2647/Modules/ROM  1.4      Module  /ROM

 /home/rsmith/MyModules/chip/ROM/ROM%1: Workspace version - 1.2
 /home/rsmith/MyModules/chip/CPU/ROM/ROM%1: Module has no hierarchical
 references.

 Finished showhrefs operation.
```

Example of Showing Hrefs on the Workspace in List Format

```
This example displays hierarchical references specifying a workspace
module, Chip%0,  with report -normal output. This example uses the
hierarchy described at the beginning of the examples section.

dss> showhrefs -format list  -recursive Chip%0

{hrefs {{relpath /ROM name ROM instance_name ROM%1 version 1.4 type \
```

```
 Module basedir /home/rsmith/MyModules/chip/CPU/ROM url \
 sync://srvr2.ABCo.com:2647/Modules/ROM}} relpath CPU name CPU \
 instance_name CPU%1 version 1.2 type Module basedir \
 /home/rsmith/MyModules/chip/CPU url \
 sync://srvr2.ABCo.com:2647/Modules/CPU}
```

Example of Showing Hrefs in Brief Report mode and List Format

```
 This example shows the brief version of the report on the hrefs shown
 in Example 1 formatted for Tcl processing.

 dss> showhrefs -format list -report brief -recursive Chip%0

  name Chip url sync://srvr2.ABCo.com:2647/Modules/Chip hrefs {{name \
  CPU url {sync://srvr2.ABCo.com:2647/Modules/CPU;Gold} hrefs {{name \
  ROM url {sync://srvr2.ABCo.com:2647/Modules/ROM;1.4}}}}}
```

Example Showing Overridden Hrefs in the Workspace in Text Format

```
 This example shows overridden hierarchical references in the
 workspace.  It uses the following hierarchical structure:

 CHIP           <= uses the default configuration
   CPU          <= uses the Trunk:Latest version
     LIB        <= uses the Gold version
   ROM        <= uses the Trunk:Latest version
     LIB      <= uses the Trunk:Latest version (static version 1.2)

 dss> showhrefs -overridden -report verbose CHIP%0

 Beginning showhrefs operation ...

 Showing hrefs of module c:/Workspaces/chip/CHIP%0 ...

 c:/Workspaces/chip/CHIP%0: Workspace version - 1.4
 c:/Workspaces/chip/CHIP%0: Href mode - normal

 Name  Url                                                  Selector
 Static Version  Type    Relative Path  Overridden
 ---------------------------------------------------------------
 CPU   sync://serv1.ABCo.com:2647/Modules/ChipDesign/CPU        Trunk:    1.2
Module  ../cpu         no
 LIB   sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB  Gold      1.2
Module  ../lib         no
 ROM   sync://serv1.ABCo.com:2647/Modules/ChipDesign/ROM        Trunk:    1.2
Module  ../rom         no

  ========================================================================

 Showing hrefs of module c:/Workspaces/cpu/CPU%1 ...

 c:/Workspaces/cpu/CPU%1: Workspace version - 1.3
```

```
c:/Workspaces/cpu/CPU%1: Href mode - normal

Name  Url                                                    Selector
Static Version  Type    Relative Path  Overridden
-------------------------------------------------------------------
LIB   sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB Gold
1.2             Module  ../lib         yes

c:/Workspaces/lib/LIB%0: Workspace version - 1.2
c:/Workspaces/lib/LIB%0: Module has no hierarchical references.

 ============================================================================

Showing hrefs of module c:/Workspaces/rom/ROM%1 ...

c:/Workspaces/rom/ROM%1: Workspace version - 1.3
c:/Workspaces/rom/ROM%1: Href mode - normal

Name  Url                                                    Selector
Static Version  Type    Relative Path  Overridden
-------------------------------------------------------------------------
LIB   sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB Trunk:
1.2            Module  ../lib         yes


Finished showhrefs operation.
```

Example Showing Overridden Hrefs in List Format

```
 This example shows overridden hierarchical references in the
workspace.  It uses the same hierarchical structure as the previous
example.

dss> showhrefs -overridden -report verbose -format list CHIP%0
{hrefs {{relpath ../LIB resolved_selector 1.2 name LIB selector Gold
type Module overridden no version 1.2 url
sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB}} relpath ../COM
resolved_selector 1.3 name COM selector Trunk: type Module overridden
no version 1.2 url sync://serv1.ABCo.com:2647/Modules/ChipDesign/COM}
{relpath ../LIB resolved_selector 1.2 name LIB selector Gold type
Module overridden no version 1.2 url
sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB} {hrefs {{relpath
../LIB resolved_selector 1.2 name LIB selector Trunk: type Module
overridden no version 1.2 url
sync://serv1.ABCo.com:2647/Modules/ChipDesign/Tools/LIB}} relpath ../ROM
resolved_selector 1.3 name ROM selector Trunk: type Module overridden
no version 1.2 url sync://serv1.ABCo.com:2647/Modules/ChipDesign/ROM} #
```

Example Showing Conflicting Hierarchical References in the Workspace

```
This example shows hierarchical conflicts in the workspace in normal
```

```
reporting mode.

dss> showhrefs -conflict TOP%0

 Beginning showhrefs operation ...

 Showing hrefs of module c:/chip/top/TOP%0 ...

 c:/chip/top/TOP%0: Workspace version - 1.4 : Selector - Trunk:
 c:/chip/top/TOP%0: Href mode - normal

 Name  Url                                                    Selector
 Static Version  Type    Relative Path
 ------------------------------------------------------------------------
 COM   sync://serv1.ABCo.com:2647/Modules/Chip-P21z/COM        Trunk:
 1.2             Module  ../COM
 LIB   sync://serv1.ABCo.com:2647/Modules/Chip-P21z/Tools/LIB  Gold
 1.2             Module  ../LIB
 ROM   sync://serv1.ABCo.com:2647/Modules/Chip-P21z/ROM        Trunk:
 1.2             Module  ../ROM

 LIB: Not present in workspace due to hierarchical conflict.
 Finished showhrefs operation.
```

showmcache

showmcache Command

NAME

```
showmcache          - Lists the modules in one or more module caches
```

DESCRIPTION

- Notes for Modules
- Notes for Legacy Modules
- Understanding the Output for Modules Objects
- Understanding the Output for Legacy Modules

```
This command lists the contents of one or more specified module and
legacy module caches.

Module caches to be listed can be specified with the -mcachepaths
option. If the -mcachepath option is not supplied, the command
uses the path list specified in the default module cache paths
registry setting.

The command searches the module caches in the order specified with
```

```
the -mcachepaths option, or previously defined with SyncAdmin.

This command supports the command defaults system.
```

Notes for Modules

```
For module mcache, the showmcache command lists all of the modules
found within the directory hierarchy of the mcache regardless of
where their base directories reside.

Note: The module cache directory must be a workspace root
      directory.
```

Notes for Legacy Modules

```
For legacy module mcache, the showmcache command lists all of the
releases whose base directory resides in the top-level directory
of the mcache.
```

Understanding the Output for Modules Objects

```
By default, or if you run the showmcache command with the '-format
text' option, the command displays a list of module cache paths
searched, followed by a table for modules.
The content of the table is displayed in order of module cache paths
searched and entries for each module cache are sorted by PATH.

 The table for modules includes the following information as columns:

o PATH              The absolute path of the module version base
                    directory.

o URL               The URL of the module.

o SELECTOR          The selector used to fetch the module.

o VERSION           The version number of the module.

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 release is available
for
                    use.
                    A module might be unavailable if, for example,
                    it is currently being fetched to the mcache.

o HIERARCHY         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.

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.

If you run the showmcache command with '-format list', it returns a Tcl list of modules and releases.

Each module in the list has the following properties:

o url — The URL of the module

o path — The base directory path of the module version.

o version — The version number of the module.

o selector — The selector used to fetch the module.

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.

o available — Indicates whether the module is available for use by the populate command.

o hrefmode — Indicates which href mode was used to fetch the module.

Note: The populate command will not create an mcache link to an mcached module version that was not fetched statically.

Understanding the Output for Legacy Modules

By default, or if you run the showmcache command with the '-format text' option, the command displays a list of module cache paths searched, followed by a table for modules, if there are any, and

table for releases. The content of the module cache table for modules is discussed in the previous section. The legacy module release table is displayed in order of module cache paths searched and entries for each module cache are sorted by PATH.

Note: The list output for module and legacy module mcache entries is different. If you are running an environment that contains both, you should the modules section as well.

The tables for releases includes the following information as columns:

- o PATH — The absolute path of the module version base directory.
- o TARGET — The URL of the release.
- o AVAILABLE — Indicates whether the legacy release is available for use by the populate command. Possible values are 1 and 0, where a value of 1 indicates the release is available for use.
  A legacy release might be unavailable if, for example, it is currently being fetched to the mcache.
- o HIERARCHY — Indicates whether the legacy release was recursively populated into the module mcache. Possible values are 'yes' and 'no', where 'yes' indicates that the legacy release was recursively populated.

If you run the showmcache command with '-format list', it returns a Tcl list of releases.

Each legacy release in the list has the following properties:

- o target — The URL of the release.
- o path — The base directory path of the release.
- o type — The type of configuration.
  Note: The type is always "Release" because the show mcache command only lists releases in legacy mcaches.
- o selector — Always empty.
- o hierarchical — Indicates whether the legacy release was recursively populated into the module mcache. Possible values are 'yes' and 'no', where 'yes' indicates that the legacy release was recursively populated.
- o available — Indicates whether the legacy release is available for use by the populate command.

SYNOPSIS

```
showmcache [-format (text | list)] [-mcachepaths <path_list>]
```

OPTIONS

- -format
- -mcachepaths

-format

```
-format text|list      Indicates the format in which the output
                       appears.

                       Valid values are:

                      o text - Displays the output in a table
                        format. This is the default behavior.

                      o list - Returns a Tcl list of name/value
                        pairs.

                      For information about the values displayed,
                      see the "Understanding the Output" section.
```

-mcachepaths

```
-mcachepaths <path_list>
                     Identifies one or more module caches to
                     search.

                     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"

                     If -mcachepaths is absent, the command uses
                     the list of paths defined in the registry
                     setting.

                     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 searches the module caches in
                       the order specified with the -mcachepaths
                       option or in the default module cache paths
```

registry setting if this option is absent.

RETURN VALUE

If you run the showmcache command with the '-format list' option, it returns a Tcl list of modules. Each list element contains a list of name/value pairs representing data about the module. If you specify '-format text' or if you do not specify the -format option, the command does not return any Tcl values. For a description of the output, see the "Understanding the Output" section.

SEE ALSO

populate, swap show, command defaults

EXAMPLES

This example shows in table format the releases available in two module caches, /A5/cache and /B1/cache:

```
 dss> showmcache -mcachepaths "/A5/cache /B1/cache" -format text

Mcachepaths search order:

/A5/cache/CPU
/B1/cache/ALU

Modules found:

PATH           TARGET                                    AVAILABLE  HIERARCHY
-------------------------------------------------------------------------
/A5/cache/CPU  sync://cpu.ABCo.com:2647/Projects/Cpu@Rel1   yes    yes
/A5/cache/SLIB sync://svr1.ABCo.com:2647/Projects/STDLIB@R1 yes    no
/B1/cache/ALU  sync://alu.ABCo.com:2647/Projects/Alu@Rel2   yes    yes
```

==========

This example shows a module cache and a legacy module cache available.

```
 stcl>  showmcache -mcachepaths {/home/dana/designs/components
        /home/dana/designs/libraries}
Mcachepaths search order:

components
libraries

Legacy Configurations found:

 PATH                TARGET                    AVAILABLE      HIERARCHY
```

```
------------------------------------------------------------------------
 home/dana/designs   sync://qewfsun8:30084/        yes           yes
  /libraries/stdlib    Projects/stdlib@REL1


 Modules found:

   PATH              URL              SELECTOR            VERSION\
 ------------------------------------------------------------------------
           HREFMOD                                AVAILABLE         HIERARCHY


 components/CPU      sync://qewfsun8:30084//Modules/CPU     GOLD     1.2\

              static                                        yes       yes
```

showmods

showmods Command

NAME

```
showmods            - Displays the modules available on a server or
                      workspace
```

DESCRIPTION

- Understanding the Output

```
This command lists the available modules and external modules within
a workspace or on a specified server, and legacy modules on a
specified server.

If the command is run within a module directory structure, the
showmods command includes the containing modules.

Note: Legacy modules in a workspace do not display with showmods.

The showmods command identifies mcache links to modules within the
workspace.

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

The output of the command depends on whether you specify a module on the server or a folder in your workspace.

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.

By default, or if you run the showmods command with the '-report normal' option, the following information is output according to the type of argument being processed:

When the argument is a server module, showmods displays the following information:

Column Titles	Property Names	Description
Name	name	Name of the module. Note: Modules are displayed alphabetically by name.
Owner	owner	User name of the person who created the module.
Path	path	The vault directory containing the contents of the module.

When the argument is a workspace folder, showmods displays the following information:

Column Titles	Property Names	Description
Name	name	Name of the module. Note: Modules are displayed alphabetically by name.
Instance	modinstname	Unique instance name of the module in the workspace.
Base Directory	basedir	Workspace directory containing the contents of the module.
Url	url	Location of the module on the server. Note: For external modules, the URL is sync:///ExternalModule/<external-type>/<external-data>
Selector	selector	Selector used to determine which version to fetch into a workspace. For more information, see the selectors help topic.
Mcache Path	mcachelink	Location of the module cache directory containing the module. Note: If you run the showmods commands with -format text (default) when there are mcaches present in the workspace, the mcache link information: name, instance name and mcache path, display in a

```
                            separate section below the module
                            information.

If you run the showmods command with '-report brief', it displays the
following information.

o For server modules - module path on the server,
  sorted by path. The column title is Module Path.  The property name
  is path.
o For workspace modules - full directory path for the workspace
  folder, sorted by path. The column title is Unique Address.  The
  property name is address.

If you run the showmods command with '-report verbose', it displays
the information shown with the -normal option and the following
additional information:

For a server module:
Column       Property
Titles       Names          Description
------       -----          ------------
Comment      comment        Comment used when creating a module.

Url          url            Full sync URL of the module.
                            Note: For external modules, the URL is
                    sync:///ExternalModule/<external-type>/<external-data>

Type         type           The type of module being viewed.
                            o standard - a regular module.
                            o legacy module - a legacy module.
                            o external - an external module
                              containing files managed by a different
                              change management system.

For a workspace module:
Column       Property
Titles       Names          Description
------       -----          ------------
Unique
Address      address        Full module address on the client side.
Version      version        The version information for the module
                            version in the workspace.

Top          toplevel       Denotes whether the module is a top-level
                            module (meaning it has no other modules
                            in the workspace containing an href to
                            the reported module). A value of "yes"
                            (or "1") means the module is a
                            top-level. A value of "no" (or "0")
                            means it is not a top-level module.

If you run the showmods command with '-report script', it displays
the same properties as the verbose report in '-format list' mode.

Note: The '-report script' mode is only applicable for workspace arguments.
```

SYNOPSIS

```
showmods [-[no]all] [-format [{text | list}] [-filter <string>]
         [-report {brief | normal | verbose | script}] [-[no]top]
         [<argument>]
```

ARGUMENTS

- Server Path
- Workspace Folder

Server Path

<server path>	A server path. You can use wildcard characters in the path for any piece of the address except the servername and port number. If the path isn't specified, the command will return information for all modules, including legacy modules, on the server.

Workspace Folder

<workspace folder>	A workspace folder. Specifying a workspace folder displays all modules with a base directory at or below the specified folder. In addition, if the folder is a member of a module whose base directory is above the folder, then that module is also reported. You can use wildcard characters for any part of the workspace folder name. Note: This will never display any legacy modules, since they do not have metadata in the root directory.

Note: If no argument is specified, showmods command uses the current directory.

OPTIONS

- -[no]all
- -format
- -filter
- -report
- -[no]top

-[no]all

```
-[no]all
                        Determines whether to report on the specified
                        workspace folder or on all modules in the
                        workspace using the workspace root directory of
                        the specified workspace folder.

                        -noall reports on the specified workspace
                         folder. (Default)

                       -all begins at the workspace root directory of the
                        workspace and reports all the modules found in
                        the workspace. Given that the workspace root is
                        usually defined in a directory path one or
                        more levels higher than the argument given to
                        the showmods command, the -all option may list
                        modules that are outside of the directory cone
                        below the argument's path.

                        This option is only valid when working with workspace
                        folder arguments.
```

-format

```
 -format <mode>         Determines the format of the output. For
                        information about the information displayed,
                        and the output sort order, see the
                        "Understanding the Output" section.
                        Valid values are:
                        o list -  Displays a list with the following
                          format:
                               {
                                name <name>

                               }

                          Note: This option replaces the deprecated
                          -report script option.

                        o text - Display a text table with headers and
                          columns. (Default)
```

-filter

```
 -filter <string>       Specify one or more extended glob-style
                        expressions to identify an exact subset of
                        module objects on which to operate.

                        The -filter option takes a list of expressions
                        separated by commas, for example: -filter
```

```
+.../ProjectA/../*,-.../RAM*

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.

For this command, the expressions are matched
against the full module URL for each module
(sync://host:1234/Modules/MyMod)

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, "...
/ProjectA/.../Rom*" matches Rom* modules in a
URL that contains "ProjectA", followed by zero
or more levels. 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.
```

-report

```
-report <mode>          Determines what the output of the command is.
                        For more information on the output of the
                        -report option, see the "Understanding the
                        Output" section.

                        Valid values are:

                        o brief - Displays path or workspace location
                          for the specified server or workspace module.

                        o normal - Displays basic module information
                          for the specified server or workspace
                          module. (Default)

                        o verbose - Displays extended module
                          information for the specified server or
                          workspace module.

                        o script - Displays the extended module
```

```
                          information for the specified workpace
                          module in a TCL list.

                          Note: When -script mode is used, The format
                          mode is automatically set to -list.
```

-[no]top

```
-[no]top                  Determines whether to display output for all
                          modules or only the top-level modules, modules
                          with no other modules in the workspace
                          containing an href to the reported module.

                          -notop Does not filter the command output and
                          displays all module information. (Defaut)

                          -top Filters the command output to only list
                           top-level modules.

                          This option is only valid when working with
                          workspace folder arguments.
```

RETURN VALUE

```
 If you run the showmods command with the '-format list' option, it
 returns a Tcl list. For a description of the output, see the
 "Understanding the Output" section.  If the command fails, it returns
 a Tcl error.
```

SEE ALSO

```
 swap show, mkmod, command defaults
,
```

EXAMPLES

- Example of Showing the Modules on the Server in Text Format
- Example of Showing the Modules on the Server in List Format
- Example Showing the Server Modules Using Verbose Report Mode
- Example Showing a TCL List of Server Modules Using Verbose Report Mode

```
 These examples list the modules available on the DesignSync server
 sync://srvr1.ABCo.com:2647. The examples show the showmods command
 running with the two report modes (normal and verbose) and both
 format options (text and list) to show the different output.
```

Example of Showing the Modules on the Server in Text Format

```
The following two examples display to the screen in a user-friendly
format the modules available on the DesignSync server
sync://srvr1.ABCo.com:2647.  The first example specifies a server
module.  The second example specifies a workspace folder.  The
workspace folder contains referenced modules, ROM and CPU and mcache
module Tools.

dss> showmods sync://srvr1.ABCo.com:2647/Modules/Chip

 Beginning showmods operation ...
 Name  Owner    Path
 --------------------------
 Chip  rsmith  Modules/Chip

 Finished showmods operation.

dss> showmods ~/MyModules
 Beginning showmods operation ...

 Name  Instance  Base Directory
     Url                                  Selector
 ---------------------------------------------
 ALU   ALU%0     /home/rsmith/MyModules/alu
     sync://srvr1.ABCo.com:2647/Modules/ALU   Trunk:
 Chip  Chip%0    /home/rsmith/MyModules/chip
     sync://srvr1.ABCo.com:2647/Modules/Chip  Trunk:
 Chip  Chip%1    /home/rsmith/MyModules/chipGold
     sync://srvr1.ABCo.com:2647/Modules/Chip  Gold:
 CPU   CPU%1     /home/rsmith/MyModules/chip/CPU
     sync://srvr1.ABCo.com:2647/Modules/CPU   Gold
 ROM   ROM%1     /home/rsmith/MyModules/chip/CPU/ROM
     sync://srvr1.ABCo.com:2647/Modules/ROM   Gold:
 SPC   SPC%0     /home/rsmith/MyModules/spc
     sync://srvr1.ABCo.com:2647/Modules/SPC   Trunk:
 300MM 300MM%0  /home/rsmith/MyModules/300mm
     sync://srv1.ABCo.com:2647/Modules/300MM  Trunk:

 MCACHE LINKS

 Name       Instance     Mcache Path
-------------------------------------------------------------------------
300MM      Chip300MM%0   /home/mcacheDir/300mm/300MM%0


 Finished showmods operation.
```

Example of Showing the Modules on the Server in List Format

```
The following two examples display to the screen in Tcl format the
modules available on the DesignSync server
sync://srvr1.ABCo.com:2647.  The first example specifies a server
```

```
module.  The second example specifies a workspace folder.  The
workspace folder contains referenced modules, ROM and CPU.

dss> showmods -format list sync://srvr1.ABCo.com:2647/Modules/Chip
  {name Chip owner rsmith path Modules/Chip}

dss> showmods -format list ~/MyModules
  {name ALU modinstname ALU%0 basedir /home/rsmith/MyModules/alu url
  sync://srvr1.ABCo.com:2647/Modules/ALU selector Trunk:} {name Chip
  modinstname Chip%0 basedir /home/rsmith/MyModules/chip url
  sync://srvr1.ABCo.com:2647/Modules/Chip selector Trunk:} {name Chip
  modinstname Chip%1 basedir /home/rsmith/MyModules/chipGold url
  sync://srvr1.ABCo.com:2647/Modules/Chip selector Gold:} {name CPU
  modinstname CPU%1 basedir /home/rsmith/MyModules/chip/CPU url
  sync://srvr1.ABCo.com:2647/Modules/CPU selector Gold} {name ROM
  modinstname ROM%1 basedir /home/rsmith/MyModules/chip/CPU/ROM url
  sync://srvr1.ABCo.com:2647/Modules/ROM selector Gold:} {name SPC
  modinstname SPC%0 basedir /home/rsmith/MyModules/spc url
  sync://srvr1.ABCo.com:2647/Modules/SPC selector Trunk:}
  {name 300MM modinstname Chip300MM%0 basedir
  /home/rsmith/MyModules/300mm url
  sync://srv1.ABCo.com:2647/Modules/300MM selector Trunk: mcachelink
  /home/mcacheDir/300mm/300MM%0}
```

Example Showing the Server Modules Using Verbose Report Mode

```
The following two examples display to the screen in a user-friendly
format the modules available on the DesignSync server
sync://srvr1.ABCo.com:2647.  The first example specifies a server
module.  The second example specifies a workspace folder.  The
workspace folder contains referenced modules, ROM and CPU.

dss> showmods -report verbose sync://srvr1.ABCo.com:2647/Modules/Chip
 Beginning showmods operation ...

 Name  Type      Owner    Path
     URL                                  Comment
 ---------------------------------------------------------------
 Chip  standard  rsmith  Modules/Chip
     sync://srvr1.ABCo.com:2647/Modules/Chip     Chip design module

 Finished showmods operation.

dss> showmods -report verbose ~/MyModules
 Beginning showmods operation ...

 Name  Instance  Base Directory
     Unique Address
     Url                                Version  Selector
 ------------------------------------------------------------------
 ALU   ALU%0     /home/rsmith/MyModules/alu
     /home/rsmith/MyModules/alu/ALU%0
     sync://srvr1.ABCo.com:2647/Modules/ALU   1.4      Trunk:
 Chip  Chip%0    /home/rsmith/MyModules/chip
```

```
     /home/rsmith/MyModules/chip/Chip%0
     sync://srvr1.ABCo.com:2647/Modules/Chip  1.5      Trunk:
 Chip  Chip%1    /home/rsmith/MyModules/chipGold
     /home/rsmith/MyModules/chipGold/Chip%1
     sync://srvr1.ABCo.com:2647/Modules/Chip  1.2.1.1  Gold:
 CPU   CPU%1     /home/rsmith/MyModules/chip/CPU
     /home/rsmith/MyModules/chip/CPU/CPU%1
     sync://srvr1.ABCo.com:2647/Modules/CPU   1.3      Gold
 ROM   ROM%1     /home/rsmith/MyModules/chip/CPU/ROM
     /home/rsmith/MyModules/chip/CPU/ROM/ROM%1
     sync://srvr1.ABCo.com:2647/Modules/ROM   1.4      Gold:
 SPC   SPC%0     /home/rsmith/MyModules/spc
     /home/rsmith/MyModules/spc/SPC%0
     sync://srvr1.ABCo.com:2647/Modules/SPC   1.4      Trunk:
 300MM 300MM%0  /home/rsmith/MyModules/300mm
     /home/rsmith/MyModules/300mm/300MM%0
     sync://srv1.ABCo.com:2647/Modules/300MM  Trunk:

 MCACHE LINKS

 Name       Instance     Mcache Path
-------------------------------------------------------------------------
300MM      Chip300MM%0   /home/mcacheDir/300mm/300MM%0


 Finished showmods operation.
```

Example Showing a TCL List of Server Modules Using Verbose Report Mode

```
The following two examples display to the screen in Tcl format the
modules available on the DesignSync server
sync://srvr1.ABCo.com:2647.  The first example specifies a server
module.  The second example specifies a workspace folder.  The
workspace folder contains referenced modules, ROM and CPU.

Note: The results of the second command are identical to specifying
'showmods -report script.'


dss> showmods -format list -report verbose \
sync://srvr1.ABCo.com:2647/Modules/Chip
  {name Chip type standard owner rsmith path Modules/Chip url
  sync://srvr1.ABCo.com:2647/Modules/Chip comment {}}

dss> showmods -report verbose -format list ~/MyModules
  {name ALU modinstname ALU%0 basedir /home/rsmith/MyModules/alu
  address /home/rsmith/MyModules/alu/ALU%0 url
  sync://srvr1.ABCo.com:2647/Modules/ALU version 1.4 selector Trunk:}
  {name Chip modinstname Chip%0 basedir /home/rsmith/MyModules/chip
  address /home/rsmith/MyModules/chip/Chip%0 url
  sync://srvr1.ABCo.com:2647/Modules/Chip version 1.5 selector
  Trunk:} {name Chip modinstname Chip%1 basedir
  /home/rsmith/MyModules/chipGold address
  /home/rsmith/MyModules/chipGold/Chip%1 url
```

```
sync://srvr1.ABCo.com:2647/Modules/Chip version 1.2.1.1 selector
Gold:} {name CPU modinstname CPU%1 basedir
/home/rsmith/MyModules/chip/CPU address
/home/rsmith/MyModules/chip/CPU/CPU%1 url
sync://srvr1.ABCo.com:2647/Modules/CPU version 1.3 selector Gold}
{name ROM modinstname ROM%1 basedir
/home/rsmith/MyModules/chip/CPU/ROM address
/home/rsmith/MyModules/chip/CPU/ROM/ROM%1 url
sync://srvr1.ABCo.com:2647/Modules/ROM version 1.4 selector Gold:}
{name SPC modinstname SPC%0 basedir /home/rsmith/MyModules/spc
address /home/rsmith/MyModules/spc/SPC%0 url
sync://srvr1.ABCo.com:2647/Modules/SPC version 1.4 selector Trunk:}
{name 300MM modinstname Chip300MM%0 basedir
/home/rsmith/MyModules/300mm address
/home/rsmith/MyModules/300mm/Chip300MM%0 url
sync://srv1.ABCo.com:2647/Modules/300MM version 1.3 selector Trunk:
mcachelink /home/mcacheDir/300mm/300MM%0}
```

showstatus

showstatus Command

NAME

```
showstatus          - Displays the status of a module in your
                      workspace
```

DESCRIPTION

- Understanding the Output
- Text Formatted Output
- List Formatted Output
- External Module Support
- Legacy Module Output

```
This command lists the status of the hierarchical references of a
module in your local work area as compared to that
module on the server. The main status changes shown by this command
include:
      o Selector changes
      o Added or removed hierarchical references
      o Hierarchical reference conflicts
      o Added, moved, or removed module members
      o Swapped module

Using this command, you can verify that your workspace is up-to-date.
By default the showstatus command does not show the status of module
members. In order to show object status, you must specify the
-objects option.
```

Notes:
 * When the persistent hrefmode of the workspace is normal, showstatus uses the setting of the "Change traversal mode with static selector on top level module" option in SyncAdmin (registry key "HrefModeChangeWithTopStaticSelector") to determine how the module hierarchy should be understood by the command. If the setting is enabled and the top-level module is populated with a static selector, then the modules populated in the workspace must match the expected static versions in order to be considered up-to-date.

 * If a module in the workpace is swapped, the showstatus commands reports the status of the swapped module as "up-to-date," and indicates that the module has been swapped.

 * 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.

 * When working with mcache links in older clients, showstatus may report the module as out of date.

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

The output can be restricted using the -report option. The -report brief and -report summary options provide summary information on the workspace status, while the -report normal option provides more detail. The -report verbose mode provides additional information about whether the hierarchical references need updating.

Note: You can set the -report normal mode to report on the "needs update" status of hierarchical references with the ShowHrefsNeedCheckinStatus registry key. For more information on setting the registry key, see the DesignSync Administrator's Guide.

The output can be formatted for easy viewing (-format text) or optimized for Tcl processing (-format list). Both formats show similar information, but the forms are very different. The following sections, "Text Formatted Output" and "List Formatted Output," provide specific information about the information returned by the showstatus command.

The module status can change depending on the options specified with the command. For more information, see the output sections below or the options definitions.

Note: For information on legacy module output, see the "Legacy Modules output" section.

Text Formatted Output

The showstatus output is formatted into different sections for ease of use:

- The module information section provides the sync URL of the server modules and the base directory of the workspace module. This section does not appear when -report summary or brief modes are specified.

- The version status section displays the version information for both the workspace and server modules and the unique identifiers (UID) for the modules, if they are different. The -report summary mode displays a single line indicating whether the versions are the same or different. The -report brief mode displays a single line if the version are the same, or the version information if the versions are different.

- The hierarchical reference section compares the hrefs in the local metadata with the appropriate server-side module designated by the selector found in the workspace. The -report summary mode displays a single line notice indicating whether the hrefs are the same or different. The -report brief mode displays a single line if the hrefs are the same, or the list of differences if they are different. This section does not appear at all if the -nohrefs option is specified.

- The href conflicts section compares the expected hrefs (those stored with the parent module) with the actual contents of the workspace. If the URL, selector, or static version doesn't match, the parent module shows a status of "Needs Checkin." For any swapped modules, a line appears for the referenced module providing information about the swapped module version. This section does not appear in -report brief or summary modes or if the -nohrefs option is specified.

- The hrefs missing section highlights any referenced objects which are not present in the workspace. This section does not appear in -report brief or summary modes or if the -nohrefs option is specified.

- The contents section displays the results of comparing the objects in the workspace to the objects in the server. This section does not appear in -report summary mode or if the -noobjects option is specified. If -report brief mode is specified, the command uses the -report brief form of compare.

  Note: When you run showstatus on a DesignSync vault, this is the only section displayed.

- The hierarchical status section recursively reports the status for all hierarchical references in the workspace. This section does not appear if the -norecursive option is specified.

At the end of the showstatus output, the command displays an overall status of the workspace and a recommended action, if an action is needed.

Status	Description
Up-to-date	The workspace and the server versions are identical.
Out-of-date	There is at least one discrepancy between the version present in the workspace and the version on the server.
Unknown	The status could not be determined. This may happen if a server is unreachable.

Recommended Action	Solution
Needs update	The workspace version needs to be repopulated to synchronize it with the server.
Needs checkin	The workspace contains information that has not been captured in a version on the server. Check in your changes, to update the server version.

List Formatted Output

When showstatus is used with the -format list option, the output is returned in the form of a tcl list describing the objects and their status. The properties returned for each object depend on the type of object being examined and the options specified on the command line.

Note: If the object doesn't exist (exist is 0) then most of the values return a null ("") value.

Note: This table lists the properties in alphabetical order, not necessarily the output order of the command.

Property Names	Description
actual	Property list for the object in a workspace. The property list consists of the following properties: o url o selector o version o uid
basedir	Absolute path of the base directory of the object.
conflicts	List of differences between the list of expected hrefs in the workspace and the actual workspace content.

content — Results of running a compare command on the contents of the workspace and the server. This report only appears when the -objects option is specified.

content_status — Status of the objects in the module or DesignSync vault.
- o Up-to-date - Objects contained in the workspace and server versions are synchronized.
- o Out-of-date - There are differences in the objects listed in the workspace and server versions. These differences include different object versions and added or removed objects.

exists — Indicates whether the object exists in the workspace. Possible values include:
- o 1 - object exists in the workspace.
- o 0 - object does not exist in the workspace.

fullname — Full, unique workspace address.

hier_status — Overall status of the referenced objects.
- o Up-to-date - Workspace and server versions are synchronized.
- o Out-of-date - Workspace and server versions are different.

hierarchy — An array of the referenced objects present in the workspace, indexed by href name.
- o name - Href name.
- o type - type of object referenced. Possible values include Module, Branch, Selector, External, Release, Alias, Deliverable, or Vault.
- o url - Server-side vault of the referenced object.
- o selector - selector for the referenced object.
- o version - if the reference object is a module, the numeric version ID. For any other object, a null ("") value.
- o relpath - relative path of the referenced object.
- o status - the status of the referenced object. Possible values include:
  - Up-to-date where the workspace and server metadata for the href instance match.
  - Out-of-data where a difference exists among one or more of the values in the hierarchy array. The notes section in the array explains the difference.
  - Local-only where an href with the href instance name is present only in the workspace metadata.
  - Server-only where an href with the href instance name is present only in the server metadata.
- o notes - List of strings describing the discrepancies mentioned in the status value. Possible values include:
  - "Relative path changed on server to <relpath>"
  - "Url changed on server to <URL>"
  - "Selector changed on server to <selector>"

```
                - "Version changed on server to <version>"

              Note: In -report brief mode, this property only lists
              out-of-data hrefs. This property does not appear in
              -report summary mode or when the -norecursive option
              is specified.

hrefs          Combined array of hrefs found on the workspace and on
               the server. This property does not appear in -report
               summary mode or when the -nohrefs option is
               specified.  The array contains the following fields:
               o name - Href name.
               o type - type of object referenced.  Possible values
                 include Module, Branch, Selector, Release, Alias,
                 Deliverable, or Vault.
               o url - Server-side vault of the referenced object.
               o selector - selector for the referenced object.
               o version - if the reference object is a module, the
                 numeric version ID. For any other object, a null
                 ("") value.
               o relpath - relative path of the referenced object.
               o status - the status of the referenced object.
                 Possible values include:
                 - Up-to-date where the workspace and server
                   metadata for the href instance match.
                 - Out-of-data where a difference exists among one
                   or more of the values in the hierarchy array.
                   The notes section in the array explains the
                   difference.
                 - Local-only where an href with the href instance
                   name is present only in the workspace metadata.
                 - Server-only where an href with the href instance
                   name is present only in the server metadata.
               o  notes - List of strings describing the
                  discrepancies mentioned in the status
                  value. Possible values include:
                  - "Relative path changed on server to <relpath>"
                  - "Url changed on server to <URL>"
                  - "Selector changed on server to <selector>"
                  - "Version changed on server to <version>"

href_status    Status of the hierarchical reference metadata in the
               workspace.
               o Up-to-date - Workspace and server versions are
                 synchronized.
               o Out-of-date - Workspace and server versions are
                 different.

legacy_status  Legacy show status for legacy configurations.  For
               more information, see the "Legacy module output"
               section.

missing        Names of the hrefs expected in the workspace that
               are not present in the workspace. This property does
               not appear in -report brief or summary modes.

               Note: Missing hrefs do not automatically mean the
```

workspace and server are out of sync. An href may have been filtered out during the workspace populate, or removed from the workspace manually.

modinstname — Workspace module instance name.

needs_checkin — Status of the object in the workspace:
0 Indicates that the object is up-to-date and does not require a checkin.
1 Indicates the object is locally modified and does require a checkin.

needs_update — Status of the object in the workspace:
0 Indicates that the object is up-to-date and does not need to updated by the server.

1 Indicates that the object is out-of-date and does need to be updated by the server.

server — Property list for the object on a server which which the workspace object is being compared. These are the properties of the module or DesignSync vault to which the workspace selector resolves.
- o url
- o selector
- o version
- o uid

status — Status of the workspace object.
- o Up-to-date where the workspace and server versions are synchronized.
- o Out-of-date where the workspace and server versions are different.
- o Unknown where the status of the object could not be determined. This might occur if a server is unavailable.

swap_conflict — Shows the properties of the swapped modules, including the href name, selector and version for each swapped module version.

swapped — Indicates whether the href is swapped.
- o 1 - object is swapped.
- o 0 - object is not swapped.

type — Workspace object type. Possible values include:
- o standard - module
- o legacy module
- o external - external module
  Note: External modules are always considered "up-to-date."
- o DS vault
- o deliverable - IP Gear deliverable.

uid — UID of the module in the workspace.

unknown — Status of the object hierarchy in the external module.

```
                0 Indicates that the external module hierarchy status
                  is known.
                1 Indicates that the external module hierarchy status
                  is not known.


version_status  Status of the workspace version.
                o Up-to-date - Workspace version and server version
                  associated with the workspace by the designated
                  select are synchronized.
                o Out-of-date - Workspace version and server version
                  associated with the workspace by the designated
                  selector are different.
```

External Module Support

DesignSync supports showing the status of an external module to determine whether the objects are current or out of date. After an external module has been populated, the showstatus command can be available to query the status of the external module members and return the results.

For information on populating an external module, see the populate command. For information on configuring showstatus for external modules, see the DesignSync Administrator's Guide.

Legacy Module Output

The legacy module output is unchanged from previous versions.

The showstatus command, by default, displays the following information:

- Configuration: Identifies the configuration for which the status is shown. This value is a Synchronicity URL.

- Base Directory: Identifies the local file system directory in which the configuration resides. Each module in a hierarchy has its own base directory.

- Information about each hierarchical reference:
  - STATUS — The status of the hierarchical reference of the configuration in the work area as compared to the server. Possible values are:
    - Local Only — Indicates the hierarchical reference exists only in the local work area.
    - Out-of-date — Indicates that the hierarchical reference in the local work area does not match the hierarchical reference on the server; for example, an alias on the server may have changed to refer to a new release or the relative path of the hierarchical reference on the server may have changed.

Displays a table of conflicts if conflicts exist between the submodule configuration that a parent module expects to find in the workspace and the submodule configuration that actually exists there. Such conflicts can be caused by: a relative path that contains a configuration different from the one that the parent configuration expects; a relative path that contains no configuration; or a relative path that doesn't exist.

Server Only — Indicates the hierarchical reference was added on the server.

Unknown — Indicates that status of the hierarchical reference cannot be determined. This status is displayed only if you specify a recursive showstatus operation. For example, if you specify 'showstatus -recursive' and the server on which a referenced configuration resides is unavailable, the showstatus operation lists the status for that configuration as Unknown.

Up-to-date — Indicates that the hierarchical reference in the local work area matches the hierarchical reference on the server.

- HREF — Identifies the submodule to which the configuration of the upper-level module is connected. This value is a Synchronicity URL.
- RELATIVE PATH — Identifies the path from the upper-level module to the submodule.

o Configuration status

Indicates the status of the configuration. (Displayed only when you specify the -recursive option.)

o Summary

Indicates the overall status of the configuration. By default (or if you specify the '-report normal' option), this value is a summary of the status of the configuration's hierarchical references. Note: If you specify the -objects option, this value represents the status of the objects contained in the configuration in combination with the status of its hierarchical references.

Possible values are:

- Local Only — Indicates the configuration exists only in the local work area.
- Out-of-date — Indicates that the configuration in the local work area does not match the configuration on the server.
- Server Only — Indicates the configuration was added on the server.
- Unknown — Indicates that status of the configuration cannot be determined. This status is displayed, for example, if the server on which the configuration resides is unavailable.
- Up-to-date — Indicates that the configuration in the local work area matches the hierarchical

reference on the server.

To show the status of the objects contained in your work area configuration (as compared to objects in the configuration on the server), you can use the showstatus command with the -objects option. Output from the command first shows the status of the configuration's hierarchical references (as described above) and then shows the status of its objects. When '-objects' is specified, the value for configuration status reflects the status of its objects in combination with the status of its hierarchical references.

Information for each object includes:

o Workspace Version — Identifies the version of the object in the work area configuration. "Unmanaged" indicates the object is not managed by DesignSync; "Unknown" indicates that the status cannot be determined. (For example, the command might display "Unknown" for an object if the server is not available.) If no information is displayed, it indicates that the version is absent from the work area.

o Configuration Version — Identifies the version of the object in the configuration on the SyncServer.

o Object Name — Identifies the name of the object for which status information is given.

If you use the showstatus command with '-format list' option, it returns a Tcl list in the following form:

```
target <module_URL>
relpath <relative_path>
[notes {
         "Old aliased release: <release_name>" |
         "New aliased release: <release_name>" |
         "Relative path changed to '<path>'" |
         "Cannot determine current value of alias on server." |
         <miscellaneous other information>
       }
]
status <Local Only | Out-of-date | Server Only | Unknown | Up-to-date>
hierstatus <Local Only | Out-of-date | Server Only | Unknown | Up-to-date>
[hrefs {{<submodule_status>} {...}}] ...
```

The returned information includes the following:

o target — The URL of the module configuration.

o relpath — The relative path from the base directory of the upper-level module configuration (Parent) to the submodule configuration (Target). This path is used when you fetch (populate) the module into your work area.

o status — The status of the configuration in your work

```
                    area (as compared to the configuration on the
                    server). Note: This status reflects the status
                    of the configuration's hierarchical
                    references. If the -objects option is specified,
                    status reflects the status of configuration's
                    hierarchical references and objects.

                    Possible values are:
   - Local Only     Indicates the hierarchical reference exists
                    only in the local work area.
   - Out-of-date    Indicates that the hierarchical reference in
                    the local work area does not match the
                    hierarchical reference on the server; for
                    example, an alias on the server could have
                    changed to refer to a new release or the
                    relative path of the hierarchical reference
                    on the server could have changed.
   - Server Only    Indicates the hierarchical reference was added
                    on the server.
   - Unknown        Indicates that status of the hierarchical
                    reference cannot be determined. This status is
                    displayed, for example, if the server on which
                    the configuration resides is unavailable.
   - Up-to-date     Indicates that the hierarchical reference in
                    the local work area matches the hierarchical
                    reference on the server.
o hierstatus        Indicates the overall status of the
                    configuration. By default (or if you specify the
                    '-report normal' option), this value is a summary
                    of the status of the configuration's
                    hierarchical references. If you specify
                    the -objects option, this value represents the
                    status of the objects contained in the
                    configuration as well as the status of its
                    hierarchical references. If you specify the
                    -recursive option, the value indicates the
                    status of the entire configuration hierarchy.
                    Possible values are:
   - Out-of-date    Indicates that the configuration in
                    the local work area does not match the
                    configuration on the server.
   - Unknown        Indicates that status of the configuration
                    cannot be determined. This status is
                    displayed, for example, if the server on which
                    the configuration resides is unavailable.
   - Up-to-date     Indicates that the configuration in
                    the local work area matches the hierarchical
                    reference on the server.
o hrefs             A list of property lists, one for each of the
                    configuration's hierarchical references.
                    (Displayed only if the configuration has
                    hierarchical references.)

Note:
o Your output will include 'notes' if an alias in your work area is
  out of date with respect to the server. For example, if DRAM@Silver
  initially references the DRAM@R1 configuration and the alias is
```

```
  changed such that DRAM@Silver now references the DRAM@R2
  configuration, your output would include the following notes:

  notes {{Old aliased release: R1} {New aliased release: R2}}

o When you run the showstatus command recursively, your output will
  include 'hrefs' status for each submodule containing hierarchical
  references. The <submodule_status> is a Tcl list of the module
  status information.

To show the status of the objects contained in your work area
configuration, use the showstatus command with the -objects and the
'-format list' options. Output from the command lists the status of
the configuration and its hierarchical references (as described
above). In addition, the output includes a Tcl list (content) that
describes the status of each of the objects contained in the
configuration.

target <module_URL>
relpath <relative_path>
[notes {{Old aliased release: <release_name>}
        {New aliased release: <release_name>}}]
status <Local Only | Server Only | Up-to-date | Unknown | Out-of-date>
content
       {
         path1 <path>
         path2 <URL>
         type <folder | file>
         objects
             {
                {
                   name <object_name>
                   type <file | folder>
                   objects {object_list}
                   name <file_name> type file
                   props1
                       {
                         state <absent | modified | present | reference
                                | unknown | unmanaged>
                         version <version_number>
                       }
                   props2
                       {
                         state <absent | modified | present | reference
                                | unknown | unmanaged>
                         version <version_number>
                       }
                }
             }
       }

   hierstatus <Up-to-date | Out-of-date | Unknown>
   [hrefs {{<submodule_status>} {...}}] ...

o content           Lists the objects in the configuration
                    and reports the status of each. (Displayed
                    only if you specify the -objects option with
```

```
                    the showstatus command.)
 o path1            The path to the work area directory containing
                    the configuration.
 o path2            The URL of the configuration on the server.
 o type             The type of object contained in the work area
                    path (folder or file).
 o objects          A list of objects and their properties.
                    (Displayed only when type is
                    folder.) For each object, the following
                    information is provided:
    - name          The name of the object
    - type          The object's type (folder or file)
    - props1 {...} props2 {...}
                    Properties of the objects in the configuration.
                    (Displayed only when object type is file.)
                    Properties are:
                    o version - The version number of the object.
                      (In certain cases, this property may not be shown.)
                    o state - The status of the object in the path
                      (your work area or configuration on the server.)
                      Possible values are:
                       - absent - Indicates that the object is not
                         present on this path (work area or
                         configuration on the server).
                       - modified - Indicates that the object has been
                         locally modified.
                       - present - Indicates that the object is
                         present on this path. (The version that is
                         present is reported in the version property.)
                       - reference - Indicates that the object is a
                         referenced object.
                       - unknown - Indicates that the object exists
                         in the work area but the fetched version
                         is unknown. This state is most commonly
                         reported when an object has been removed from
                         the workspace (with the rmfile command)
                         and then recreated.
```

SYNOPSIS

```
showstatus  [-format <type>] [-[no]hrefs] [-[no]objects]
            [-[no]recursive] [-releases]
            [-report {brief | normal | verbose | summary | script}]
            [-xtras <xtras>] <argument>
```

ARGUMENTS

- Workspace Module
- Legacy Module Base Directory
- External Module Instance

Workspace Module

```
<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.
```

Legacy Module Base Directory

```
<legacy module base     Specifies the workspace legacy module base
 directory>             directory. It is compared against the
                        corresponding server folder.

                        Note: You cannot run the showstatus command
                       against a module sub-folder.  The command must
                       be run against the top level module directory.
```

External Module Instance

```
<external_mod>          Specifies the external module instance. The
                        external module must be populated into the
                        workspace.
```

OPTIONS

- -format
- -[no]hrefs
- -[no]objects
- -[no]recursive
- -releases
- -report
- -xtras

-format

```
-format <type>          Determines the format of the output.
                        Valid values are:
                        o list -  Displays a list with the following
                          format:
                               {
                                name <name>

                               }

                          For a list of properties displayed, see the
                          "Understanding the Output" section above.

                        o text - Display a text table with headers and
                          columns. (Default) Objects are shown in
```

alphabetical order.

-[no]hrefs

-[no]hrefs — Determines whether to follow the hierarchical references to determine if their status is current.
-nohrefs does not trace the hierarchical references. This eliminates the time and server load that might be required to follow the hierarchical trail. (Default)
-hrefs traces the hierarchical references to determine if the reported status is current.

-[no]objects

-[no]objects — Indicates whether command should run the compare command to compare the status of each object in the workspace with the corresponding object version on the server.

-noobjects skips the object comparison. (Default)

-objects checks the status of the workspace objects.

-[no]recursive

-[no]recursive — Indicates whether the command should return the status for the specified module, or the specified module and all referenced modules.

-norecursive displays the status for the specified module only. (Default)

-recursive displays the status for the specified modules and all referenced modules and identifies why particular hierarchical references are not recursed.

Notes: If you run the showstatus command with the '-format list' option, the showstatus command captures all errors encountered in the hierarchy and displays a message containing all the error messages.

-releases

-releases — Indicates that the showstatus command should run recursively against a legacy module releases. (Legacy modules only.)

Note: If this option is not supplied, the status of hierarchical references to releases is always listed as up-to-date.

-report

-report <mode> — Specifies the type of status information to be displayed.

Valid values are:

- brief - Displays a summary for all data and detailed data for any items that are out of sync. For a description of the status information, see "Understanding the Output".
- normal - Displays the status of the hierarchical references (and optionally, file status) for the module. (Default) Also displays a table of conflicts if conflicts exist between the expected submodule and the actual submodule. For a description of the status information, see "Understanding the Output".
  Note: You can set the -report normal mode to report on the "needs Â update" status of hierarchical references with the ShowHrefsNeedCheckinStatus registry key. For more information on setting the registry key, see the DesignSync Administrator's Guide.
- verbose - Displays the status of the hierarchical references and additional information about whether the hierarchical references need updating (and optionally, file status) for the module. Displays a table of conflicts if conflicts exist between the expected submodule and the actual submodule, and additional information. For a description of the status information, see "Understanding the Output".
- summary - Displays the target and base directory of the module, the status of each module, and the overall status of the module in the workspace. Also displays a table of conflicts if conflicts exist between the expected submodule and the actual submodule. For a description of the status information, see "Understanding the Output".

```
                        o script - Returns a Tcl list of
                          config_name/property_list pairs.  This is
                          identical to using running showstatus with
                          -report verbose -format list.
```

-xtras

```
 -xtras <xtras>           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

```
 If you run the showstatus command with the '-format list' option, it
 returns a Tcl list. For a complete description of the output, see the
 "Understanding the Output" section.
```

SEE ALSO

```
 addhref, rmhref, compare, ls, swap show, edithrefs, command defaults
,
```

EXAMPLES

- Module Hierarchy for Module Examples
- Example Showing Module Href Status Where Hrefs are Current
- Example Showing Module Href Status Where Hrefs are Outdated
- Example Showing Outdated Module Href Status in List Format
- Example Showing Legacy showstatus Command Formats
- Example of using showstatus on a legacy module

Module Hierarchy for Module Examples

```
 All of the modules example assume the following hierarchy in your
 work area.
 Top                    stored in ~/MyModules/Chip
    CPU;Trunk:Gold      stored in ~/MyModules/Chip/CPU
      ALU               stored in ~/MyModules/Chip/CPU/ALU
```

Example Showing Module Href Status Where Hrefs are Current

```
This example lists the status of the hierarchical references in your
local work area as compared to the server.

 dss> showstatus -recursive Chip%0
  Beginning showstatus operation ...

  Status of module Chip%0 ...

  Chip%0: url - sync://srv2.ABCo.com:2647/Modules/Chip;Trunk:
  Chip%0: base directory - /home/rsmith/MyModules/chip

  Chip%0: Workspace version 1.7
  Chip%0: Server version    1.7
  Chip%0: Version is Up-to-date

  Href Name  Status      Url                                  Selector \
           Version  Relative Path
  ------------------------------------------------------------------
  CPU     Up-to-date  sync://srv2.ABCo.com:2647/Modules/CPU Trunk:Gold
            1.3      CPU
  ROM     Up-to-date  sync://srv2.ABCo.com:2647/Modules/ROM Trunk:
            1.2      /ROM

  Chip%0: No hierarchical reference conflicts found.

  Chip%0: Hrefs are Up-to-date

  Status of module CPU%1 ...

  CPU%1: url - sync://A/Modules/CPU;Trunk:Gold
  CPU%1: base directory - /home/rsmith/MyModules/chip/CPU

  CPU%1: Workspace version 1.3
  CPU%1: Server version    1.3
  CPU%1: Version is Up-to-date

  Href Name  Status      Url                                  Selector
          Version  Relative Path
   ------------------------------------------------------------
  ALU        Up-to-date  sync://srv2.ABCo.com:2647/Modules/ALU  Trunk:
          1.2      ALU

  CPU%1: No hierarchical reference conflicts found.

  CPU%1: Hrefs are Up-to-date

  Status of module ALU%2 ...

  ALU%2: url - sync://srv2.ABCo.com:2647/Modules/ALU;1.2
  ALU%2: base directory - /home/rsmith/MyModules/chip/CPU/ALU

  ALU%2: Workspace version 1.2
  ALU%2: Server version    1.2
  ALU%2: Version is Up-to-date

  ALU%2: No hierarchical references.
```

```
   ALU%2: Module hierarchy is Up-to-date.

   ALU%2: Module is Up-to-date.

   CPU%1: Module hierarchy is Up-to-date.

   CPU%1: Module is Up-to-date.

   Chip%0: Module hierarchy is Up-to-date.

   Chip%0: Module is Up-to-date.

   Finished showstatus operation.
```

Example Showing Module Href Status Where Hrefs are Outdated

```
 This example shows output of an showstatus operation where a
 hierarchical references in the module is out of date.

 stcl> showstatus -objects Chip%0

  Beginning showstatus operation ...

  Status of module Chip%0 ...

  Chip%0: url - sync://srv2.ABCo.com:2647/Modules/Chip;Trunk:
  Chip%0: base directory - /home/rsmith/MyModules/chip

  Chip%0: Workspace version 1.5
  Chip%0: Server version    1.6
  Chip%0: Version is Out-of-date

  Href Name  Status       Url
           Selector    Version  Relative Path
 ---------------------------------------------
  CPU        Up-to-date   sync://srv2.ABCo.com:2647/Modules/CPU
           Trunk:Gold  1.3      CPU
  ROM        Server Only  sync://srv2.ABCo.com:2647/Modules/ROM
           Trunk:      1.2      /ROM


  Chip%0: No hierarchical reference conflicts found.

  Chip%0: Hrefs are Out-of-date

  Workspace       Configuration          Status             Object
  Version         Version                                   Name
  ---------       -------------          ------             ------
  1.1              1.1                    Identical          chip.c
  1.1              1.1                    Identical          chip.doc
  1.1              1.1                    Identical          chip.h
```

```
Chip%0: Module is Out-of-date.
Chip%0: Needs update.

Finished showstatus operation.
```

Example Showing Outdated Module Href Status in List Format

```
This example shows the same data as in the previous example, an out
of data hierarchical reference, but the output is presented in list
format.

stcl> showstatus -format list -objects Chip%0
 href_status Out-of-date exists 1 basedir /home/rsmith/MyModules/chip
 type standard needs_checkin 0 content {path1
 /home/rsmith/MyModules/chip/Chip%0 path2
 sync://srv2.ABCo.com:2647/Modules/Chip@Trunk: type folder props1
 {type module url sync://srv2.ABCo.com:2647/Modules/Chip version 1.5
 relpath {} basedir /home/rsmith/MyModules/chip} props2 {type module
 url sync://srv2.ABCo.com:2647/Modules/Chip version 1.6 relpath {}
 modulepath {}} objects {{name chip.doc type file state identical
 props1 {state present version 1.1} props2 {state present version
 1.1}} {name chip.c type file state identical props1 {state present
 version 1.1} props2 {state present version 1.1}} {name chip.h type
 file state identical props1 {state present version 1.1} props2
 {state present version 1.1}}}} conflicts {} content_status
 Up-to-date hierarchy {} server {selector Trunk: uid
 1ba413d31cfbd405591dba00f2ef564a version 1.6 url
 sync://srv2.ABCo.com:2647/Modules/Chip} hrefs {{status Up-to-date
 relpath CPU selector Trunk:Gold name CPU version 1.3 type Module
 basedir /home/rsmith/MyModules/chip/CPU url
 sync://srv2.ABCo.com:2647/Modules/CPU modinstname CPU%1} {status
 {Server Only} relpath /ROM selector Trunk: name ROM version 1.2 url
 sync://srv2.ABCo.com:2647/Modules/ROM}} needs_update 1
 version_status Out-of-date missing {} actual {version_ci {} selector
 Trunk: uid 1ba413d31cfbd405591dba00f2ef564a version 1.5 url
 sync://srv2.ABCo.com:2647/Modules/Chip} hier_status Up-to-date
 fullname /home/rsmith/MyModules/chip/Chip%0 status Out-of-date
 modinstname Chip%0 #
```

Example Showing Legacy showstatus Command Formats

o Show the status of hierarchical references of a module configuration hierarchy in the work area as compared to the server:

```
dssc> showstatus -recursive <ModInstance>

For example:
dssc> showstatus -recursive Chip%0
```

o Show the status of hierarchical references and objects contained in a module configuration hierarchy in the work area as compared to the server:

```
dssc> showstatus -recursive -objects <ModInstance>
```

o Show the status of each configuration in the module configuration hierarchy, followed by a summary of the overall status of the hierarchy. (Note: Because '-files' is specified, each configuration's status represents the status of the configuration's hierarchical references and its objects.)

```
dssc> showstatus -recursive -objects -report summary \
     <ModInstance>
```

Example of using showstatus on a legacy module

This example lists the status of the hierarchical references in your local work area as compared to the server. It assumes the following data hierarchy in your work area.

```
 Top             kept in directory Designs/Top
   IO@TEST      kept in directory Designs/Top/IO
   Mem@DEV      kept in directory Designs/Top/Mem

 dss> showstatus -recursive Chip%0
```

This command displays the following output:

```
Target:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top

STATUS      HREF                                              RELATIVE PATH
-------------------------------------------------------------------------
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/IO@TEST  IO
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Mem@DEV  Mem

Configuration status: Up-to-date

============================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/IO@TEST
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/IO

No local or remote hierarchical references found for configuration.

Configuration status: Up-to-date

============================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/Mem
```

```
No local or remote hierarchical references found for configuration.

Configuration status: Up-to-date

================================================================

Status of all visited configurations.
STATUS      TARGET                                          PATH
----------------------------------------------------------------
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
/home/jsmith/Designs/Top/Mem
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/IO@TEST
/home/jsmith/Designs/Top/IO
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Top
/home/jsmith/Designs/Top

Summary: Up-to-date
```

showproduct

showproduct Command

NAME

```
showproduct         - shows the associated enterprise server Product
                      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 informtaion on
defining the ENOVIA server ,see the DesignSync Data Manager
Administrator's Guide.

The Product in ENOVIA must have a defined DSFA connection to the
module object in DesignSync.
```

SYNOPSIS

```
showproduct <object> [-branch <selector>] [-version <selector>]
```

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            Specifies the version of a module associated with
<selector>          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 an objects to
                    check into the local workspace branch that was
                    previously not present on the branch.
```

RETURN VALUE

```
This command has no return value. The command launches the default
the default web browser to display the information returned.
```

SEE ALSO

```
entobj show, entobj synchronize, populate
```

showlocks

Description

showlocks Command

NAME

```
showlocks           - Shows all locked items in the vault
```

DESCRIPTION

- Understanding the Output (Module-based)
- Understanding the Output (File-based)

This command provides a list of all items locked on the server.

Note: To show locked items in the workspace, use the ls command with the '-locked -workspace' options.

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 (Module-based)

By default the command reports the branch and version information of the folder or module being queried, and a table or Tcl list, containing a list of items locked on the server. The information is presented in order of objects searched and entries for each object are sorted by relative path.

The showlocks command returns the following information:

Column Titles	Property Names	Description
User	user	Displays the username of the user who owns the object lock.
Date	date	Displays the date and time the object was locked.
Name	name	Displays the name of the locked object.
Where	where	Displays the full path location of the workspace where the object is locked, including the machine name where the workspace is located.
	log	Displays the lock comment. Note: Only module branches contain lock comments.
	uid	Displays the unique identifier for the object.

Understanding the Output (File-based)

By default the command reports the branch and version information of the folder being queried, and a table or Tcl list, containing a list of items locked on the server. The information is presented in order of objects searched and entries for each object are sorted by relative path.

The showlocks command returns the following information:

Column Titles	Property Names	Description

User	user	Displays the username of the user who owns the object lock.
Branch	branch	For DesignSync objects and legacy module objects, the branch information is included in the table.
Tags	tags	Displays the tag information for DesignSync objects and legacy module objects.
Date	date	Displays the date and time the object was locked.
Name	name	Displays the name of the locked object.
Where	where	Displays the full path location of the workspace where the object is locked, including the machine name where the workspace is located.
	log	Displays the lock comment. Note: Only module branches contain lock comments.
	uid	Displays the unique identifier for the object.

SYNOPSIS

```
showlocks [-alloverriden] [-format text|list] [-modulecontext <context>]
          [-[no]overridden] [-[no]recursive] <argument>
```

ARGUMENTS

- Server URL (Module-based)
- Workspace (Module-based)
- Server URL (File-based)
- Workspace (File-based)

Server URL (Module-based)

```
<serverURL>    Specifying the server URL provides a list of locked
               objects on the server. If you specify a branch, only
               objects locked on that branch of the module are
               displayed. Specify the module's server URL in the
               format:
                    sync://<machine>:<port>/Modules/<category>/
                    <module_name>[.branchid][;<selector>].
```

Workspace (Module-based)

```
<workspace module>  Specifying the workspace argument provides a list
<workspace folder>  of locked objects at the corresponding server
                    location.
                    For a list of all the locked items in a
                    workspace, use the ls command with the -locked
```

```
                    option.
```

Server URL (File-based)

```
 <serverURL>    Specifying the server URL provides a list of locked
                objects on the server. If you specify a branch, only
                objects locked on that branch are displayed. Specify
                the server URL in the format:
                     sync://<machine>:<port>/<path>[.branchid][;<selector>].
```

Workspace (File-based)

```
 <workspace folder>  Specifying the workspace argument provides a list
                     of locked objects at the corresponding server
                     location.
                     For a list of all the locked items in a
                     workspace, use the ls command with the -locked
                     option.
```

OPTIONS

- -alloverridden
- -format
- -modulecontext (Module-based)
- -[no]overriden
- -[no]recursive

-alloverridden

```
 -alloverridden
```

-format

```
 -format text|       Determines the format of the output.  For
  list               information about the information displayed
                     see the "Understanding the Output" section.
                     Valid values are:
                     o text - Display a text table with headers and
                              columns. (Default) Objects are shown in
                              alphabetical order.
                     o list - Displays a list with the following
                              format:
                               {
                                 name <name>
                               }
```

-modulecontext (Module-based)

```
-modulecontext      Identifies the module version to operate on.
 <context>          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.
```

-[no]overriden

```
-[no]overriden
```

-[no]recursive

```
-[no]recursive      Indicates whether the command should return
                    the status for the specified argument, or the
                    specified argument and all subfolders.

                    -norecursive displays the status for the
                    specified argument only. (Default)

                    -recursive displays the status for the specified
                    argument and all subfolders.
                    Note: -recursive does not traverse module
                    hierarchies.
```

SEE ALSO

```
command defaults, ci, co, lock, populate, url locktime
```

EXAMPLES

- Example Showing the Locks on Module Members (Module-based)
- Example Showing a Module Branch Lock (Module-based)
- Example Showing the Locks on the Server from a Workspace Argument (File-based)

Example Showing the Locks on Module Members (Module-based)

```
This example shows the showlocks command running on a module with the
-format text option.
 dss> showlocks Chip
  Module Chip/Chip, branch 1 (Trunk) has content locks:

  User    Date               Name        Where
  ----    ----               ----        -----
  bobt   09/07/2006 15:09   /source.c    /home/rsmith/Chip/source.c

This example shows the showlocks command running on a module with the
-format list option.

dss> showlocks -format list Chip
 {branch 1 tags Trunk contents {{user bobt date {09/07/2006 \
  15:09} name /source.c where /home/rsmith/Chip/source.c log {}}}}
```

Example Showing a Module Branch Lock (Module-based)

```
This example shows the showlocks command running on a locked module
branch.
dss> showlocks ROM%1

 Module ROMDesign/ROM, branch 1.7.2 (Alpha) locked by user janh
on 04/05/2010 07:38 'Alpha released finalized.'
```

Example Showing the Locks on the Server from a Workspace Argument (File-based)

```
This example shows the showlocks command running on a workspace folder.

dss>  showlocks .
  sync://srv2.ABCo.com:2647/mbom
  User  Branch  Tags    Date               Name     Where
  ----  ------  ----    ----               ----     -----
  janh  1       Trunk   09/11/2006 15:18   abc.c    srv2.ABCo.com:/mbom
  janh  1       Trunk   09/11/2006 15:18   abc.txt  srv2.ABCo.com:/mbom

This example shows the showlocks command running on a workspace
folder with the -format list option.

dss> showlocks -format list .
  {name abc.c user janh branch 1 tags Trunk date {09/11/2006
  15:56} where srv2.ABCo.com:/mbom} {name abc.txt user janh branch 1
  tags Trunk date {09/11/2006 15:56} where srv2.ABCo.com:/mbom}
```

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
- enterpriseRegistryFile
- portRegistryFile
- projectRegistryFile
- serverRegistryFiles
- siteRegistryFile
- syncRegistryFile
- userRegistryFile
- usingSyncRegistry
- Customization Information
- customDir
- customSiteDir
- customEntDir

- siteConfigDir
- usrConfigDir
- userConfigFile
- Client Information
- connectTimeout
- commAttempts
- defaultCache
- fileEditor
- htmlBrowser
- proxyNamePort
- somTimeout
- Server Information
- berkdbIsShmEnabled
- berkdbShmKey
- isTestMode
- serverMetadataDir
- serverDataDir
- serverMachine
- serverName
- serverPort
- User Information
- home
- userName

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 Module Version	M	N/A	Include Module version that have tags, 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:

- Object - Workspace path to the object..
- Vault URL - Vault URL associated with the object.
- 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.

- Branch - Branch number.
- Branch tags - Branch tag names.
- 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.
- 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 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      Data Property
Label     Key  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.

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.

- Version - Version number.
- Version tags - Version tag names.
- Derived From - numeric parent version.
- Merged From - version used to create the current version.
- Date - version creation date.
- Author - version author.
- 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
- -exclude
- -format
- -lastbranches
- -lastversions
- -maxtags
- -modulecontext (Module-based)
- -output
- -[no]recursive (File-based)
- -report
- -[no]selected
- -stream
- -xtras (Module-based)
- --

-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
```

Workflows

SITaR

sitr Command

NAME

```
sitr                - SITaR commands
```

DESCRIPTION

```
These commands provide access to the SITaR (Submit, Integrate, Test,
and Release) environment. SITaR provides a simplified work-flow to
establish and distribute a baseline set of functionality to
development teams.  This SITaR commands implement this work-flow in a
module-based environment.

Notes:
* You must be using modules to take advantage of the SITaR
  environment. If you have legacy modules or DesignSync vaults that
  you want to use with the SITaR work-flow, you must upgrade them to
  modules using the upgrade command.

* DesignSync does not support an overlapping module structure in an
  environment where module caching is enabled.

 To display a list of available 'sitr' commands, do the following:

 dss> sitr <Tab>

Note: The 'sitr' commands are available from all DesignSync client
      shells.

SITaR commands do not support the command defaults environment.  To
minimize potential conflicts, you should disable command defaults in
your client session before using SITaR commands by running the
command "defaults off" in your client session.
```

SYNOPSIS

```
sitr <command> [<command_options>]
```

```
Usage: sitr  [env|integrate|lookup|mkbranch|mkmod|populate|release|
             select|status|submit|update]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
dss, dssc, help, stcl, stclc
```

EXAMPLES

```
See specific sitr commands.
```

sitr

sitr Command

NAME

```
sitr                - SITaR commands
```

DESCRIPTION

```
These commands provide access to the SITaR (Submit, Integrate, Test,
and Release) environment. SITaR provides a simplified work-flow to
establish and distribute a baseline set of functionality to
development teams.  This SITaR commands implement this work-flow in a
module-based environment.

Notes:
* You must be using modules to take advantage of the SITaR
  environment. If you have legacy modules or DesignSync vaults that
  you want to use with the SITaR work-flow, you must upgrade them to
  modules using the upgrade command.
```

```
* DesignSync does not support an overlapping module structure in an
  environment where module caching is enabled.

 To display a list of available 'sitr' commands, do the following:

 dss> sitr <Tab>

Note: The 'sitr' commands are available from all DesignSync client
      shells.

SITaR commands do not support the command defaults environment.  To
minimize potential conflicts, you should disable command defaults in
your client session before using SITaR commands by running the
command "defaults off" in your client session.
```

SYNOPSIS

```
sitr <command> [<command_options>]

Usage: sitr  [env|integrate|lookup|mkbranch|mkmod|populate|release|
             select|status|submit|update]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
dss, dssc, help, stcl, stclc
```

EXAMPLES

```
See specific sitr commands.
```

sitr context

sitr context Command

NAME

```
sitr context        - View the context for a submitted module
```

DESCRIPTION

- Understanding the Output

```
This command displays the context information for one or more
releases.  The context URL returned by the command can then be used
by the integrator to populate a sitr workspace that is identical to
the developer workspace at the time the release was submitted.

By populating the context module to recreate the development
environment, the integrator can locate any discrepancies between
their workspaces.  If integrator-performed tests are failing, a look
at the difference between the workspaces can help pinpoint the
appropriate fix or determine if a subsequent release within the module
hierarchy has caused the problem.

In order to use the module context, you must have enabled saving the
module context with the sitr environment variable
sitr_context_required and have defined a module to store the context
information with the sitr environment variable sitr_context_module.

Note: The module context does not include information about any
persistent views or filters applied to the developer's workspace.
```

Understanding the Output

```
The sitr context command returns a TCL list containing the following
information:
 o release tag for the SITaR release associated with the module
   context
 o Sync URL of the module version of the submission context.
 o list of first level modules with dynamic selectors with local
   modifications at the time the module was submitted.
```

SYNOPSIS

```
sitr context -allconfigs | -release <release> <argument>
```

ARGUMENTS

- Module Name
- Module URL

Module Name

```
<moduleName>    Specify the desired module using the relative module
                path:
                [<category>...]/<ModuleName>]
                For example:
                ChipDesigns/Chip

                Notes:
                If you do not specify the fully qualified module
                URL, the sitr lookup command searches for the module
                on the known servers first by checking the container
                server defined in the sitr_server variable, and then
                by checking the servers defined in the
                sync_servers.txt file. For more information on the
                sync_servers.txt file, see the url servers command.

                You must provide the full relative path to the
                module. If the module is in a category (or a
                category path), you must provide the category(s)
                along with the module, for example:
                ChipDesigns/300mm/Chip
```

Module URL

```
<ModuleURL>     Specify the desired module using the fully qualified
                module URL:

                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

- -allconfigs
- -release

-allconfigs

```
-allconfigs     Returns a list of module context version URLs for all
                releases of the module whose workspace context was
                saved.
```

```
                The -allconfigs is mutually exclusive with the
                -release option. Either the -allconfigs or the
                -release option must be specified.
```

-release

```
 -release       Returns the module context version URL for the
  <release>     specified module release. Specify the release as the
                release tag applied to the release desired.

                The -release is mutually exclusive with the
                -allconfigs option. Either the -allconfigs or the
                -release option must be specified.
```

RETURN VALUE

```
 This command returns a tcl list, indexed by release name. If the
 module context was not saved with the release, there is no return
 value.
```

SEE ALSO

```
 sitr env, sitr release, sitr submit
```

EXAMPLES

sitr env

sitr env Command

NAME

```
 sitr env            - Displays SITaR environment variables
```

DESCRIPTION

```
 This command displays the status of the environment variables and
 settings required by SITaR.

 The following table lists the environment variables referenced by
 SITaR:
```

Note: The container module referred to is the top-level module in the SITaR module hierarchy.

Variable name	Description
sitr_role	Defines the role of the user in their current SITaR workspace. There are two possible values: Integrate - when the user performs integration tasks such as reviewing, testing, and integrating submitted sub-module changes into a new baseline release. Design - when the user performs designer tasks, such as modifying a sub-module and submitting the changes to an integrator.
sitr_server	Defines the URL of the server hosting the container module. (sync://host:port)
sitr_container	Defines the name of the container module, for example "Top".
sitr_alias	Defines the tag used by SITaR to identify the last qualified release of the container module, for example, "golden".
sitr_workdir	Defines the workspace directory of the container module, for example, "~/Workspaces/top".
sitr_relpath	Defines the relationships between the top module and the sub-modules. There are two possible values: Peer - indicates a flat structure, where all modules are at the same level in the workspace. Cone - sets up a hierarchical structure, where modules are defined as sub-directories of the top-level module in the workspace. Note: This only affects the structure of the resulting workspace.
sitr_automcache	Determines whether automatic mcaching is on or off. By default, automcaching is on. To disable automcache, set the value to 0. If the scripted mirror functionality is used to maintain the mcache, turning off sitr_automcache can provide a performance enhancement to sitr populate.
sitr_branch	Defines the default branch of the container module. This default is used by the sitr integrator for the sitr integrate and sitr release commands. If the variable is not defined, "Trunk:" is used as the default branch.

Note: Instead of specifying sitr_branch, users with the Design role use the sitr_alias environment variable to identify the container module version defined by the integrator as the last qualified module.

sitr_integrator_update — Indicates whether the integrator is allowed to run the "sitr update" command. Ordinarily the user must have a Design role, to be allowed to run the "sitr update" command.

There are two possible values:

0 (zero) - Indicates that the integrator is not allowed to run the "sitr update" command. (Default)

1 (one) - Indicates that the integrator is allowed to run the "sitr update" command.

sitr_min_comment — Indicates whether a minimum comment is required by any sitr command that has a -comment option.

0 (zero) or no variable - indicates that no minimum comment is required.

1 (one) or greater - indicates the minimum comment length required.

sitr_context_required — Indicates whether the module context information is captured during a submit type action.
There are two possible values:

0 (zero) - Indicates that the module context information is not gathered during submit and release operations. This means that you may not be able to recreate the workspace conditions exactly, for example, if, during test failure, you want to recreate the test in the workspace to see why it worked before integration. (Default)

1 (one) - Indicates that the module context information is preserved when the workspace is submitted. This means that should there be a need, for example, during a test failure, the integrator could exactly reproduce the development workspace that submitted the change.

sitr_context_module — If the sitr_context_required variable is enabled (sitr_context_required=1), this variable must be set to a valid module to store the context for submit and release actions within sitr.

	Note: Do not specify a selector for the module. To specify a branch selector, use the sitr_context_branch variable.
sitr_context_branch	If the sitr_context_required variable is enabled, this variable can be set to specify the branch of the submission context for the module. The branch name, in conjunction with the sitr_context_module defines the module and branch information for the module used to store submittal context information. If no branch is specified, sitr uses the default, "Trunk" branch.

The "sitr env" command also indicates whether module caches (Mcaches) are enabled and whether auto-creation of module caches is enabled. If auto-creation of module caches in enabled, this command displays the top level mcache directory.

SYNOPSIS

```
sitr env
```

OPTIONS

This command has no options.

RETURN VALUE

If the sitr env command is successful, DesignSync returns an empty string (""). If the command fails, DesignSync returns an error explaining the failure.

SEE ALSO

sitr populate, sitr update, sitr submit, sitr integrate, sitr release, sitr mkmod

EXAMPLES

This example shows setting the SITaR environment variable settings for a SITaR Design role in Bourne shell (sh).Call this file with the user's login script.

```
Notes:
o There is no sitr_branch environment variable defined for the
  Design role.  The designer users automatically use the baseline
  defined for them by the integrator user, specified with
  sitr_alias.

o There is no sitr_integrator_update variable defined for the Design
  role. The designer automatically has access to the upgrade
  operation.

o The sitr_automcache variable is not set.  If mcaching were enabled
  in the environment, this would mean that sitr populate would be
  updating the mcache automatically.

#!/bin/sh
#SITaR Environment Variables file (.sitr_env)
sitr_role=Design
sitr_container=ProjxContainer
sitr_alias=baseline
sitr_server=sync://srv2.ABCo.com:2647
sitr_workdir=~/Workspaces/projxContainer
sitr_relpath=Cone
sitr_min_comment=10
sitr_context_required=1
sitr_context_module=sync://srv1.ABCo.com:2647/Modules/Context/ProjxContext
sitr_context_branch=Trunk

export sitr_role
export sitr_container
export sitr_alias
export sitr_server
export sitr_workdir
export sitr_relpath
export sitr_min_comment
export context_required
export_context_module
export_context_branch

$ . .sitr_env
$ set
 ...
 sitr_alias=baseline
 sitr_container=ProjxContainer
 sitr_relpath=Cone
 sitr_role=Design
 sitr_server=sync://srv2.ABCo.com:2647
 sitr_workdir=~/Workspaces/projxContainer
 sitr_min_comment=10
 sitr_context_required=1
 sitr_context_module=sync://srv1.ABCo.com:2647/Modules/Context/ProjxContext
 sitr_context_branch=Trunk

$ stcl
...
stcl> sitr env
 SITaR environment variable settings:
```

```
   sitr_role            = Design
   sitr_server          = sync://srvr2.ABCo.com:2647
   sitr_container       = ProjxContainer
   sitr_alias           = final
   sitr_workdir         = /home/rsmith/Workspaces/projxContainer
   sitr_relpath         = Cone
   sitr_min_comment     = 10
   sitr_automcache      = (NOT SET)
   sitr_branch         = Trunk:
   sitr_integrator_update = 0
   sit_min_comment      = 10
   sitr_context_required= 1
   sitr_context_module  = sync://srv1.ABCo.com:2647/Modules/Context\
       /ProjxContext
   sitr_context_branch  = Trunk

   Container Workspace  = /home/rsmith/Workspaces/projxContainer
   Container Module     = sync://srvr2.ABCo.com:2647/Modules/ProjxContainer
   Container Baseline   = sync://srvr2.ABCo.com:264/Modules\
       /ProjxContainer@final

Auto-creation of module caches is NOT enabled.
  (No module cache directory is defined in the 'module cache paths' list)
```

sitr integrate

sitr integrate Command

NAME

```
sitr integrate      - Integrates selected changes into the project
                      integration workspace
```

DESCRIPTION

- SITaR Integration interactive mode

```
This command is used by SITaR integrators to integrate selected
changes to the integration configuration of the container
(top level) module specified by the sitr_branch environment variable.
 If the sitr_branch environment variable is not set, the integration
branch is Trunk: by default.

Note: DesignSync does not support using overlapping modules in a
configuration in which you have module caching enabled.

The sitr integrate command is usually used as part of the SITaR
integrate process which involves:

1) Locating newly submitted candidate releases using the sitr lookup
   command.
```

```
2) Selecting the changes using the sitr select command.

3) Integrating the changes to make them available for integration
   testing using the sitr integrate command.

4) Testing the changes.

5) Releasing approved changes using the sitr release command.

The "sitr integrate" command integrates the selected changes and
populates the integration workspace with the appropriate container
module and sub-modules in preparation for testing.

The "sitr integrate" command provides both an interactive mode, which
allows you to review the items selected for integration prior to
confirming the changes, and a force mode which accepts and implements
all changes immediately.
```

SITaR Integration interactive mode

```
The SITaR interactive mode for integration provides a selection table
The selection table contains these columns:

Column Name          Definition
-----------          ----------
 A/D                 Indicates whether a module configuration is to be:
                     A - added to the default configuration
                     D - removed from the default configuration.

                     Note: This column is blank for configurations
                           that have no changes to be integrated.

Module Config        Displays the name of the module configuration in
                     the following format:
                     <moduleName>@<configName>

                     Note: If '-report verbose' is specified, this
                     column displays the full URL of the module
                     configuration in the following format:
                sync://<host>:<port>/Modules/<moduleName>@<configName>

Owner                Displays the login name of the person that
                     submitted the configuration/release.

Release Date         Displays the creation date for the tagged version
                     of the module.

Description          Displays the description text associated with the
                     configuration. If the description is long, it is
                     truncated to 25 characters followed by an ellipsis.
                     (...).

Relative Path        Displays the directory path to the sub-module
```

```
                    configuration, relative to the container module
                    directory.
```

SYNOPSIS

```
sitr integrate [-force] [-nopopulate] [-noprompt]
               [-report {brief | normal | verbose}] [-[no]truncate]
```

OPTIONS

- -force
- -nopopulate
- -noprompt
- -report
- -[no]truncate

-force

```
-force              Bypasses the integrate interactive mode, and
                    integrates all specified changes without prompting
                    for confirmation.  You still see a listing of the
                    changes made after the command completes.

                    Note:  If this is specified with -report verbose,
                    -force is ignored.
```

-nopopulate

```
-nopopulate         Disables the automatic populate of the integration
                    workspace after the integration option completes.
                    This means that your integration workspace does
                    not accurately reflect the updated configuration.
```

-noprompt

```
-noprompt           Specifies accepting the default of all interactive
                    dialogs, for example, when you specify -force,
                    sitr integrate prompts you to confirm the action
                    or review the affected objects. Specifying
                    -noprompt bypasses that query.
```

-report

```
-report [brief|    Specifies the format of the command output. This
 normal|verbose]   information is discussed in more detail in the
                   "SITaR Integration interactive mode" Possible
                   values are:

                   brief - Displays the same information as 'normal'.

                   normal - Displays the selection table and prompts
                   for confirmation of the individual integration
                   operations. (Default)  This mode is ignored when
                   -force is selected.

                   verbose - Displays the selection table with the
                  full module configuration URL.  When this mode is
                  specified with -force, -force is ignored.
```

-[no]truncate

```
-[no]truncate   Specifies whether to display a full long comment or
                truncate the comment to a single line.

                -notruncate displays the full length of the comment.
               Comments can be up to 1MB in length.

                -truncate displays only the first 25 characters of
               the comment, followed by an ellipsis (...) to show
               that the comment is longer. (Default)
```

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

```
sitr lookup, sitr select, sitr submit, sitr release, addhref, rmhref,
edithrefs
```

EXAMPLES

- Example Showing Integrating in SITR in Interactive Mode
- Example Showing Integrating in SITR

Example Showing Integrating in SITR in Interactive Mode

```
This example shows integrating the first version of the Chip module
into the SITaR container module using the interactive mode of the
sitr integrate command.

dss> sitr integrate

 A/D  Name  Module Config  Relative Path  Owner  Release Date  Description
 ------------------------------------------------------------------------
 Add  Chip  Chip@v1.1      Chip           rsmith  2007-03-07
                                                  Initial SITaR Module Release
 Do you want to integrate the above changes? <No>
yes

 Beginning addhref operation ...

 sync://srv2.ABCo.com:2647/Modules/top: Added hierarchical reference:
     Name:           Chip
     Object:         sync://srv2.ABCo.com:2647/Modules/Chip
     Type:           Module
     Selector:       v1.1
     Version:        1.1
     Relative Path:  Chip

 sync://srv2.ABCo.com:2647/Modules/top: Created new module version
 1.2.

 Finished addhref operation.
 Creating/Updating Workspace for Integration Use....

 top%0: Version of module in workspace updated to 1.2

 Creating Sub Module Instance 'Chip%0' with base directory
 '/home/rsmith/sitr_workspaces/integrate/Chip'
 ======================================================

 Creating Sub Module Instance with
    Base Directory  = /home/rsmith/sitr_workspaces/integrate/Chip
    Name            = Chip
    URL             = sync://srv2.ABCo.com:2647/Modules/Chip
    Selector        = v1.1
    Instance Name   = Chip%0
    Metadata Root   = /home/rsmith/sitr_workspaces
    Parent Instance = top%0


 Chip%0: Version of module in workspace updated to 1.1

 Beginning showhrefs operation ...

 Showing hrefs of module /home/rsmith/sitr_workspaces/integrate/top%0  ...

 /home/rsmith/sitr_workspaces/integrate/top%0: Workspace version - 1.2
 /home/rsmith/sitr_workspaces/integrate/top%0: Href mode - normal

 Name  Url                                Selector  Static Version
 Type    Relative Path
-----------------------------------------------------------------
```

```
  Chip  sync://srv2.ABCo.com:2647/Modules/Chip  v1.1      1.1
  Module  Chip

  Finished showhrefs operation.
```

Example Showing Integrating in SITR

```
This example shows the integration of a module version in without
interactive prompts.  The module configuration being integrated is
Chip v1.4.  As part of the integration activity, the previous module
version, Chip v1.1 is being removed from the integration workspace.

dss> sitr integrate -force

 Beginning rmhref operation ...

 sync://srv2.ABCo.com:2647/Modules/top: Created new module version 1.4.

 Finished rmhref operation.

 Beginning addhref operation ...

 sync://srv2.ABCo.com:2647/Modules/top: Added hierarchical reference:
     Name:           Chip
     Object:         sync://srv2.ABCo.com:2647/Modules/Chip
     Type:           Module
     Selector:       v1.4
     Version:        1.5
     Relative Path:  Chip

  sync://srv2.ABCo.com:2647/Modules/top: Created new module version 1.5.

  Finished addhref operation.
  Creating/Updating Workspace for Integration Use....

  top%0 : Version of module in workspace updated to 1.5

  ======================================================

  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: 1 Kbytes, 0 file(s), 0 collection(s), 100.0% complete
  Progress: 1 Kbytes, 3 file(s), 0 collection(s), 100.0% complete


  Chip%0 : Version of module in workspace updated to 1.5
```

sitr lookup

sitr lookup Command

NAME

```
sitr lookup         - Displays a list of modules matching the
                      specified criteria
```

DESCRIPTION

- Understanding the Output

```
This command can be used to look up information about all the
submodules associated with the SITaR container module, or information
about a specified module, module branch, or module configuration.  A
module configuration is a tagged module version.

SITaR integrators use this command to find module versions that have
been earmarked for integration into the SITaR container module. SITaR
designers and integrators use this command to lookup module
information across a number of servers.
```

Understanding the Output

```
The output of the sitr lookup command can be formatted for easy
viewing (-report brief, normal,and verbose options) or optimized for Tcl
processing (-report script).  Both viewing formats show the same
information, but may have different names. In the table below, the
Column Titles column shows the column heading used with -report and
-report verbose and the Property Names column shows list output key
value.

The sitr lookup command displays the following information:

Column          Property
Titles          Names           Description
------          -----           ------------
Module Config   (key)           Name of the module version in
                                <moduleName>@<ConfigName> format.  If
                                '-report verbose' or '-report script'
                                is specified, the command displays the
                                full module configuration URL.

                                Note: When -since is used, or no module
                                argument is specified, an asterisk (*)
                                is prepended to the module
                                configuration name to indicate module
                                versions posted after the specified
                                -since value. For more information see
                                the -since option.

Owner           author          Login name of the user who created the
                                configuration/release.
Release Date    date            Date that the release was created.
                                Note: Date is displayed in seconds. You
```

```
                              can change the format with the tcl
                              "clock format" command.

 Description    comment       The description text associated with
                              the configuration. If the text is
                              longer than a single line, it will be
                              truncated and an ellipsis (...) will be
                              used to show that the text has been
                              truncated.  The full comment is
                              displayed if the -notruncate option is
                              specified when the command is run.

                tagName       The SITaR version tag.
                tags          A list of all the tags associated with
                              the module configuration.
```

SYNOPSIS

```
sitr lookup [-allconfigs] [-report {brief | normal | verbose | script}]
            [-since <timestamp> | <rel_config>] [-[no]truncate]
            [<argument> ...]
```

ARGUMENTS

- Module Name
- Server URL

```
When no argument is specified, the sitr lookup command returns
information about all modules in the container module for the
referenced branch.

Notes:
 o When a module or module version is selected, the sitr lookup
   command displays the first matching module.

 o When -since <timestamp> is specified with a module argument, the
   sitr lookup command returns only the list of module versions
   created after the specified timestamp. For more information, see
   the -since option.
```

Module Name

```
<moduleName>     Specify the desired module using the relative module
                 path:
                 [<category>...]/<ModuleName>][@<selector>]
                 For example:
                 ChipDesigns/Chip@Trunk:Gold
```

```
                Notes:
                If you do not specify the fully qualified
                module URL, the sitr lookup command searches for the
                module on the known servers first by checking the
                container server defined in the sitr_server
                variable, and then by checking the servers defined
                in the sync_servers.txt file. For more information
                on the sync_servers.txt file, see the url servers
                command.

                You must provide the full relative path to the
                module. If the module is in a category (or a
                category path), you must provide the category(s)
                along with the module, for example:
                ChipDesigns/300mm/Chip
```

Server URL

```
<ServerURL>     Specify the desired module using the fully qualified
                module URL:

 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, <module> is the name of
                the module, and <selector> is the optional selector
                that identifies the specific module version.
                For example:
                sync://serv1.abco.com:1024/Modules/ChipDesigns/Chip
```

OPTIONS

- -allconfig
- -report
- -since
- -[no]truncate

-allconfig

```
-allconfigs     Specifies all versions of the module, including those
                on other branches or versions dated before the -since
                value, for inclusion in the output.  Versions after
                the -since value are differentiated by an asterisk
                (*) prepended to the module config name.
```

Note: When a module argument is specified, -allconfig is automatically set as the default unless -since is explicitly specified.

-report

-report normal|verbose |script	Specifies the format of the output. For information on the values displayed in the report, see the "Understanding the Output" section. Possible values are: o brief - Displays the same information as 'normal'. o normal - Displays the output in a user-friendly table format.(Default) o verbose - Displays the full module configuration URL. (sync://host:port/Modules/...) for the Module config field. Note: When the sitr_verbose_lookup environment variable is set to 1, this is the default behavior. o script - Returns a Tcl array in list format, keyed by the full URL of each module configuration.

-since

-since <timestamp> | <rel_config>	Defines a filter to restrict the command output. You can specify a timestamp or a module configuration to use as a filter. Normally when -since is not specified, the sitr lookup command displays information about all of the releases of all of the container submodules that have been created 'since' the most recent 'baseline' release was created. Note: When you specify a module argument to the sitr lookup command, the command does not use -since <baseline> as the default. It displays all versions of the specified module. -since <timestamp> Specify a timestamp value. Any configurations created after the specified value are displayed in the output. DesignSync uses the same public-domain date parser as the GNU family of tools. The parser supports a wide range of date and time specifications. This section lists a few of the date formats, but for a larger list, see the Date Format section of selectors help topic, or visit a GNU website for the complete specification.

```
Possible date and time formats include:
o Specifying the date as a numeric ([MM/DD/[YY]YY]
  [hh:mm:ss]), for example: 07/23/2007 10:40:00.
o Specifying the date by keyword such as the day of
  the week or a logical day of the week, for example:
  Yesterday.
o Specifying the time as a logical time, for example:
  10:40am.

-since <ModuleVersion> Specify a release
configuration or tagged version. DesignSync automatically
converts the configuration into a timestamp using the
release time and displays all configurations created
since that time.

Note:  If you do not specify a full configuration
name, sitr lookup applies the version to the
container module.  It does not use the argument to
determine the configuration.  For more information
see Example 3.
```

-[no]truncate

```
-[no]truncate   Specifies whether to display a full long comment or
                truncate the comment to a single line.

                -notruncate displays the full length of the comment.
               Comments can be up to 1MB in length.

                -truncate displays only the first 25 characters of
               the comment, followed by an ellipsis (...) to show
               that the comment is longer. (Default)
```

RETURN VALUE

```
When you run 'sitr lookup' with the '-report script' option, it returns
a Tcl array in list format.  For information on the information
contained in array, see the Understanding the Output section.  For a
sample Tcl file that processes the output of the -report script file,
see Example 2.

If you run 'sitr lookup' with any other -report option, it returns an
empty string ("").  If the command fails, DesignSync returns an error
explaining the failure.
```

SEE ALSO

```
url servers
```

EXAMPLES

- Example of Looking Up Configurations in SITaR
- Example of Sample Tcl procedure using Lookup Output
- Example of Using the -since Option
- Example of Looking up all module configurations for all modules

Example of Looking Up Configurations in SITaR

```
This example shows a lookup of the configurations of the Chip
module.  The first example uses the -report normal format.  The
second example uses the -report script format.

dss> sitr lookup Chip

 Module Config  Owner    Release Date  Description
 ---------------------------------------------------------------
 Chip@v1.1      rsmith   2007-02-28    Initial SITaR Module Release
 Chip@v1.2      rsmith   2007-02-28    Initial Chip release

dss> sitr lookup -report script Chip
 sync://srv2.ABCo.com:2647/Modules/Chip@v1.1{immutable 1 \
 tag_properties {{v1.1 1 {Initial SITaR Module Release}}} tagName \
 v1.1 date 1172679543 version 1.1 comment {Initial SITaR Module \
 Release} tags v1.1 author rsmith} \
 sync://srv2.ABCo.com:2647/Modules/Chip@v1.2 {immutable 1 \
 tag_properties {{v1.2 1 {Initial Chip release}} {Latest 0 {}}} \
 tagName v1.2 date 1172689484 version 1.2 comment \
 {Initial Chip release} tags {v1.2 Latest} author rsmith}
```

Example of Sample Tcl procedure using Lookup Output

```
This example shows a sample Tcl procedure that uses the output of
-report script.

 proc moduleInfo {moduleName} {
    array set moduleConfigArray [sitr lookup -report script $moduleName]
    foreach moduleConfig [lsort [array names moduleConfigArray]] {
        array unset configInfoArray
        array set configInfoArray $moduleConfigArray($moduleConfig)
        puts "moduleConfig = $moduleConfig"
        puts "configInfo:"
        parray configInfoArray
     }
 }
```

Example of Using the -since Option

```
This example shows some of the formats you can specify for the -since
```

```
option.

When you specify a version without a module configuration name, the
command uses the container module version.  This example shows the
-since option when used with and without a specified module
configuration.

This example lists all module configurations since the v1.1 version
of the Chip module was created.

dss> sitr lookup -since v1.1
 Looking for releases (submittals) of the sub-modules of the 'top'
 container


 Module Config  Owner    Release Date  Description
 ------------------------------------------------------------
 * Chip@v1.3    rsmith  2007-09-04    Incorporated QA comments

 '*' flags that the configuration was created since
     the date of the 'top@v1.1' release (2007-09-04 17:54:57)


dss> sitr lookup -since Chip@v1.1
Looking for releases (submittals) of the sub-modules of the 'top' container


 Module Config  Owner    Release Date  Description
 ------------------------------------------------------------------
 * Alu@v1.1     rsmith   2007-09-04    Initial SITaR Module Release

 Chip@v1.1      rsmith   2007-09-04    Initial SITaR Module Release
 * Chip@v1.2    rsmith   2007-09-04    initial QA version
 * Chip@v1.3    rsmith   2007-09-04    Incorporated QA comments

 '*' flags that the configuration was created since
     the date of the 'Chip@v1.1' release (2007-09-04 14:33:08)
```

Example of Looking up all module configurations for all modules

```
This example shows a lookup of all submitted module configurations
for all available modules.

dss> sitr lookup -allconfigs
 Looking for releases (submittals) of the sub-modules of the 'top' container


 Module Config  Owner    Release Date  Description
 ------------------------------------------------------------------
 ALU@v1.1       rsmith   2007-04-16    Initial SITaR Module Release

 Chip@v1.1      rsmith   2007-03-16    Initial SITaR Module Release
 Chip@v1.2      rsmith   2007-03-16    Added files to Chip
 * Chip@v1.3    rsmith   2007-04-16    update to chip.doc
```

```
  '*' flags that the configuration was created since
      the latest 'final' release of 'top' (2007-04-16 10:21:00)
```

sitr mkbranch

sitr mkbranch Command

NAME

```
sitr mkbranch       - Creates a module branch of a SITaR module
```

DESCRIPTION

```
This command creates a new branch from the specified module
version. You can branch the container module and any submodules. The
SITaR module branch permits SITaR projects to support parallel
development.

The sitr mkbranch command performs the following actions:
o Determines the appropriate module to branch.
o Tags the new branch with the specified branch name.

Optionally, the sitr mkbranch command performs the following
additional actions:
o Populates the local workspace with the new branch version and
  changes the selector of the workspace to point to new branch.
o Integrates new sub-module branches into the container module.
```

SYNOPSIS

```
sitr mkbranch  [-[no]comment <text> | -description <description>]
               [-[no]integrate] [-[no]populate]
               <branchname> [<argument> ...]
```

ARGUMENTS

- Server Module Version
- Workspace Module
- Workspace Module Base Directory

Server Module Version

```
<server module   Fully qualified URL for the sitr module version in
 version>        the format sync://<host>:<port>/Modules/[<category>]/
                 <module_name>@v<sitr_version_number>
```

```
                where <host> is the SyncServer on which the module
                resides, <port> is the SyncServer port number,
                <category> is the optional category classification,
                <module_name> is the name of the module, and
                <sitr_version_number> is the SITaR release version
                number.
```

Workspace Module

```
<workspace      Workspace module instance name.
 module>
```

Workspace Module Base Directory

```
<workspace      Workspace directory containing the module.  If you
 module base    specify a directory, it may contain only one
 directory>     module.  If the workspace directory contains more
                than one module, use the workspace module instance
                name to uniquely specify the module to branch.
```

```
Note: If no arguments are specified, the behavior of the sitr
mkbranch command depends on the user's role.  If the user's role is
Integrate, the container module in the workspace is branched.  If the
user's role is Design, and there is only one the sub-module in the
workspace, the sub-module is branched. If there is no container
module in the workspace, or there is more than one sub-module, the
command fails.
```

OPTIONS

- -comment
- -description
- -[no]integrate
- -[no]populate
- Branch Name

-comment

```
-comment        Specifies whether to submit the release with or
 <text>         without a description of changes.
                -comment <string> stores the value of <string> as
                the lock comment. This description is used in all SITaR
                reports and provides the integrator with information
                about the version being submitted.  If the comment
                includes spaces, enclose the description in double
                quotes. DesignSync accepts a comment of any
                length up to 1MB.  Long comments may be truncated in
```

the output of the commands that show sitr comments. If your comment includes ampersand (&) or equal (=) characters, they are replaced by the underscore (_) character in revision control notes.

This command respects the minimum comment length defined in the SiTaR environment variables, if the sitr_min_comment variable has been set.

Note: The -comment option is mutually exclusive with -description. If you do not specify either of these options, DesignSync prompts you to enter a check-in comment.

-description

-description <text> — Specify a description of the release being submitted. This description is used in all SITaR reports and provides the integrator with information about the version being submitted.

This command respects the minimum comment length defined in the SiTaR environment variables, if the sitr_min_comment variable has been set.

Note: The -description option is being deprecated in favor of -comment. The -comment and -descriptions options are mutually exclusive.

-[no]integrate

-[no]integrate — Specifies the behavior of the new branch within the container module. This option is only available for users with the "Integrate" role.

The -integrate option integrates the branched module into the container module. If the container module already contains one release of a sub-module, that reference is updated to point to the newly branched sub-module. If the container module is linked to multiple versions of the sub-module being branched, the module is not integrated.

Note: Because the container module cannot be integrated into itself, you cannot specify the -integrate option when branching the container module.

The -nointegrate option does not attempt to integrate the branched module into the container module. (Default)

-[no]populate

```
-[no]populate   Specifies the behavior of the new branch in the
                workspace.

                The -populate option updates the workspace with the
               files in the new branch and changes the selector for
               the workspace to point to the branch so all future
               updates to the workspace point to new branch.

                The -nopopulate option does not update the workspace
               or workspace selector to use the new branch. (Default)

                If both the -populate and -integrate options are
                used, the container module is populated into the
                workspace with the appropriate branched sub-modules.
                Only users with the Integrate role can specify the
                -integrate option.
```

Branch Name

```
<branchname>    Specifies the name to use for the new branch. For
                information on allowed branch names, see the "Tag
                Name Syntax" section for the tag command.
```

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

```
mkbranch, tag, sitr populate, sitr integrate
```

EXAMPLES

- Example of Branching the Container Module
- Example of Branching a Submodule
- Example of Creating a New Release From a Submodule Branch

Example of Branching the Container Module

```
This example shows performing a sitr mkbranch operation on only a
container module. The branch maintains the links to any existing submodules.

stcl> sitr mkbranch NXV-LA
 No arguments supplied so defaulting to
 'sync://srv2.ABCo.com:2647/Modules/top;Trunk:' ...

 Branching modules ...

 Beginning MkBranch operation...

 Branching:   sync://srv2.ABCo.com:2647/Modules/top;1.5 : Success -
 Created branch 1.5.1, tagged NXV-LA

 MkBranch operation finished.


 Tagging first module version on new module branches ...
 Beginning module tag operation on 'sync://srv2.ABCo.com:2647' ...

 Tagging:     sync://srv2.ABCo.com:2647/Modules/top;NXV-LA: :
 Added tag 'NXV-LA_v1.1' to version '1.5.1.1'

 Module tag operation finished on 'sync://srv2.ABCo.com:2647'.
```

Example of Branching a Submodule

```
This example shows performing a sitr mkbranch operation on only the
ALU submodule.  The sub-module belongs to the "NXV" category.  Note
that because the -populate option is used, the selector of the
workspace is changed to the new branch.

stcl> showmods

 Beginning showmods operation ...

 Name   Instance  Base Directory
     Url                                                Selector
 ------------------------------------------------------------------------
 ALU  ALU%0   /home/rsmith/NXV/alu
     sync://srv2.ABCo.com:2647/Modules/SITR/ALU  Trunk:Latest

 Finished showmods operation.


stcl> sitr mkbranch -populate -comment "Creating limited
 availability release" NXV-LA ALU%0

 Branching modules ...

 Beginning MkBranch operation...

 Branching:   sync://srv2.ABCo.com:2647/Modules/NXV/ALU;1.2 :
```

```
 Success - Created branch 1.2.1, tagged NXV-LA

 MkBranch operation finished.


 Tagging first module version on new module branches ...
 Beginning module tag operation on 'sync://srv2.ABCo.com:2647' ...

 Tagging:     sync://srv2.ABCo.com:2647/Modules/NXV/ALU;NXV-LA: :
 Added tag 'NXV-LA_v1.1' to version '1.2.1.1'

 Module tag operation finished on 'sync://srv2.ABCo.com:2647'.


 Updating modules in workspace ...


 Chip%0 : Version of module in workspace updated to 1.2.1.1


 top%0 : Version of module in workspace not updated (Due to not
 operating on entire module contents).

stcl>  stcl> showmods

 Beginning showmods operation ...

 Name   Instance  Base Directory
     Url                                                 Selector
 ------------------------------------------------------------------------
 ALU  ALU%0   /home/rsmith/NXV/alu
     sync://srv2.ABCo.com:2647/Modules/NXV/ALU  NXV-LA:Latest

 Finished showmods operation.
```

Example of Creating a New Release From a Submodule Branch

```
This example shows creating a new release, NXV-LA,  from newly
branched sub-module, RAM. Note that the previous integration link
from the TOP module to the RAM module is updated with a link to the
new branch.

stcl> sitr mkbranch -integrate -populate NXV-LA RAM%0

 Checking workspace status of container 'top%0' ...

 Branching modules ...

 Beginning MkBranch operation...

 Branching:   sync://srv2.ABCo.com:2647/Modules/NXV/RAM;1.2 : Success
 - Created branch 1.2.2, tagged NXV-LA

 MkBranch operation finished.
```

```
Tagging first module version on new module branches ...
Beginning module tag operation on 'sync://srv2.ABCo.com:2647' ...

Tagging:     sync://srv2.ABCo.com:2647/Modules/NXV/RAM;NXV-LA: :
Added tag 'NXV-LA_v1.1' to version '1.2.2.1'

Module tag operation finished on 'sync://srv2.ABCo.com:2647'.


Selecting modules for integration ...
Selecting sync://srv2.ABCo.com:2647/Modules/NXV/RAM@v1.2 (href name
= 'RAM', relpath = 'RAM') for Deletion
Selecting sync://srv2.ABCo.com:2647/Modules/NXV/RAM@NXV-LA_v1.1
(href name = 'RAM', relpath = 'RAM') for Addition

Integrating modules ...

Beginning rmhref operation ...

sync://srv2.ABCo.com:2647/Modules/top: Created new module version 1.6.

Finished rmhref operation.

Beginning addhref operation ...

sync://srv2.ABCo.com:2647/Modules/top: Added hierarchical reference:
    Name:           RAM
    Object:         sync://srv2.ABCo.com:2647/Modules/NXV/RAM
    Type:           Module
    Selector:       NXV-LA_v1.1
    Version:        1.2.2.1
    Relative Path:  RAM

sync://srv2.ABCo.com:2647/Modules/top: Created new module version 1.7.

Finished addhref operation.
Creating/Updating Workspace for Integration Use....



top%0 : Version of module in workspace updated to 1.7

=====================================================


RAM%0 : Version of module in workspace updated to 1.2.2.1
```

sitr mkmod

sitr mkmod Command

NAME

```
sitr mkmod          - Creates and releases the initial version of
                      the module
```

DESCRIPTION

```
This command creates and releases the initial version of a SITaR module.

All modules that are part of the SITaR work-flow should have at least
one compatible release name before being integrated into the
container module. The sitr mkmod command allows you to create a new
module or configure an existing module for use with the SITaR
work-flow.

 The sitr mkmod command performs the following actions:
  o Creates the module using the mkmod command, if needed.
  o Releases the module with the immutable tag, v1.1.

Optionally, you can use the sitr mkmod command to create the initial
container module that stores the integrated releases of the
sub-modules.  The name of the container module is defined by the
sitr_container environment variable and is created with the -top
option.
```

SYNOPSIS

```
sitr mkmod -top
sitr mkmod -context
sitr mkmod -name [category/]<name> | -vaultpath <url>
                 [-comment <text> | description <text>] [-nomcache]]
```

OPTIONS

- -comment
- -context
- -description
- -name
- -nomcache
- -top
- -vaultpath

-comment

```
-comment      Specifies whether to submit the release with or
 <text>       without a description of changes.

              -comment <string> stores the value of <string> as the
              description of the module. This description is used in
```

all SITaR reports and provides the integrator with information about module. The module version can also have a comment associated with it. The initial module version created by the sitr mkmod command always has the initial comment, "Initial Module Release."

If the comment includes spaces, enclose the description in double quotes. DesignSync accepts a comment of any length up to 1MB. Long comments may be truncated in the output of the commands that show sitr comments. If your comment includes ampersand (&) or equal (=) characters, they are replaced by the underscore (_) character in revision control notes.

If you do not enter a description, SITaR automatically enters the default description, "SITaR module."

This command respects the minimum comment length defined in the SiTaR environment variables, if the sitr_min_comment variable has been set.

Note: The -comment option is mutually exclusive with -description.

-context

-context — Specifies creation of the context module defined by the sitr_context_module environment variable. The sitr_context_module environment variable must already exist.

The context module is used to store the developer context for each development submittal. For more information, see the stir submit command or the sitr env command.

-description

-description "<text>" — Provide a text description of the module. A multi-word description must be enclosed within double quotation marks (""). If no description is entered, DesignSync assigns the default description "SITaR module."

The module description is for the module itself and remains with the module throughout its history. Each release of the module can also have a description. The description for the initial module release generated by this command is always "Initial Module Release."

This command respects the minimum comment length

defined in the SiTaR environment variables, if the sitr_min_comment variable has been set.

Note: This command has been deprecated in favor of -comment. The -comment option is mutually exclusive with -description.

-name

-name

Specify the category (optional) and name of the module being created and released on the DesignSync server associated with the $sitr_server environment variable. This option should only be used for new modules.

The location of the module is defined as $sitr_server/Modules/<name>. The module name must be unique for its location and must conform to the DesignSync module requirements:
- Must contain only printable characters
- May not contain spaces
- May not contain any of the following characters:
  ~ ! @ # $ % ^ & * ( ) , ; : | ` ' " = [ ] / \ < >

The -name option is mutually exclusive with the -top option and the -vaultpath option.

-nomcache

-nomcache

Specify whether automated module caching should be disabled for this module. By default, automated module caching is enabled.

You can change the module cache status of an existing module with the url setprop command using the following form:
url setprop sync://<host>:</port>/Modules/<modulename> \
sitrNoMcache 1

For more information on module caching with SITaR, see "Using Module Caches" in the Description section of the sitr populate command.

-top

-top

Specifies creating the container module. When you create the container module, DesignSync automatically sets the container module to the module named in the $sitr_container environment variable.

When the first release is created, the immutable v1.1 tag is automatically applied to the container module.

This option should only be used once, as part of the initial SITaR environment setup. This option is mutually exclusive with the other sitr mkmod options.

-vaultpath

-vaultpath Specify the location of a module vault to incorporate into the SITaR work-flow. The vaultpath allows you to specify modules on remote servers or specify an existing module for use with SITaR.

If you are creating a new module, you do not need to specify a vaultpath if the module is located on the server specified by the $sitr_server environment variable.

If you are adding an existing module to the SITaR workflow, use sitr mkmod with the -vaultpath option.

Note: If you're adding an existing DesignSync vault or legacy module to the SITaR work-flow, you must upgrade the data using upgrade first, then use sitr mkmod with the -vaultpath option to add the module to the SITaR work-flow.

The -vaultpath option is mutually exclusive with the -name option and the -top option.

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

sitr env, sitr update, sitr submit, sitr integrate, sitr release, url setprop, mkmod, sitr populate

EXAMPLES

- Example of Creating a Top-Level Module Container
- Example of Creating Module in SITR

Example of Creating a Top-Level Module Container

```
This is an example of running the sitr mkmod command to create the
top-level module container.

 dss> sitr mkmod -top
  Making module sync://srvr2.ABCo.com:2647/Modules/BigChip
  Creating module BigChip on the server...
  Module successfully created on the server.
  Module creation completed.
```

Example of Creating Module in SITR

```
This is an example of creating a module with the sitr mkmod
command. During the module creation, the module is automatically
tagged with an immutable configuration tag.

dss> sitr mkmod -name CPU -comment "CPU module"
 Creating module CPU on the server...
 Module successfully created on the server.
 Module creation completed.
 Beginning module tag operation on 'sync://srvr2.ABCo.com:2647' ...

 Tagging:     sync://srvr2.ABCo.com:2647/Modules/CPU;Trunk: \
 : Added tag 'v1.1' to version '1.1'

 Module tag operation finished on 'sync://srvr2.ABCo.com:2647'.

 {Objects succeeded (1)} {}
```

sitr populate

sitr populate Command

NAME

```
sitr populate       - Updates the workspace with the specified module
                      version
```

DESCRIPTION

- Using module caches

```
This command is used to populate workspaces with the appropriate
objects for both Design and Integrate work in the SITaR work-flow. It
is also often used by Designers to update their workspaces to accept
```

```
 the changes introduced by a new baseline release (as generated as a result
of
 the integrator executing the sitr release command).  You can use the
 sitr populate command to fetch either the baseline configuration or a
 specific configuration.

 By default, the sitr populate command fetches the baseline
 configuration of the container module appropriate for your role and
 the container module's dependencies.  Your role is defined by the
 $sitr_role environment variable.

   o If $sitr_role is set to "Design", meaning that your role is as a
     Designer, the sitr populate command populates the baseline
     configuration as defined by the sitr_alias environment
     variable. This should be the "latest qualified" release of the
     container module.  By default, sitr populate does not overwrite
     any locked, unmanaged, or modified objects already in your
     workspace. If you do not wish to preserve locked or modified
     objects, specify the -force option.

   o If $sitr_role is set to "Integrate", meaning that your role is as
     an integrator, the sitr populate command populates the default
     configuration of the container module. The integrator can then
     use the sitr integrate command to update this workspace with any
     desired changes and use it to evaluate and test the submitted
     changes for compatibility prior to the creation of a new baseline
     release.

 The sitr populate command provides a mechanism to overwrite locked or
 modified objects, or to view a list of locked or modified objects in your
 modules.  For more information, see the -force option.

 Notes:
   The sitr populate commands respects the href mode "normal" behavior
   defined for populate with the "HrefModeChangeWithTopStaticSelector"
   registry key. For more information, see the "Module Hierarchy"
   topic in the ENOVIA Synchronicity DesignSync Data Manager User's
   Guide.

   FOR INTEGRATORS:  If you want to populate a sitr module with the
   stored context information, create a new workspace, and run the
   DesignSync populate command using the context module's version
   SyncURL. If you do not know the context module's version url, use
   the sitr context command to locate it.  For more information, see
   the sitr context command.
```

Using module caches

```
 DesignSync features a module cache capability for UNIX systems that
 allows end users to populate their workspaces with symbolic links to
 immutably tagged versions of modules residing within a module
 cache. This reduces disk usage and enhances performance.

 Note: SiTaR automcache respects the cachability of an object.  If an
```

object is designed uncachable by either the caching for the object being disabled (caching disable) or sitr automcaching for the object being disabled (property sitrNoMcache set to 1 on the module URL), the object will not be included the sitr mcache.

SITaR can be configured to automatically create a module cache entry, for each submodule version that it populates. This can be managed one of two ways:
 o scripted mirrors with sitr automcache disabled
 o sitr automcache enabled

Scripted mirrors with sitr automcache disabled:
  When scripted mirrors are used, the module caches are maintained by the scripted mirror system. The environment variable should be disabled (sitr_automcache set to 0). Since sitr populate does not need to update the automcache, it does not perform checks on the mcache to determine if it is up-to-date, thus improving sitr populate performance. Since sitr_populate does not update the mcache, the mcache directory does not need to be writable by user executing the sitr_populate command.

sitr populate with module caching enabled:
  When the sitr environment variable sitr_automcache is enabled, the sitr_populate command is responsible for maintaining the mcache. When you use sitr_populate, it checks to determine if the module cache is up-to-date and, if it is not, it updates it.

  Administrators enable module caches by designating a group-writable directory as a module cache directory using SyncAdmin. This module cache directory stores the module cache entries as sub-directories within the specified module cache directory. The module cache directory must also be a workspace root directory.

  Important: If there is more than one module cache directory specified within SyncAdmin, SITaR uses the first directory on the list. When you add module cache directories, verify that the SITaR module cache directory is always listed first.

  Note: You can disable module caching for a particular module when you create the module by using the -nomcache option to the sitr mkmod command or by using the url setprop command on the module vault url to set the sitrNoMcache property to 1.

To determine if module caching (both manual and automatic) has been enabled for your SITaR installation, use the sitr env command.

When auto-creation of module caches is enabled, the initial sitr populate command automatically populates the module cache with the baseline version of each submodule, prior to creating the symbolic links.

Subsequent sitr populate commands populate new baseline versions of the submodules in to the module cache, as needed.

When module caching is enabled, and you run the sitr populate command with the -force option, SITaR prompts you to confirm the action when the workspace contains unmanaged files. Confirming the action removes

```
the unmanaged objects prior to replacing the module directory with a
symbolic link.  You can continue, cancel the sitr populate operation,
or display a listing of the affected objects.

When SITaR creates a module cache entry for a released submodule, it
disables write permissions for all objects and sub-directories in the
module cache. If either Designer or Integrator users attempt to
modify these objects directly, they receive the following error
message:

     No write permission, or
     Failed:som: Error 79: No rights to access working file


The module cache must be writeable by all users who access it.  The
users must also have their umask value set to 002.

Note: When the 'sitr update' command is used on a sub-module, the
      symbolic link is replaced with a physical directory containing
      the Trunk:Latest versions of the objects.  The sitr populate
      command with the -force option can be used to replace all of
      the directories previously populated with the sitr update
      command with symbolic links to the baseline configurations of
      all the submodules.
```

SYNOPSIS

```
sitr populate [-config <tagname>] [-force] [-noprompt]
              [-report {error | brief | normal | verbose}]
             [-xtras <list>]
```

OPTIONS

- -config
- -force
- -noprompt
- -report
- -xtras

-config

```
-config      Specifies an explicit released configuration of the
 <tagname>   container module to populate into the workspace. The
             released configuration must be an existing configuration
            of the container module.

             If you are populating a workspace with a different
             configuration than the one already present in the
             workspace, you must use the -force option.
```

Note: If you are using the sitr populate command to change your workspace from a specified configuration (populated with the -config option), to the baseline configuration, use the sitr populate command with no arguments.

-force

-force — Forces a populate of all modules and objects in the workspace to the specified configuration. The configuration is either the appropriate baseline configuration for your role or the configuration specified with the -config option.

Using the -force option, the sitr populate command overwrites locally modified objects with the specified versions of the modules and objects. If module caching is enabled, it replaces all module cache enabled sub-module workspaces with symbolic links to the appropriate module cache directory, removing any modified or unmanaged objects from the workspace.

The sitr populate command evaluates whether objects in the workspace will be removed when the command is run with the -force option. When there are objects that will be removed, the sitr populate command provides a prompt asking you to confirm the action, cancel the action, or optionally, provides a list of the affected objects. You can then review this list and decide whether to continue the populate command.

-noprompt

-noprompt — Specifies accepting the default of all interactive dialogs, for example, when you specify -force, SITaR asks you to confirm the action or review the affected objects. Specifying -noprompt bypasses that query.

-report

-report error|brief|normal|verbose — Specifies the amount and type of information displayed by populate command.

error - lists failures, warnings, and success/failure count.

brief - lists failures, warnings, module create/remove messages, some informational messages, and success/failure count. (Default)

```
                    normal - includes all information from brief, and
                    lists all the updated objects.

                    verbose - provides full status for each
                    object processed, even if the object is not
                    updated by the operation.
```

-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.
```

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

```
 caching, sitr env, sitr update, sitr submit, sitr integrate,
 sitr release, url setprop, command defaults
```

EXAMPLES

```
 This shows an example of using sitr populate to update a developer
 workspace. Note: The sitr populate command calls the populate command
 with the -brief option.

 dss> sitr populate -force
  Creating/Updating Workspace for Design Use....

  top%1 : Version of module in workspace updated to 1.7

  ====================================================
  /chip.c : Already Fetched and Unmodified Version 1.1
 /chip.doc : Already Fetched and Unmodified Version 1.3
  Total data to transfer: 1 Kbytes (estimate), 1 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), 100.0% complete
  Progress: 1 Kbytes, 1 file(s), 0 collection(s), 100.0% complete
```

```
  Chip%1 : Version of module in workspace updated to 1.6
```

sitr release

sitr release Command

NAME

```
sitr release        - Creates a new module release and updates the
                      baseline alias
```

DESCRIPTION

```
This command is used by the sitr integrator to create a new release
of the container module and update the baseline tag to point to the
new release. When a new release has been generated, designers can
update their workspaces to use it by running the sitr populate
command.

Each release is tagged with an automatically incrementing version tag
with the prefix v.  When the initial release configuration is created
with the sitr mkmod command, it is automatically tagged v1.1,
indicating that it is the initial version.  The subsequent
configurations are automatically tagged v1.2, v1.2, v1.3 etc.

When you want to increment the major version number, use the -release
option to the sitr release command to specify the new version number.

Notes:

o The baseline tag is identified by the sitr_alias environment
  variable.  The sitr integrator is identified by the sitr_role
  environment variable. The container module branch updated by the
  release command is identified by the sitr_branch environment
  variables.  If the sitr_branch environment variable is not set, the
  default branch, Trunk: is updated.

o SITaR uses the highest value version tag as the baseline for the
  auto-increment number. If for instance, after v1.15 the major
  version number is incremented to v2.0, and after that, a developer
  specifies v1.16 with the -release option, the next available
  version number is 2.1.  SITaR does not revert to a lower version
  number.

o Tags cannot be reused.
```

SYNOPSIS

```
sitr release  [-[no]comment <text> | -cfile <filename> |
```

```
 -description "<text>"] [-noprompt]
[-noupdate] [-release <sitr_branch><NUM.NUM>]
```

OPTIONS

- -comment
- -cfile
- -description
- -noprompt
- -noupdate
- -release

-comment

```
-comment          Specifies whether to submit the new release with or
 <text>           without a description of changes.

                  -comment <string> stores the value of <string> as
                  the release comment. This description is used in all
                  SITaR reports and provides the Integrator with
                  information about the version being submitted.  If
                  the comment includes spaces, enclose the description
                  in double quotes. DesignSync accepts a comment of
                  any length up to 1MB.  Long comments may be
                  truncated in the output of the commands that show
                  sitr comments. If your comment includes ampersand
                  (&) or equal (=) characters, they are replaced by
                  the underscore (_) character in revision control
                  notes.

                  If you do not enter a description, SITaR
                  automatically enters the default description, "SITaR
                  Release."

                  This command respects the minimum comment length
                  defined in the SiTaR environment variables, if the
                  sitr_min_comment variable has been set.

                  Note: The -comment option is mutually exclusive with
                  -description and -cfile. If you do not specify any
                  of these options, 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."
```

-cfile

```
-cfile            Specifies a file containing a text comment to use as
```

<file> 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 sitr comments. If your comment includes ampersand (&) or equal (=) characters, they are replaced by the underscore (_) character in revision control notes.

This command respects the minimum comment length defined in the SiTaR environment variables, if the sitr_min_comment variable has been set.

Note: The -cfile option is mutually exclusive with -comment and -description. If you do not specify any of these options, 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."

-description

-description <text> Specifies a description of the new release. If you do not specify a description on the command line, the sitr release command prompts for one before executing the release. If you do not enter a description after being prompted, or when using the -noprompt option, SITaR uses "SITaR Release" as the default description.

The description provides a method to identify changes to releases. The description information is used in SITaR lookups and reports.

This command respects the minimum comment length defined in the SiTaR environment variables, if the sitr_min_comment variable has been set.

Note: This option has been deprecated by the -comment option. The -description option is mutually exclusive with -comment and -cfile. If you do not specify any of these options, 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."

-noprompt

-noprompt Specifies accepting the default of all interactive

dialogs, for example, if you do not enter a description, specifying -noprompt sends the release with the default description.

-noupdate

-noupdate

Specify this option when you do not want to update the baseline configuration to match the configuration being released. This allows developers and integrators to explicitly select the release for testing or development, but does not create a new baseline for the designer's workspaces.

If this option is not specified, the sitr release command updates the baseline with the new release, making it generally available to integrators and designers by allowing it to be fetched as part of the standard sitr populate operation.

-release

-release <sitr_branch> <NUM>.<NUM>

Specify the release tag for the configuration being released. Generally this option is used only to update the major version number, for example, from version 1.x to version 2.0.

For releases on the Trunk: branch, The release tag should be in the format v<NUM>.<NUM>. For releases on other branches, the format should be <sitr_branch_name><NUM>.<NUM>. Generally, when you specify a release configuration tag, you do so only to update the major version number, but you can also update both numbers, for example, to align the release numbers of different modules in preparation for larger release.

If you do not use the standard format for the -release option, SITaR displays a prompt telling you that it is not recommended to use a release name that does not match the <sitr_branch><NUM>.<NUM> standard and asking you to confirm the action.

Tip: When you apply a non-standard tag to save the current state of the integration of the container module, use the -noupdate option, which applies the non-standard tag to the container, but does not move the baseline tag.

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

```
sitr populate, sitr update, sitr integrate
```

EXAMPLES

```
This example shows the integrator role releasing a tested
configuration of the Chip module for general use.

dss> sitr release
 Having a description associated with every release is useful
 for future reference. Please enter a description for this
 release:
Incorporating Fixes: 32021, 34553, and 32342
 Verifying the container module 'top%0' is up to date.  Please wait ...
 Beginning module tag operation on 'sync://srv2.ABCo.com:2647' ...

 Tagging:     sync://srv2.ABCo.com:2647/Modules/top;1.5 : Added tag
 'v1.3' to version '1.5'

 Module tag operation finished on 'sync://srv2.ABCo.com:2647'.

 Beginning module tag operation on 'sync://srv2.ABCo.com:2647' ...

 Tagging:     sync://srv2.ABCo.com:2647/Modules/top;1.5 : Added tag
 'final' to version '1.5'

 Module tag operation finished on 'srv2.ABCo.com:2647'.
```

sitr select

sitr select Command

NAME

```
sitr select         - Identifies module configurations to process
```

DESCRIPTION

- Viewing the sitr select Command Buffer
- Working with Multiple Module Versions

This command is used by SITaR integrators to select module configurations to be added to, or deleted from, the "integration/<Default>" configuration of the container module.

Generally this command is used to select the latest submitted module version for inclusion into the baseline module, or to revert a baseline configuration back to the last "good" version.

The sitr select command is an integral part of the SITaR integrate process which involves locating the newly submitted candidate releases using the sitr lookup command, selecting the desired releases using the sitr select command, and integrating the releases into the integration workspace.

The sitr select command allows you to perform one operation on multiple module configurations. To run multiple operations within a single integrate command, you may need to run the sitr select command more than once to select all of the desired changes. Once all the changes have been selected, run the sitr integrate command to implement the selections. The work-flow continues with testing and culminates with the release of the updated configuration as the new baseline.

Note: The sitr select command accumulates selections until the sitr integrate command is run, the -clear option to the sitr select command is used, or the client executing the select commands closes.

Viewing the sitr select Command Buffer

You can view the selections in the sitr select command buffer by running the sitr select command with no options or by specifying the -report verbose option. This provides a table documenting the submodules versions in the current default/integration configuration of the container module, and a list of selected integration changes in the following format:

Column Name	Description
A/D	Indicates whether a module configuration is to be: A - added to the default configuration D - removed from the default configuration. Note: This field is blank for module configurations that are not selected for change
Name	Displays the unique hierarchical reference name for the container module link to the sub-module.
Module Config	Displays the name of the module configuration in the following format: <moduleName>@<configName> Note: If '-report verbose' is specified, this column displays the full URL of the module

configuration in the following format:
sync://<host>:<port>/<Modulepath>/<moduleName>@<configName>

Relative Path	Displays the directory path to the sub-module configuration, relative to the container module directory.
Owner	Displays the login name of the person that submitted the configuration/release.
Release Date	Displays the date the release was created.
Description	Displays the description text associated with the configuration. If the comment is long, the first 25 characters are displayed followed by an ellipsis (...). The full comment is displayed if the -notruncate option is specified when the command is run.

Working with Multiple Module Versions

All module selections are associated with a relative path (relpath) and hierarchical reference name in the workspace of the container module. The sitr select command tracks all selected changes based on the 'relpath' location of each container submodule and the hierarchical reference name. This allows for overlapping modules to share the same relative path.

When you update a module release to the relpath of a selected sub-module already in the container module, SITaR targets the old version for removal and replaces it with the new version when the integrate command is run.

If the neither the -relpath option nor the -name option is explicitly specified, SITaR uses the the default relpath location and the default href name. If the default href name is already in use, or if more than one href exists to the specified module, SITaR returns an error indicating that a unique relpath or href name must be specified. If an href exists for that module in the container, the old version is targeted for removal and replaced with the version when the sitr integrate command is run.

Note: The default location for sitr_relpath is based on the hierarchical setup defined for your SITaR installation. For more information on determining how the default relpath is calculated, see the -relpath option section.

If a -relpath option is specified, but the -name option is not specified, SiTaR uses the default relpath location and the specified href name. If the default href name is already in use, SITaR returns an error indicating that a unique href is required. If an href exists for that module in the container, the old version is targeted for removal and replaced with the new version when the sitr integrate

```
command is run.

For example, if the 'top' container module currently contains a
reference to the ALU@v1.2 release of the ALU submodule with a default
(Cone) relpath of 'ALU' and you run the following sitr select command:

  sitr select ALU@v1.3

The ALU@v1.2 configuration in the relpath location 'ALU' is selected
for deletion from the container, and the ALU@v1.3 configuration is
selected for addition at the same relpath.

Using the -relpath option, however, you can place modules in
non-default relpath locations.  This allows you to place versions of
the same module into a container multiple times.

If, for instance, using the previous example, you want to retain
ALV@v1.3 and add the new module release ALUv1.4, use the sitr select
command:

   sitr select -relpath ALU.4  ALU@v1.4

When you perform the sitr integrate command, SITaR makes both versions
of the ALU module available within the container module.

If you subsequently realize that you do not want the ALU@v1.3 version,
you can specify it with the sitr select -delete option, and it is
removed from the container module after the next sitr integrate
command.
```

SYNOPSIS

```
sitr select [-clear] [-delete] [-name <href name>]
            [-relpath <relpath>] [-report {brief | normal | verbose}]
            [-[no]truncate] [<argument>...]
```

ARGUMENTS

- Module Name
- Server URL

Module Name

```
<moduleName>     Specify the desired module using the relative module
                 path:
                 [<category>...]/<ModuleName>@<selector>
                 For example:
                 ChipDesigns/Chip@Trunk:Gold

                 Notes:
                 If you do not specify the fully qualified
```

```
                  module URL, the sitr lookup command searches for the
                  module on the known servers first by checking the
                  container server defined in the sitr_server
                  variable, and then by checking the servers defined
                  in the sync_servers.txt file. For more information
                  on the sync_servers.txt file, see the url servers
                  command.

                  You must provide the full relative path to the
                  module.  If the module is in a category (or a
                  category path), you must provide the category(s)
                  along with the module, for example:
                  ChipDesigns/300mm/Chip
```

Server URL

```
 <ServerURL>      Specify the desired module using the fully qualified
                  module URL:

  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
```

OPTIONS

- -clear
- -delete
- -name
- -relpath
- -report
- -[no]truncate

-clear

```
 -clear           Removes any existing selections from the sitr select
                  command buffer.

                  Note: Closing the client session containing the sitr
                  select command buffer also clears the command buffer.
```

-delete

-delete — Marks the specified module configuration for removal from the container module.

If -delete is not specified, the specified module configurations will be selected for addition to the container module.

-name

-name <hrefname> — Specifies the href name the container module uses to refer to the submodule. When integrate is run, an href is created between the container module and the specified submodule using this name. If you do not specify the -name option, SITaR uses the module name, if it can. If there are multiple valid modules, you must specify the specific module desired.

Note: The name value must be unique within the scope of the container module. If an href already exists with the name specified, the sitr select command fails.

-relpath

-relpath <relpath> — Specifies the directory path, relative to the directory of the container module, to place the specified module configuration.

If -relpath is not specified, SITaR uses the sitr_relpath environment variable to determine what relpath to use.

sitr_relpath	relpath
Cone	./<moduleName>
Peer	../<moduleName>

Note: The -relpath option can only take a single module configuration argument.

-report

-report — Specifies the format of the output. Possible values are:

```
o brief -  Displays command output including any
           warning or error messages.

o normal - Displays the selection table with the
           Name, Module Config (as a module instance
           name and version), and Relative Path
           columns when no module configurations are
           specified.  Displays status, warning and
           error messages when a module
           configuration is specified.

o verbose - Displays the selection table with the
            columns included in the "normal" display
            and Owner, Release, and Description. For
            Module config, verbose uses the full
            module configuration URL, instead of the
            module instance name and version.
```

-[no]truncate

```
-[no]truncate   Specifies whether to display a full long comment or
                truncate the comment to a single line.

                -notruncate displays the full length of the comment.
               Comments can be up to 1MB in length.

                -truncate displays only the first 25 characters of
               the comment, followed by an ellipsis (...) to show
               that the comment is longer. (Default)
```

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

```
sitr lookup, sitr submit, sitr integrate, sitr release, addhref,
rmhref, edithrefs
```

EXAMPLES

- Example of Selecting a Submitted Module
- Example of Replacing a Selection with an Updated Module
- Example of Showing the Selected Modules

Example of Selecting a Submitted Module

```
This example shows the initial selection of a submitted module, Chip,
for integration.

dss> sitr select Chip@v1.1
Selecting sync://srv2.ABCo.com:2647/Modules/Chip@v1.1 (href name =
'Chip', relpath = 'Chip') for Addition
```

Example of Replacing a Selection with an Updated Module

```
This example shows a subsequent selection of a submitted module,
Chip, for integration.  Notice that in addition to adding the
selected configuration of Chip, it also removes the active configuration
from the integration workspace.

dss> sitr select Chip@v1.4
 Selecting sync://srv2.ABCco.com:2647/Modules/Chip@v1.1 (href name =
 'Chip', relpath = 'Chip') for Deletion
 Selecting sync://srv2.ABCo.com:2647/Modules/Chip@v1.4 (href name =
 'Chip', relpath = 'Chip') for Addition
```

Example of Showing the Selected Modules

```
This example shows a list of the modules that have been selected.
When you run the sitr integrate command, these module configurations
will be processed as indicated (added or deleted from the integration
workspace.)

dss> sitr select
 A/D  Name  Module Config  Relative Path  Owner    Release Date
 Description
 ----------------------------------------------------------------
 Del  Chip  Chip@v1.1      Chip           rsmith  2007-03-16
 Initial SITaR Module Release

 Add  Chip  Chip@v1.4      Chip           rsmith  2007-06-06
 Fixes 32021, 34553, and 32342
```

sitr status

sitr status Command

NAME

```
sitr status         - Compares the local workspace to the specified
                      configuration
```

DESCRIPTION

- Understanding the Output

```
This command displays the status of your workspace and compares it to
the expected workspace configuration.  If your role is Design, the
expected configuration is the current baseline configuration as
defined by the sitr_alias environment variable.  If your role is
Integrate, the expected configuration is the 'Default' configuration.

The specific module version expected in the hierarchy by the sitr
status command depends on the value of hrefmode mode specified as a
persistent href mode on the workspace. If static mode is specified,
the status reported for the hierarchy follows the static version of
the object at the time the href was created.  If dynamic mode is
specified, the status evaluates the selectors dynamically.  If normal
mode is specified, contents follows 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.
```

Understanding the Output

```
The sitr status command displays the following information:

Column
Titles           Description
------           ------------
Module Instance  The unique "instance name" of each module in the
                 workspace or the directory name for the module cache
                 directory.

Workspace        The SITaR release name associated with each module
                 in the workspace.

<TaggedVersion>  The SITaR release name associated with server module
                 being compared against the workspace.  Generally
                 this is the baseline or default version, depending
                 on your defined role, but you can use the -versus
                 option to explicitly specify any release version.

Relpath          The relative path from the container module to the
                 base directory for each submodule.

Status           The status of the workspace module with respect to
                 the server module it is being compared to.  If the
                 workspace contains a symbolic link to a module
                 cached SITaR module version, the status contains
                 the prefix "(Mcache)".  Possible status values
                 include:

                 o In-sync - The compared urls, selectors, and
                   relpaths of the workspace and server-side
```

configuration are identical.

- Changed [selector,relpath,url] - The selector, relative path, or URL of the workspace module is different than the server module.

  Note: The status message displays only the appropriate values for your module instance, for example, if the selector has changed, but the URL and relative path are the same, the status message reads: "Changed selector".

  Selectors change when the version of the module in the baseline changes. This happens when the baseline is updated or regressed to a different version.

  Relative paths change when a different relative path is specified for a module by an integrator during the sitr select/integrate/test/release process.

  URL paths change when a module changes server.

- Empty relpath - The sub-module is referenced on the server, but is not populated in the workspace. This can happen when the href is created with an empty relpath. The "Empty relpath" label implies that the workspace should not have the files associated with the module.

- Server Only - The sub-module href exists on the server but is not in the workspace. This usually happens when a new module has been added to the baseline version of the container module since the last populate.

- Local Only - The sub-module exists in the workspace but is not referenced on the server. This usually happens when the sub-module has been removed from the baseline version of the container module.

- Update Mode - The sub-module selector for the workspace is Trunk: instead of the tagged baseline configuration. This usually happens when the user ran 'sitr update' on the sub-module because it's a module the user is actively changing.

SYNOPSIS

```
sitr status -versus <tagname> [-xtras <list>]
```

OPTIONS

- -versus
- -xtras

-versus

```
-versus       Specify an explicit tagged configuration of the
 <tagname>    container module to compare against the local
              workspace. The tagged configuration must be an
             existing configuration of the container module.
```

-xtras

```
-xtras       List of command line options to pass to the
 <list>      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

```
If the command is successful, DesignSync returns an empty string
("").  If the command fails, DesignSync returns an error explaining
the failure.
```

SEE ALSO

```
sitr populate, sitr env, showstatus
```

EXAMPLES

```
This example shows the status for an up-to-date module, Chip, and a
module in update mode, ALU.

dss> sitr status

 Workspace Directory:        /home/rsmith/sitr_workspaces/develop

 Comparing the submodule versions in the workspace with the versions
 of the submodules in the current baseline (final = v1.4)
 configuration of top

 Module Instance  Workspace  final  Relpath  Status
```

```
  ------------------------------------------------------
  top%1             v1.4       v1.4
    ALU%0           Trunk:     v1.1   ALU      Update mode
    Chip%1          v1.5       v1.5   Chip     In-sync
```

sitr submit

sitr submit Command

NAME

```
sitr submit         - Submits a module version for integration
```

DESCRIPTION

- Storing the Module Context

```
This command creates new releases of modules for integration
into the container module. When a developer has completed
development and is prepared to release a module to incorporate into
the baseline, the developer submits the release.

During the development process, you may release "check-point"
releases, or interim releases that require integration testing,
but should not be part of the baseline configuration.  You can
indicate these releases in any or all of three ways:

1. Use the -release option to provide a non standard release name for
   the release.
2. Use the -comment option to manually insert text into the
   description that indicates to the integrator that this release
   should not be integrated into the baseline.
3. Use the -nointegrate option to append the text "(do not
   integrate.)" to the end of the description.  This allows you to
   use the description text to describe the release configuration
   itself.

The sitr submit command prompts for confirmation of the submit action
for any of the following situations:
o You specify a non-standard release tag.
o You have a selector list specified as the workspace selector.
o Your workspace selector and your expected release tag do not match,
  for example, if your workspace uses a branch selector, but the
  expected release version is vNum.Num.

When submitting a directory hierarchy containing overlapping modules,
you should select the module instances individually in the submit
command.

Notes:

o SITaR uses the highest value version tag as the baseline for the
```

auto-increment number. If for instance, after v1.15 the major version number is incremented to v2.0, and after that, a developer specifies v1.16 with the -release option, the next available version number is 2.1. It does not revert to a lower version number.

- Tags cannot be reused.

- When a release is submitted and the sitr_context_required environment variable is set to 1 (meaning context is required), the sitr submit action captures the entire hierarchy of the designerâ€™s workspace root directory when the module is submitted. The hierarchy is stored as a version of a submission context module indicated by the sitr_context_module environment variable. This variable must be set to a valid module if the sitr_context_required variable is set, or the submit will fail.

Storing the Module Context

When you submit a release, you can preserve the module context; the local modifications to any of the other modules linked within the workspace. This allows the integrator to recreate the developer's workspace, in case the integrator's test results differ from the developers results or expectations.

To preserve the module context, both the developer and the integrator set the sitr environment variables for sitr_context_required and sitr_context_module. The sitr_context_required variable is used to enable preserving the context, and the stir_context_module variable stores the data.

Important: When the sitr integrator populates the context module, they will use the DesignSync populate command, not the sitr populate command.

SYNOPSIS

```
sitr submit [-comment <comment> | -cfile <filename> |
            -description <description>] [-force] [-nointegrate]
            [-[no]modified] [-noprompt]
            [-release [<branch_>]v<NUM>[.<NUM>]] [<argument> [...]]
```

ARGUMENTS

- Workspace Module Base Directory
- Workspace Module

Workspace Module Base Directory

```
<Workspace          Specify the desired workspace directory, for example:
 module base        /home/user/rsmith/mymods/Chip, or ../mymods/Chip.
 directory>         Workspace directories submitted for integration must
                    meet the following criteria:
                    o The workspace directory contains a valid module.
                    o All objects within the workspace are in-sync
                      with the server module version.  There can be no
                      locally modified objects or objects that need to be
                      updated to match the server version in the
                      workspace.
                    Notes:
                      * If you do not specify a workspace directory, SITaR
                        uses the current directory.
                      * If you have multiple modules (overlapping
                         modules) in the specified module base
                         directory, the command will fail.  You must
                         specify a specific module using the module
                         instance name.
```

Workspace Module

```
<workspace          Workspace module instance name, for example
 module>            Chip%1. When your module base directory contains
                    overlapping modules, you must specify the module
                    instance name.
```

OPTIONS

- -comment
- -cfile
- -description
- -force
- -nointegrate
- -[no]modified
- -noprompt
- -release

-comment

```
-comment            Specifies the comment to include with the submission.
 <text>             This description is used in all SITaR reports and
                    provides the Integrator with information about the
                    version being submitted.  If the comment includes
                    spaces, enclose the description in double
                    quotes. DesignSync accepts a comment of any length up to
                    1MB.  Long comments may be truncated in the output of
                    the commands that show sitr comments. If your comment
                    includes ampersand (&) or equal (=) characters, they are
```

replaced by the underscore (_) character in revision control notes.

If you do not enter a description using -comment, -description, or -cfile, SITaR automatically enters the default description, "SITaR Submittal."

This command respects the minimum comment length defined in the SiTaR environment variables, if the sitr_min_comment variable has been set.

Note: The -comment option is mutually exclusive with -description and -cfile. If you do not specify any of these options, 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."

-cfile

-cfile <file>

Specifies a file containing a text comment to use as the description of the submittal. DesignSync accepts a comment of any length up to 1MB. Long comments may be truncated in the output of commands that show sitr comments. If your comment includes ampersand (&) or equal (=) characters, they are replaced by the underscore (_) character in revision control notes.

This command respects the minimum comment length defined in the SiTaR environment variables, if the sitr_min_comment variable has been set.

Note: The -cfile option is mutually exclusive with -comment and -description. If you do not specify any of these options, DesignSync prompts you to enter a check-in commenteither 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."

-description

-description <text>

Specify a description of the release being submitted. This description is used in all SITaR reports and provides the Integrator with information about the version being submitted.

If you do not enter a description, SITaR

automatically enters the default description, "SITaR Submittal." DesignSync accepts a comment of any length up to 1MB. Long comments may be truncated in the output of the commands that show sitr comments.

This command respects the minimum comment length defined in the SiTaR environment variables, if the sitr_min_comment variable has been set.

Note: This option has been deprecated by the -comment option. The -description option is mutually exclusive with -comment and -cfile. If you do not specify any of these options, DesignSync prompts you to enter a check-in comment.

-force

-force — Specify this option if the work being submitted is not the version in Trunk:Latest. By default SITaR expects submittals to come from workspaces containing the Trunk:Latest version of the module. If, for example, you want to submit a previously submitted release or a branched version, use the -force option.

-nointegrate

-nointegrate — Specify this option when you are working with an interim module release that you do not want to integrate in the baseline configuration. Specifying this option appends the text "(do not integrate)" to the description field as a note to integrators.

-[no]modified

-[no]modified — Specifies what to do if a first level module with a dynamic selector contains locally modified files.

-nomodified indicates that the submission should fail if there are modified objects in a first level module with a dynamic selector, or within its hierarchy. (default)

-modified indicates that the submittal allows locally modified objects.

Important: A submittal always fails if the module being submitted contains locally modified files. If there are any first level modules with dynamic

selectors containing locally modified files, that information is be preserved on the server with the submitted module configuration.

-noprompt

-noprompt — Specifies accepting the default of all interactive dialogs, for example, when you specify -force, SITaR asks you to confirm the action or review the affected objects. Specifying -noprompt bypasses that query.

-release

-release [<branch_>] v<NUM>[.<NUM>] — Specify the release name for the submodule. Generally you will only specify this for major release number changes. If you do not specify the release tag, it is automatically calculated by incrementing the minor number from the last submitted release. For example, if the previously submitted release was tagged v1.9, the next release is tagged v1.10).

Note: The release number is incremented to the next available number, even if the previously released configuration was not integrated into the baseline.

When the module is located on the Trunk branch, the release name uses the format vNUM.NUM. When the module is located on the any other branch, the release name format is Branch_vNUM.NUM, where Branch is the branchname specified when the branch was created.

In order to use the SITaR auto-numbering release feature, the release tag must be specified in the following format:
BranchName_v<MajorReleaseNumber>.<MinorReleaseNumber>

Note: If the module is on the Trunk branch, you do not need to specify the branch name.

You can use -release to specify a non-standard SITaR release tag. For example, if you want to create a release to test in the integration environment, but not included in the baseline, you can create a release called "test" with the -release option. This does not affect the numbering of subsequent releases which continue to auto-increment from the last release tag. For example, if the release before "test" was 1.3, the release after test,

```
                assuming the -release option is not specified, is
                1.4.

                Note: When you specify a non-standard SITaR release
                tag, the system issues a warning and prompts you to
                confirm the action.

                Tip: Do not use the -release option to decrement the
                version number.
```

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

```
sitr populate, sitr update, sitr lookup, sitr integrate,
sitr release, sitr mkmod
```

EXAMPLES

```
This shows an example of a developer submitting changes to a module,
Chip, for the integrator to integrate into his workspace for review
and test.

dss> sitr submit
 Having a description associated with every submittal is useful
 for future reference. Please enter a description for this
 submittal:
Fixes to issues: 32021, 34553, and 32342
 Beginning module tag operation on 'srv2.ABCo.com:2647' ...

 Tagging:     sync://srv2.ABCo.com:2647/Modules/Chip;1.6 : Added tag
 'v1.5' to version '1.6'

 Module tag operation finished on 'srv2.ABCo.com:2647'.
```

sitr update

sitr update Command

NAME

```
sitr update         - Updates a submodule workspace with the
                      specified module version
```

DESCRIPTION

This command sets the workspace selector to the specified module version, usually the "Latest" version for the branch in the workspace, and populates the workspace. When the selector is set for the latest version of the workspace branch, the workspace is in "update" mode. When the designer has completed their development, they use the "sitr submit" command to submit the submodule for integration. The workspace remains in update mode. To set the workspace to the released version of the module, the developer can use the "sitr populate" command with the -force option.

Note: If there are multiple branch tags associated with the workspace branch and DesignSync is unable to determine the proper selector to use to update the workspace, you must use the -config option to supply a selector.

The "sitr populate" command, by default, does not modify a workspace submodule in update mode. Other submodules in the workspace, whose workspace selectors are that of the baseline configuration, are updated with respect to the baseline configuration.

When updating a directory hierarchy containing overlapping modules, you should select the module instances individually in the update command.

The update command operates in a hierarchically recursive manner by removing the outdated module versions and replacing them with the new module versions.

The sitr update command allows you to:

1) Switch submodule workspace directories from read-only mode, populated with the baseline configuration, to update mode, populated with the "Latest" version of the specified module branch.

2) Update the contents of submodule workspace directories already in update mode with the Latest versions of the submodule files. This allows the developer or integrator to pick up changes introduced since the last sitr update command was run on the workspace, for example, changes made by other users.

3) Populate a submodule workspace with a specified configuration of the module. This allows you to overwrite the current submodule configuration in the workspace.

Notes

- If the default "read only fetch" preference is set (for the "lock" model), designers must use the UNIX "chmod" command to adjust the write permissions to make the objects editable.

- The sitr update command should be run on all submodule workspace directories prior to performing any edit tasks, such as modifying, adding or removing any objects in the submodule.

```
  o You cannot run sitr update on a directory that already contains
    writable objects that have been modified, unless you specify how to
    handle modified objects in the workspace.

    To update all unmodified objects, use the -nooverwrite option.
    This preserves all modifications, and updates the other objects.

    To overwrite all objects with the specified configuration version,
    use the -force option. This replaces all modifications with the
    specified version.

  o When you update module cache enabled sub-module workspaces
    with symbolic links to the module cache entries, you must use
    the -force option if the directory contains modified or unmanaged
    files.

  o These files are considered editable by the SITaR system, but may
    be read-only within your workspace depending on your default
    populate settings.

  o Update mode is not restricted to a single developer use model.

  o The sitr update command uses the persistent hrefmode specified on
    the workspace to determine which referenced module version to
    populate. If static mode is specified, the hierarchy follows the
    static version of the object at the time the href was created.  If
    dynamic mode is specified, hierarchy follows the dynamic
    selector.  If normal mode is specified, the hierarchy follows 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.
```

SYNOPSIS

```
sitr update [-config <tag>] [-force] [-nooverwrite]
            [-noprompt] [-xtras <list>. [<argument>[ ...]]
```

ARGUMENTS

- Workspace Module Base Directory
- Workspace Module

Workspace Module Base Directory

```
<Workspace       Specify the desired workspace directory, for example:
 module base     /home/user/rsmith/mymods/Chip, or ../mymods/Chip.
 directory>
                 Notes:
                   * If you do not specify a workspace directory, SITaR
```

```
  uses the current directory.
* If you have multiple modules (overlapping
  modules) in the specified module base
  directory, the command will fail.  You must
  specify a specific module using the module
  instance name.
```

Workspace Module

```
<workspace       Workspace module instance name, for example
 module>         Chip%1. When your module base directory contains
                 overlapping modules, you must specify the module
                 instance name.
```

OPTIONS

- -config
- -force
- -nooverwrite
- -noprompt
- -xtras

-config

```
-config      Specify an explicit released configuration of the module
 <tag>       to populate into the workspace. The configuration must
             be an existing configuration of the module being updated.
             If the -config option is not specified, the workspace is
             updated with the "Latest" version of the specified
             module for the branch specified by the workspace
             selector.

             If you are updating a workspace with a different
             configuration than the one already present in the
             workspace, you must use the -force option.
```

-force

```
-force       Force update of the specified modules and objects in the
             workspace to the specified configuration. The
             configuration is either the appropriate baseline
             configuration for your role or the configuration
             specified with the -config option.

             Using the -force option, the sitr update command
             overwrites locally modified objects with the specified
             versions of the modules and objects.
```

Important: If module caching is enabled, and the configuration being populated is not 'Trunk:Latest,' it replaces all module cache enabled sub-module workspaces with symbolic links to the appropriate module cache entries. This removes any modified or unmanaged objects from the workspace. If the workspace contains modified or unmanaged objects, the sitr update command prompts you to confirm the action, cancel the action, or optionally, provides a list of the affected objects. You can then review this list and decide whether to continue the update command.

The -force and -nooverwrite options are mutually exclusive. If you specify both options, the command exits without updating any files. If neither the -force nor -nooverwrite options are specified, and the existing submodule workspace contains modified files, the sitr update command exits without updating any of the files and provides an error message explaining why.

-nooverwrite

-nooverwrite Specify this option to update any unmodified files to the specified configuration while leaving any modified files in their current state.

Note: If the -nooverwrite option is used with the -config option, the resulting workspace may contain a mixture of modified files from the previously populated configuration, and files from the '-config' specified configuration.

-noprompt

-noprompt Specifies accepting the default of all interactive dialogs, for example, when you specify -force, SITaR asks you to confirm the action or review the affected objects. Specifying -noprompt bypasses that query.

-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.

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

sitr populate, sitr submit, sitr release

EXAMPLES

This example shows a workspace, Chip, containing an earlier released version being updated for edit. Note: The sitr update command calls the populate command with the -brief option.

```
dss> sitr update Chip
 /chip.c : Already Fetched and Unmodified Version 1.1
 /chip.doc : Already Fetched and Unmodified Version 1.3
 Total data to transfer: 1 Kbytes (estimate), 1 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), 100.0% complete
 Progress: 1 Kbytes, 1 file(s), 0 collection(s), 100.0% complete


  Chip%1 : Version of module in workspace updated to 1.6
```

Legacy Module-Based Design

hcm

hcm Commands

NAME

```
hcm                 - Commands to manipulate legacy module data
                      (legacy)
```

DESCRIPTION

```
These commands allow you to create and manage your legacy modules.

To display a list of available 'hcm' commands, do the following:

 dss> hcm <Tab>

Notes:
 o  The 'hcm' commands are available from all DesignSync client shells.
 o  The legacy module, 'hcm' commands, must be typed with the hcm
    prefix. The non-legacy module commands are not typed with an hcm
    prefix.  For more information on non-legacy module command, see
    module.

Accessing Help
--------------
You can access information about the 'hcm' commands from the command
line.
- To display complete information for an 'hcm' command, do any of the
  following:

  dss> help "<commandName>"

  dss> <commandName> -help

  dss> <commandName> -?

  The following examples return help information for the hcm Example
  command. The double quotes are required, and there must be exactly
  one space between 'hcm' and 'Example':

  dss> help "hcm Example"

  dss> hcm Example -help

  dss> hcm Example -?

- To display brief (synopsis) information for an 'hcm' command, do
```

```
either of the following:

dss> help -brief "<commandName>"

dss> <commandName> -usage

The following examples return help information for the hcm addhref
command. The double quotes are required, and there must be exactly
one space between 'hcm' and 'addhref':

dss> help -brief "hcm addhref"

dss> hcm addhref -usage
```

SYNOPSIS

```
hcm <hcm_command> [<hcm_command_options>]

Usage:  hcm (addhref|addlogin|doc|get|mkalias|mkconf|mkmod|put|release|
            rmalias|rmconf|rmhref|rmlogin|rmmod|showconfs|showhrefs|
            showlogins|showmcache|showmods|showstatus|upgrade|version)
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
dss, dssc, help, stcl, stclc
```

EXAMPLES

```
See specific 'hcm' commands.
```

hcm addhref

hcm addhref Command

NAME

```
hcm addhref          - Creates a hierarchical reference between
                       legacy module configurations (legacy)
```

DESCRIPTION

- Controlling the Ability to Create Hierarchical References

```
This command creates a hierarchical reference, or connection, from the
configuration of an upper-level module to any of the following: a
submodule configuration, a DesignSync vault, or an IP Gear deliverable.
HCM stores the hierarchical reference on the server on which the
upper-level module resides.

This command is not the same as the addhref command for managing
non-legacy modules.  For help with the addhref command, type help
addhref.

For this command to be successful, the following must be true:
- The items you want to connect must exist.
- The module configuration from which the connection is made cannot
  be an alias, a release, a DesignSync vault, or an IP Gear
  deliverable.
- The server(s) on which the configurations reside must be available.
- The relative path must be unique within the scope of the upper-level
  module and cannot be nested. They cannot be contained within another
  relative path of the same configuration.

Notes:
- The hcm addhref command affects only the server definition of the
  hierarchical references. For example, if you fetched a configuration
  into your work area and later a hierarchical reference is added to
  the configuration on the server, the "added" hierarchical reference
  does not exist in the local definition. To identify hierarchical
  references added on the server, run the hcm showstatus command. To
  synchronize your local work area with the server, run the hcm get
  command.
- When specifying Synchronicity URLs, you should enter host names
  consistently; always use the same host/domain name for a particular
  host, regardless of whether the name is the actual machine name or a
  DNS alias. For access outside of the origin server's LAN, you
  should use fully qualified domain names.
- To take advantage of a performance enhancement for references on the
  same server, specify the same domain name for the -fromtarget and
  -totarget options. (A domain name is needed when the referenced
  submodule is on the same server so that the reference can be followed
  by users outside the server's LAN.) if you specify a different domain
  name for -fromtarget and -totarget, the addhref operation succeeds,
  but HCM does not utilize the performance enhancement.
```

Controlling the Ability to Create Hierarchical References

```
The ability to create hierarchical references can be
access-controlled.  By default, any user can create new
hierarchical references. If this policy is not acceptable for your
project, you (as the administrator or project leader) can restrict
or deny this capability for your site or individual
SyncServers. For more information, see the Synchronicity Access
Control Guide. See also information on the access allow and access
deny commands.

The command defaults system does not support this command.  The hcm
addhref command supported by the command defaults system is for
modules, not legacy modules.
```

SYNOPSIS

```
hcm addhref -fromtarget <uppermod_url> [-relpath <relative_path>]
            -totarget <submod_url>
```

OPTIONS

- -fromtarget
- -relpath
- -totarget

-fromtarget

```
-fromtarget             Specifies the URL of a configuration
 <uppermod_url>         corresponding to an upper-level module from
                        which you want to create a connection.

                        o To specify the default configuration of the
                          upper-level module, use the following syntax:
                           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.

                        o To specify a configuration other than the
                          default configuration, use the following
                          syntax:
                          sync[s]://<host>[:<port>]/<vaultPath>@<config>
                          where <host> is the SyncServer on which the
                          module resides, <port> is the SyncServer port
                          number, and <vaultPath> identifies the module
                          and <config> identifies the specific
                          configuration.
```

Notes:
- The upper-level module must exist and the server on which it resides must be available.
- The module configuration from which the connection is made cannot be an alias, a release, a DesignSync vault, or an IP Gear deliverable.

-relpath

-relpath <relative_path>	Indicates the path from the upper-level module to the submodule, IP Gear deliverable, or DesignSync vault. The hcm get command uses this path when recursively fetching items into your work area to create additional levels of hierarchy. For more information, see "Example 1b: Fetching the Entire Hierarchy" in the hcm get command topic. Paths: - Cannot be absolute. - Cannot be './'. - Should not contain at signs (@), number signs (#), back slashes (\), question marks (?), asterisks (*), semicolons (;), ampersands (&), pipes (|), or square brackets ([ ]). - Must be unique within the scope of the upper-level module and cannot be nested. They cannot be contained within another relative path of the same configuration. - Can be empty (""). Specify an empty path if you do not want to fetch the contents of the submodule into your local work area. This behavior may be desirable if you want to reference a ProjectSync project that tracks defects against a CAD tool. If you do not specify the -relpath option, the addhef operation uses the -totarget module name as the name of the submodule's base directory and places it below the upper-level module's base directory.

-totarget

-totarget <submod_url>	Specifies the URL of an IP Gear deliverable, DesignSync vault, or a submodule configuration to which you want to create a connection. o To specify a default configuration of a submodule, use the following syntax:

```
      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.

  o To specify a configuration other than the
    default configuration, use the following
    syntax:
    sync[s]://<host>[:<port>]/<vaultPath>@<config>
    where <host> is the SyncServer on which the
    module resides, <port> is the SyncServer port
    number, and <vaultPath> identifies the module
    and <config> identifies the specific
    configuration.

  o To specify an IP Gear deliverable, use the
    following syntax:
      sync[s]://<host>[:<port>]/Deliverable/<ID>
    where <host> is the SyncServer on which the
    module resides, <port> is the SyncServer port
    number, and <ID> is the deliverable ID number;
    for example,
sync://publisher.ipgsrvr1.com:2647/Deliverable/142.

  Note:
  - The submodule, DesignSync vault, or IP Gear
    deliverable must exist, and the server on
    which it resides must be available.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
access allow, access deny, hcm get, hcm release, hcm rmhref,
hcm showhrefs, hcm showstatus
,
```

EXAMPLES

- Creating Hierarchical References
- Creating Hrefs to Branch and Selector Configurations
- Using Empty Relative Paths
- Troubleshooting Embedded Relative Paths

Creating Hierarchical References

```
This example creates this hierarchy:
Chip               <= uses the default configuration
   Cpu             <= uses the default configuration
   Alu@R1          <= R1 is already released
     Dpath@R2      <= R2 is already released

It guides you through the creation of the connections between the
default configuration of the Chip module and the default configuration
of the Cpu module, between the default configuration of the Chip
module and the R1 configuration of the Alu module, and between the R1
configuration of the Alu module and the R2 configuration of the Dpath
module.

"Example 1: Creating, Fetching, and Releasing Module Hierarchies" in
the hcm Example topic includes the steps for creating the modules and
configurations that are presented in this example.

dss> hcm addhref -fromtarget sync://chip.ABCo.com:2647/Projects/Chip \
         -totarget sync://cpu.ABCo.com:2647/Projects/Cpu \
         -relpath Cpu

Hierarchical reference added on server sync://chip.ABCo.com:2647
sync://chip.ABCo.com:2647/Projects/Chip ->
sync://Cpu.ABCo.com:2647/Projects/Cpu : relative path = "Cpu"

dss> hcm addhref -fromtarget sync://chip.ABCo.com:2647/Projects/Chip \
         -totarget sync://alu.ABCo.com:2647/Projects/Alu@R1 \
         -relpath Alu

Hierarchical reference added on server sync://chip.ABCo.com:2647
sync://chip.ABCo.com:2647/Projects/Chip ->
sync://alu.ABCo.com:2647/Projects/Alu@R1 : relative path = "Alu"

Once you have created the hierarchical references, when you recursively
fetch the default configuration of the Chip module (run the hcm get
command with the -recursive option), the following directory structure
is added to your work area, /username/Designs/:

/username/Designs/Chip <= contains the data for the Chip module
/username/Designs/Chip/Alu <= contains the data for the Alu@R1
                              module
/username/Designs/Chip/Alu/Dpath <= contains the data for the
                                    Dpath@R2 module
```

Creating Hrefs to Branch and Selector Configurations

```
This example creates the following hierarchy in your work area:
Top                <= uses the default configuration
   Mem@DEV         <= DEV is a branch configuration
   IO@TEST         <= TEST is a selector configuration

It guides you through the creation of the connections between the
default configuration of the Top module and the DEV configuration
of the Mem module and between the default configuration of the Top
```

```
module and the TEST configuration of the IO module.

"Example 2: Creating and Removing Module Configurations" in the
hcm Example topic includes the steps for creating the modules and
configurations that are presented in this example.

dss> hcm addhref -fromtarget sync://srvr1.ABCo.com:2647/Projects/Top \
         -totarget sync://srvr1.ABCo.com:2647/Projects/Mem@DEV \
         -relpath Mem

Hierarchical reference added on server sync://srvr1.ABCo.com:2647
sync://srvr1.ABCo.com:2647/Projects/Top ->
sync://srvr1.ABCo.com:2647/Projects/Mem@DEV : relative path = "Mem"

dss> hcm addhref \
         -fromtarget sync://srvr1.ABCo.com:2647/Projects/Top \
         -totarget sync://srvr1.ABCo.com:2647/Projects/IO@TEST \
         -relpath IO

Hierarchical reference added on server sync://srvr1.ABCo.com:2647
sync://srvr1.ABCo.com:2647/Projects/Top ->
sync://srvr1.ABCo.com:2647/Projects/IO@TEST : relative path = "IO"

Once you have created the hierarchical references, fetch recursively
the default configuration of the Top module into your work area using
the hcm get command:

dss> hcm get -recursive  -target sync://chip.ABCo.com:2647/Projects/Top \
             -path /home/jsmith/Designs/Top

The hcm get command adds the following directory structure to your work
area directory, /username/Designs/:

/username/Designs/Top <= contains the data for the Top module
/username/Designs/Top/Mem <= contains the data for the Mem@DEV module
/username/Designs/Top/IO <= contains the data for the IO@TEST module
```

Using Empty Relative Paths

```
This example illustrates how you might track defects in your
design tools that affect your design. The ALU team relies on some
critical design tools, for example, DesignCompiler, that are not part
of their project. They use ProjectSync to track defects in the current
versions of these tools. When team members do a ProjectSync hierarchical
query on Alu, they want all DesignCompiler defects to be reported.

Notes:
- This example assumes that the ALU team has set up the DesignCompiler
  project in ProjectSync. The DesignCompiler project exists only to
  have notes attached to it; it does not have any data files checked
  into it.
- This example uses an empty relative path. The empty relative path
  ensures that when the ALU team recursively fetches the Alu module
  with the hcm get command that it does not fetch the contents of the
```

```
  DesignCompiler module into the local work area.

dss> hcm addhref -fromtarget sync://alu.ABCo.com:2647/Projects/Alu \
         -totarget sync://tools.ABCo.com:2647/Projects/DesignCompiler \
         -relpath ""

Hierarchical reference added on server sync://host2:port2
sync://alu.ABCo.com:2647/Projects/Alu ->
sync://tools.ABCo.com:2647/Projects/DesignCompiler: relative path = ""
```

Troubleshooting Embedded Relative Paths

```
This example identifies an embedded relative path (which cannot exist)
and shows how to correct it to successfully create the following
hierarchy:

Block              <= uses the default configuration
   Mem2            <= uses the default configuration
      DRAM@Silver  <= Silver is an alias

"Example 3: Creating, Modifying, and Putting Configurations" in the
hcm Example topic includes the steps for creating the modules and
configurations that are presented in this example.

Note: The order of addhref operations presented in this example
      differs from the order of addhref operations in the
      hcm Example topic. Additionally, the hierarchy presented in
      this example is a subset of the hierarchy presented in the
      hcm Example topic.

The following command creates the connection between the default
configuration of the Block module and the default configuration of
the Mem2 module.

dss> hcm addhref  -fromtarget sync://srvr2.ABCo.com:2647/Projects/Block \
         -totarget sync://srvr2.ABCo.com:2647/Projects/Mem2 \
         -relpath Mem2

Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Block ->
sync://srvr2.ABCo.com:2647/Projects/Mem2: relative path = "Mem2"

The following command tries to connect DRAM@Silver to Block rather
than to Mem2, which is not the desired hierarchy. Notice that the
relative path contains Mem2 (the relative path between the default
configuration of the Block module and the default configuration of
the Mem2 module).

dss> hcm addhref  -fromtarget sync://srvr2.ABCo.com:2647/Projects/Block \
         -totarget sync://srvr2.ABCo.com:2647/Projects/DRAM@Silver \
         -relpath Mem2/DRAM

ERROR: href with relpath "Mem2" (ancestor of "Mem2/DRAM") exists to
module "sync:///Projects/Mem2".
```

```
This command fails because the Mem2 relative path is embedded within
the Mem2/DRAM relative path. Relative paths cannot be nested; that is,
they cannot be embedded within another relative path. In this example,
the first hcm addhref command defines the relative path from Block to
Mem2 as Mem2. The second hcm addhref command attempts to define the
relative path from Block to DRAM@Silver as Mem2/DRAM, which includes
the existing relative path Mem2.

The following command successfully connects DRAM@Silver to Mem2,
which is the desired hierarchy.

dss> hcm addhref  -fromtarget sync://srvr2.ABCo.com:2647/Projects/Mem2 \
         -totarget sync://srvr2.ABCo.com:2647/Projects/DRAM@Silver \
         -relpath DRAM

Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Mem2 ->
sync://srvr2.ABCo.com:2647/Projects/DRAM@Silver:
 relative path = "DRAM"
```

hcm Example

hcm Example

NAME

```
hcm Example          - How to use the legacy commands (legacy)
```

DESCRIPTION

- Creating, Fetching, and Releasing Module Hierarchies
- Creating and Removing Module Configurations
- Creating, Modifying, and Putting Configurations
- Controlling Access to Servers

```
The examples presented in this topic include the setup for the
examples found in the individual legacy (HCM) command topics. The
tasks in these examples are illustrated using the Synchronicity Tcl
(stclc) command shell. The DesignSync and HCM operations could be
performed using any of the DesignSync client shells. Some specific
commands (for example "exec date > ModuleName1") will only work in
the Synchronicity Tcl command shell.

All of the commands presented in this topic are explained in other
topics of this help system. The HCM Help presents a flow that you might
use to share and reuse design work. For more information, see the
ENOVIA Synchronicity DesignSync Data Manager HCM User's Guide topic
"Overview".
```

Creating, Fetching, and Releasing Module Hierarchies

This comprehensive example provides the commands that set up examples presented in the hcm addhref, hcm get, hcm release, hcm mkalias, and hcm showhrefs command topics of this help system.

The commands in this example create and release the following module hierarchy:

```
Chip
  Cpu
  Alu@R1
    Dpath@R2
```

The following commands invoke the Synchronicity Tcl client shell, create the Dpath module, create a Dpath working folder in your local work area (/home/jsmith/Designs), fetch the default configuration of the Dpath module into the Dpath working folder, add a file to the default configuration of the Dpath module, and release the default configuration of the Dpath module.

```
% stclc
stcl> hcm mkmod -target sync://dpath.ABCo.com:2647/Projects/Dpath \
                -description "HCM Module for Dpath Block"
stcl> mkfolder Dpath
stcl> cd Dpath
stcl> hcm get -target sync://dpath.ABCo.com:2647/Projects/Dpath
stcl> exec date > Dpath1
stcl> ci -new -comment "First file in Dpath Block" Dpath1
stcl> hcm release -name R2
stcl> cd ..
```

The following commands create the Alu module, create an Alu working folder in your local work area (/home/jsmith/Designs), fetch the default configuration of the Alu module into the Alu working folder, and add a file to the default configuration of the Alu module.

```
stcl> hcm mkmod -target sync://alu.ABCo.com:2647/Projects/Alu \
               -description "HCM Module for Alu Block"
stcl> mkfolder Alu
stcl> cd Alu
stcl> hcm get -target sync://alu.ABCo.com:2647/Projects/Alu
stcl> exec date > Alu1
stcl> ci -new -comment "First file in Alu Block" Alu1
```

The following commands create the hierarchy between the Alu module and the R2 configuration of the Dpath module, fetch the Alu module hierarchy and its files into your local work area, and release the default configuration of the Alu module.

```
stcl> hcm addhref \
          -fromtarget sync://alu.ABCo.com:2647/Projects/Alu \
          -totarget sync://dpath.ABCo.com:2647/Projects/Dpath@R2 \
          -relpath Dpath
stcl> hcm get -recursive
```

```
stcl> hcm release -name R1
stcl> cd ..
```

The following commands create the Cpu and Chip modules, create the Cpu and Chip working folders in your local work area (/home/jsmith/Designs), fetch the default configurations of the Cpu and Chip modules into the respective working folders, and add files to each default configuration.

```
stcl> hcm mkmod -target sync://cpu.ABCo.com:2647/Projects/Cpu \
                -description "HCM Module for Cpu Block"
stcl> mkfolder Cpu
stcl> cd Cpu
stcl> hcm get -target sync://cpu.ABCo.com:2647/Projects/Cpu
stcl> exec date > Cpu1
stcl> ci -new -comment "First file in Cpu Block" Cpu1
stcl> cd ..
stcl> hcm mkmod -target sync://chip.ABCo.com:2647/Projects/Chip \
                -description "HCM Module for Chip Block"
stcl> mkfolder Chip
stcl> cd Chip
stcl> hcm get -target sync://chip.ABCo.com:2647/Projects/Chip
stcl> exec date > Chip1
stcl> ci -new -comment "First file in Chip Block" Chip1
```

The following commands create the hierarchy between the default configuration of the Chip module and both the default of the Cpu module and the R1 configuration of the Alu module, and fetch the Chip module hierarchy and its files into your local work area. For a discussion of these commands, see "Example 1: Creating Hierarchical References" in the hcm addhref command topic and "Example 1b: Fetching the Entire Hierarchy" in the hcm get command topic. For a discussion of the hcm showhrefs command, see "Example 1: Displaying the Hierarchical References Associated with a Configuration" in the hcm showhrefs command topic.

```
stcl> hcm addhref \
             -fromtarget sync://chip.ABCo.com:2647/Projects/Chip \
             -totarget sync://cpu.ABCo.com:2647/Projects/Cpu \
             -relpath Cpu
stcl> hcm addhref \
             -fromtarget sync://chip.ABCo.com:2647/Projects/Chip \
             -totarget sync://alu.ABCo.com:2647/Projects/Alu@R1 \
             -relpath Alu
stcl> hcm get -recursive \
              -target sync://chip.ABCo.com:2647/Projects/Chip

stcl> hcm showhrefs -path /home/jsmith/Designs/Chip \
                    -report command
```

The following commands release the default configuration of the Chip module, fetch the released Chip hierarchy into your local work area, and create an alias to the release. For a discussion of the hcm release command, see "Example 1: Recursively Releasing a Module" in the hcm release command topic. For a discussion of the hcm mkalias command, see the example in the hcm mkalias command topic.

```
stcl> hcm release -recursive -name R1
stcl> hcm get -recursive \
              -target sync://chip.ABCo.com:2647/Projects/Chip@R1
stcl> hcm showhrefs -target sync://chip.ABCo.com:2647/Projects/Chip@R1 \
                    -report command

stcl> hcm mkalias -target sync://chip.ABCo.com:2647/Projects/Chip@R1 \
                  -name Golden

The following command displays the configurations of the Chip module.
For a discussion of the hcm showconfs command, see the example in the
hcm showconfs command topic.

stcl> hcm showconfs \
          -target sync://chip.ABCo.com:2647/Projects/Chip
```

Creating and Removing Module Configurations

```
This comprehensive example provides the commands that set up examples
presented in the hcm mkconf, hcm addhref, hcm rmhref, hcm rmconf,
hcm showhrefs, and hcm showconfs command topics of this help system.

The commands in this example create the following module hierarchy:

Top
 Mem@DEV
 IO@TEST

The following commands invoke the Synchronicity Tcl client shell,
create the Top module, create a working folder in your local work area
(/home/jsmith/Designs) and associate the Top module with that folder,
create and check a file into the vault associated with the newly
created working folder, and release the default configuration of the
Top module.

% stclc
stcl> hcm mkmod -target sync://srvr1.ABCo.com:2647/Projects/Top \
                -description "Upper-level HCM (Top) Block"
stcl> mkfolder Top
stcl> cd Top
stcl> hcm get -target sync://srvr1.ABCo.com:2647/Projects/Top
stcl> exec date > Top1
stcl> ci -new -comment "First file in Top Block" Top1
stcl> cd ..

The following commands create the Mem module, create a working folder
in your local work area (/home/jsmith/Designs) and associate the Mem
module with that folder, and create and check a file into the vault
associated with the newly created working folder. For a discussion of
the hcm mkconf command, see "Example 1: Creating a Branch Configuration"
in the hcm mkconf command topic.

stcl> hcm mkmod -target sync://srvr1.ABCo.com:2647/Projects/Mem \
                -description "HCM Module for Mem"
```

```
stcl> hcm mkconf -branch DEV -name DEV \
                 -target sync://srvr1.ABCo.com:2647/Projects/Mem
stcl> mkfolder Mem
stcl> cd Mem
stcl> hcm get -target sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
stcl> exec date > Mem1
stcl> ci -new -comment "First version in DEV Branch for Mem1" Mem1
stcl> cd ..
```

The following commands create the IO module, create a working folder in your local work area (/home/jsmith/Designs) and associate the IO module with that folder, create and check a file into the vault associated with the newly created working folder, and create a TEST selector configuration of the IO module. For a discussion of the hcm mkconf command, see "Example 2: Creating a Selector Configuration" in the hcm mkconf command topic.

```
stcl> hcm mkmod -target sync://srvr1.ABCo.com:2647/Projects/IO \
                -description "HCM Module for IO"
stcl> hcm mkconf -name TEST -selector Trunk:TEST \
                 -target sync://srvr1.ABCo.com:2647/Projects/IO
stcl> mkfolder IO
stcl> cd IO
stcl> hcm get -target sync://srvr1.ABCo.com:2647/Projects/IO@TEST
stcl> exec date > IO1
stcl> ci -new -comment "First version in TEST Branch for IO1" IO1
stcl> tag -recursive TEST .
stcl> cd ..
```

The following commands create the hierarchy between the Top module and the DEV configuration of the Mem module and the TEST configuration of the IO module, fetch the Top module hierarchy and its files into your local work area, and display the hierarchical references in your local work area. For a discussion of these commands, see "Example 2: Creating Hierarchical References to Branch and Selector Configurations" in the hcm addhref command topic.

```
stcl> hcm addhref \
            -fromtarget sync://srvr1.ABCo.com:2647/Projects/Top \
            -totarget sync://srvr1.ABCo.com:2647/Projects/Mem@DEV \
            -relpath Mem
stcl> hcm addhref \
            -fromtarget sync://srvr1.ABCo.com:2647/Projects/Top \
            -totarget sync://srvr1.ABCo.com:2647/Projects/IO@TEST \
            -relpath IO
stcl> hcm get -recursive \
             -target sync://srvr1.ABCo.com:2647/Projects/Top \
             -path /home/jsmith/Designs/Top
stcl> hcm showhrefs -recursive \
                    -path /home/jsmith/Designs/Top
```

The following commands remove the hierarchical reference between the default configuration of the Top module and the TEST configuration of the IO module and remove the TEST configuration.

Note: By default, no users can remove configurations. If this policy is not acceptable for your project, you (as the administrator

```
      or project leader) can control this capability for your site
      or individual SyncServers. For more information, see the
      Synchronicity Access Control Guide.

stcl> hcm rmhref \
            -fromtarget sync://srvr1.ABCo.com:2647/Projects/Top \
            -totarget sync://srvr1.ABCo.com:2647/Projects/IO@TEST
stcl> hcm version
stcl> hcm rmconf -target sync://srvr1.ABCo.com:2647/Projects/IO@TEST

The following command shows the importance of fetching the Top
configuration after hierarchical references are added or deleted. For
a discussion of the hcm showhrefs command, see "Example 2:
Understanding When Hierarchical References Are Updated" in the
hcm showhrefs command topic. For a discussion of the hcm showstatus
command, see "Example 1" in the hcm showstatus command topic.

stcl> hcm showhrefs -recursive -path Top
stcl> hcm showstatus -recursive -path Top

The following commands display the modules available on the DesignSync
server sync://srvr1.ABCo.com:2647. The first two commands highlight
the different types of output available through the hcm showmods
command. The third command shows the correct usage of the
'-report script' option. For a discussion of these commands, see
"Example 1" and "Example 2" in the hcm showmods command topic.

stcl> hcm showmods -target sync://srvr1.ABCo.com:2647 -report normal

stcl> hcm showmods -target sync://srvr1.ABCo.com:2647 -report script

stcl> catch {hcm showmods -target sync://srvr1.ABCo.com:2647 \
                          -report script} modules_list
stcl> foreach module $modules_list {puts $module}
```

Creating, Modifying, and Putting Configurations

```
This comprehensive example provides the commands that set up examples
presented in the hcm get, hcm put, hcm release, hcm showconfs, and
hcm showstatus command topics of this help system.

The commands in this example create, update, manipulate, and release
the following module hierarchy:

Block
   BIST@TEST
   Mem2
      DRAM@Silver
      Collar@DEV

The following commands invoke the Synchronicity Tcl client shell,
create the Block module, create a working folder in your local
work area (/home/jsmith/Designs) and associate the Block module
with that folder, and create and check a file into the vault
```

associated with the newly created working folder.

```
% stclc
stcl> hcm mkmod \
          -target sync://srvr2.ABCo.com:2647/Projects/Block \
          -description "Upper-level HCM (Block) Module"
stcl> mkfolder Block
stcl> cd Block
stcl> hcm get -target sync://srvr2.ABCo.com:2647/Projects/Block
stcl> exec date > Block1
stcl> ci -new -comment "First file in Block" Block1
```

The following commands release the default configuration of the Block module. For information about the hcm release command, see "Example 2: Understanding the Necessity of Fetching All Configurations" in the hcm release command topic.

```
stcl> hcm release -name R2 -recursive
stcl> cd ..
```

The following commands create the DRAM module, create a working folder in your local work area (/home/jsmith/Designs), associate the DRAM module with the newly created folder, create and check a file into the vault associated with the newly created working folder, release the default configuration of the DRAM module, and create a Silver alias.

```
stcl> hcm mkmod -target sync://srvr2.ABCo.com:2647/Projects/DRAM \
                -description "HCM Module for DRAM Block"
stcl> mkfolder DRAM
stcl> cd DRAM
stcl> hcm get -target sync://srvr2.ABCo.com:2647/Projects/DRAM
stcl> exec date > DRAM1
stcl> ci -new -comment "First file in DRAM Block" DRAM1
stcl> hcm release -name R1
stcl> hcm mkalias \
           -target sync://srvr2.ABCo.com:2647/Projects/DRAM@R1 \
           -name Silver
stcl> cd ..
```

The following commands create the Collar module, create a working folder in your local work area (/home/jsmith/Designs) and associate the Collar module with that folder, and create and check a file into the vault associated with the newly created working folder.

```
stcl> hcm mkmod
          -target sync://srvr2.ABCo.com:2647/Projects/Collar \
          -description "HCM Module for Collar"
stcl> hcm mkconf -branch DEV -name DEV \
                 -target sync://srvr2.ABCo.com:2647/Projects/Collar
stcl> mkfolder Collar
stcl> cd Collar
stcl> hcm get -target sync://srvr2.ABCo.com:2647/Projects/Collar@DEV
stcl> exec date > Collar1
stcl> ci -new -comment "First version in DEV Branch for Collar1" \
          Collar1
stcl> cd ..
```

```
The following commands create the Mem2 module, create a working
folder in your local work area (/home/jsmith/Designs), associate the
Mem2 module with that folder, and create and check a file into the
vault associated with the newly created working folder.

stcl> hcm mkmod -target sync://srvr2.ABCo.com:2647/Projects/Mem2 \
                -description "HCM Module for Mem2 Block"
stcl> mkfolder Mem2
stcl> cd Mem2
stcl> hcm get -target sync://srvr2.ABCo.com:2647/Projects/Mem2
stcl> exec date > Mem21
stcl> ci -new -comment "First file in Mem2 Block" Mem21

The following commands create the hierarchy between the Mem2 module
and the Silver configuration of the DRAM module and between the Mem2
module and the default configuration of the Collar module, and fetch
the Mem2 module hierarchy and its files into your local work area.

stcl> hcm addhref \
          -fromtarget sync://srvr2.ABCo.com:2647/Projects/Mem2 \
          -totarget sync://srvr2.ABCo.com:2647/Projects/Collar@DEV \
          -relpath Collar
stcl> hcm addhref \
          -fromtarget sync://srvr2.ABCo.com:2647/Projects/Mem2 \
          -totarget sync://srvr2.ABCo.com:2647/Projects/DRAM@Silver \
          -relpath DRAM
stcl> hcm get -recursive
stcl> cd ..

The following commands create the BIST module, create a working
folder in your local work area (/home/jsmith/Designs) and associate
the BIST module with that folder, create and check a file into the
vault associated with the newly created working folder, and create a
TEST selector configuration of the BIST module.

stcl> hcm mkmod -target sync://srvr2.ABCo.com:2647/Projects/BIST \
                -description "HCM Module for BIST"
stcl> hcm mkconf -name TEST \
                 -target sync://srvr2.ABCo.com:2647/Projects/BIST
stcl> mkfolder BIST
stcl> cd BIST
stcl> hcm get -target sync://srvr2.ABCo.com:2647/Projects/BIST@TEST
stcl> exec date > BIST1
stcl> ci -new -comment "First version in TEST Branch for BIST1" \
         BIST1
stcl> cd ..

The following commands create the hierarchy between the default
configuration of the Block module and the TEST configuration of the
BIST module and between the default configuration of the Block module
and the default configuration of the Mem2 module.

stcl> cd Block
stcl> hcm addhref \
          -fromtarget sync://srvr2.ABCo.com:2647/Projects/Block \
          -totarget sync://srvr2.ABCo.com:2647/Projects/Mem2 \
```

```
          -relpath Mem2
stcl> hcm addhref \
          -fromtarget sync://srvr2.ABCo.com:2647/Projects/Block \
          -totarget sync://srvr2.ABCo.com:2647/Projects/BIST@TEST \
          -relpath BIST
stcl> hcm get -force -recursive \
              -target sync://srvr2.ABCo.com:2647/Projects/Block
stcl> hcm showstatus -recursive
```

The following commands check in an updated Block1 file into the default configuration of the Block module, fetch the modified configuration to your local work area, and release the default configuration of the Block module.

```
stcl> co -lock -comment "Updating Block1" Block1
stcl> exec date > Block1
stcl> hcm put -path /Designs/Block/ \
              -comment "Checking changes into Block Module"
stcl> hcm release -recursive -name R3
```

The following commands remove the BIST module, create the IO2 module, create an IO2 working folder in your local work area (/home/jsmith/Designs) and associate the default configuration of the IO2 module with that folder, add a file to the default configuration of the IO2 module, and create a connection between the default configuration of the Block module and the default configuration of the IO2 module. For a discussion of the hcm rmmod command, see the example in the hcm rmmod command topic.

```
stcl> hcm rmmod -target sync://srvr2.ABCo.com:2647/Projects/BIST \
                -vaultdata
stcl> cd ..

stcl> hcm mkmod -target sync://srvr2.ABCo.com:2647/Projects/IO2 \
                -description "HCM Module for IO2 Block"
stcl> mkfolder IO2
stcl> cd IO2
stcl> hcm get -target sync://srvr2.ABCo.com:2647/Projects/IO2
stcl> exec date > IO21
stcl> ci -new -comment "First file in IO2 Block" IO21
stcl> cd ..

stcl> cd Block
stcl> hcm addhref \
          -fromtarget sync://srvr2.ABCo.com:2647/Projects/Block \
          -totarget sync://srvr2.ABCo.com:2647/Projects/IO2 \
          -relpath IO2
```

Notes:

- The hierarchy in your local work area and the hierarchy on the server will be different until you fetch the default configuration of the Block module into your local work area.
- A hierarchical reference between the default configuration of the Block module and the TEST configuration of the BIST module still exists.

The following hierarchy now exists on the server:

```
Block
   IO2
   Mem2
      DRAM@Silver
      Collar@DEV
```

The following command lists the status of the hierarchical references in your local work area as compared to the server. For a discussion of the hcm showstatus command, see "Example 2" in the hcm showstatus command topic.

```
stcl> hcm showstatus -recursive
stcl> hcm showhrefs -recursive
stcl> hcm rmhref \
          -fromtarget sync://srvr2.ABCo.com:2647/Projects/Block \
          -totarget sync://srvr2.ABCo.com:2647/Projects/BIST@TEST
stcl> hcm get -recursive
stcl> hcm showstatus -recursive
```

Controlling Access to Servers

This comprehensive example provides the commands that set up examples presented in the hcm addlogin, hcm showlogins, and hcm rmlogin command topics of this help system.

The following commands store logins 'queryuser' and 'guest' on DesignSync server sync://chip.ABCo.com:2647.

```
stcl> hcm addlogin -fromtarget sync://chip.ABCo.com:2647 \
        -fromallusers -totarget sync://alu.ABCo.com:2647 \
        -touser queryuser
stcl> hcm addlogin -fromtarget sync://chip.ABCo.com:2647 \
        -fromallusers -toalltargets -touser guest
```

The following command displays the stored logins on the DesignSync server sync://chip.ABCo.com:2647.

```
stcl> hcm showlogins -fromtarget sync://chip.ABCo.com:2647
```

The following commands remove the stored logins from the DesignSync server sync://chip.ABCo.com:2647.

```
stcl> hcm rmlogin -fromtarget sync://chip.ABCo.com:2647 \
        -fromallusers -totarget sync://alu.ABCo.com:2647
stcl> hcm rmlogin -fromtarget sync://chip.ABCo.com:2647 \
        -fromallusers -toalltargets
```

hcm get

hcm get Command

NAME

```
hcm get             - Fetches a legacy module configuration into your
                      work area (legacy)
```

DESCRIPTION

- Identifying the Configuration to Fetch
- Forcing, Replacing, and Non-Replacing Modes
- Module Cache Mode

This command fetches a legacy module configuration and/or updates the configuration in your work area. By default, the hcm get command fetches only the objects for one module configuration. By supplying the -recursive option, you can fetch an entire design hierarchy rooted by the configuration specified.

Note: For non-legacy modules, the functionality of configurations is now available through tags on module versions and branches. To populate non-legacy modules and work areas, use the populate command.

Note: The functionality of configurations is now available through tags on module versions and branches. To populate non-legacy modules and work areas, use the populate command.

To define a design hierarchy, use the hcm addhref command to create hierarchical references (hrefs) between module configurations. The -recursive option causes hcm get to process hrefs and fetch into your work area objects associated with configurations (also called submodules) connected by the hrefs. The relative path value stored with the href to the submodule defines the work area location of the objects for the submodule. For example, the module configuration Chip@Dev has an href to CPU@V1.0 and the relative path is defined as "./subs/CPU". No matter what directory in your work area Chip@Dev is fetched into, a subdirectory structure of "subs/CPU" will be created containing the objects of CPU@V1.0. Relative paths are cumulative as an entire design hierarchy is fetched. In this example, CPU@V1.0 has an href to ADDER@V3.1 at a relative path of "subs/ADDER". When Chip@Dev is fetched recursively, hcm get with the -recursive option creates the subdirectory structure of "subs/CPU/subs/ADDER" with the objects for ADDER@V3.1.

To view the hierarchical references that have been defined for a module configuration, use the hcm showhrefs command. This command also shows the relative paths defined for each hierarchical reference.

This command does not have a dedicated access control action but does obey the DesignSync Checkout access control action. However, when used in module cache mode, the command does not obey the Checkout access control because the checkout is not from the vault

```
but is instead linking to or copying from a local directory.

Notes:
- The hcm get command is not affected by DesignSync symbolic link
  settings.
- The hcm get command supports all DesignSync default fetch states
  except for 'Links to mirror'.
- The hcm get command follows DesignSync registry settings.
- If you get an IP Gear deliverable, HCM creates a record of this
  fetch in IP Gear. (Prior to IP Gear 2.1, the record was in a
  Usage note; as of 2.1, the record of the fetch is in a Download
  note.) The hcm get operation also creates or updates the Usage
  note on the IP Gear server and registers you for IP Gear
  information about the component to which the deliverable is
  attached. (As of IP Gear 2.1, this registration happens
  automatically only if Usage tracking is required for that
  deliverable in IP Gear. If Usage tracking is not required, but
  you want to be registered for updates, you can still register a
  Usage note in IP Gear manually.)

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

Identifying the Configuration to Fetch

```
You can specify which module configuration you want to fetch by
using the hcm get command with the -target option. If you want to
update a configuration that you previously fetched, you can set
your working directory to any directory that is part of the
configuration's data.  Alternatively, you can use the -path option
to specify which working directory to use. If the directory
specified is not the top-level directory containing the
configuration's data (also called the base directory), the command
walks up the directory hierarchy chain to find the base
directory. It is necessary to find the base directory because the
command always fetches the entire configuration's data contents. It
does not work in a piecemeal fashion (that is, it does not fetch
just a subdirectory). You can optimize the time spent checking to
see whether data needs to be fetched by using the -incremental
option.

Understanding Forcing, Replacing, Non-Replacing, and Module Cache
Modes
================================================================
You can fetch a configuration's data into your work area using any
of four modes: forcing, replacing, non-replacing, and module cache
mode. Forcing, replacing, and non-replacing modes all govern how
the get operation updates your work area. Module cache mode
determines whether the configuration's data is fetched from the
module cache or the server.
```

Forcing, Replacing, and Non-Replacing Modes

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

- Forcing mode (specified with the -force option) synchronizes your work area with the incoming data. In this mode, the hcm get command updates the managed objects in your work area. It replaces or removes managed objects regardless of whether the objects have or have not been locally modified and whether the objects are or are not part of the configuration being fetched. Thus, for the configuration, forcing modifies your work area's managed objects to match the set of objects being fetched.

- Replacing mode (specified with the -replace option which is the command's default behavior) synchronizes your work area with the incoming data. In this mode, the hcm get command updates managed objects that have not been locally modified and that are part of the configuration being fetched. It also removes any unmodified managed objects that are not part of the configuration being fetched.

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

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

  In this mode, the hcm get 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 part of the configuration being fetched. Thus, the work area essentially becomes a union of the data from the previous configuration and the configuration being fetched.

  Non-replacing mode, unlike the forcing mode, leaves intact any objects that have been locally modified, and, unlike the replacing mode, leaves unmodified managed objects intact.

  Non-replacing mode also leaves intact managed objects belonging to a referenced submodule configuration even when the reference has been removed. If the reference has been changed to a different configuration of the submodule, then the '-noreplace' operation:
  - Leaves intact managed objects that belong to the previous configuration but not to the new configuration.
  - Updates managed objects that belong to both configurations to the version belonging to the new configuration.

Note: The hcm get command follows the DesignSync registry setting for choosing to delete or retain empty folders during a populate operation (Populate empty directories). The default setting is to remove empty directories during populate. To change the default setting, your Synchronicity administrator can use the SyncAdmin tool. For information, see ENOVIA Synchronicity DesignSync Data

Manager Administrator's Guide.

The behavior of the forcing and replacing modes is similar. The exception is that the replacing mode (-replace option) leaves intact any managed objects with local modifications.

- Specify the -replace option if you want to:
  - Delete unmodified, managed objects that exist in your work area but that are not part of the configuration being fetched.
  - Keep local modifications to managed objects.
- Specify the -force option if you want to:
  - Delete unmodified, managed objects that exist in your work area but that are not part of the configuration being fetched.
  - Delete local modifications to managed objects.

Note:

- Unmanaged objects in your work area are not affected by any of these modes.

- If a configuration in your module hierarchy has been deleted since you fetched that hierarchy into your work area, when you update the hierarchy, the hcm get command removes the objects associated with the deleted configuration based on whether -replace, -noreplace or -force was specified.

Your work areas likely were built by getting module configurations, either individually or as part of an entire design hierarchy. The work area directory into which a configuration's data is placed is called the base directory of the configuration. When updating a configuration in an existing base directory, HCM causes one of several actions to happen:

- The configuration being fetched is the same configuration that was fetched previously. In this case, the configuration's data is refreshed with whatever objects currently define the configuration. This refreshing of objects follows the forcing, replacing, or non-replacing behaviors described above.

  You may have identified the configuration being fetched in this case by the command's options (-target, -path), by using the current working directory, or by virtue of fetching an entire design hierarchy (by using the -recursive option). There are no differences in how either situation behaves when just refetching the same configuration.

- The configuration being fetched is a different configuration of the same module (for example, ALU@V1.1 was fetched previously and now you want to update to ALU@V1.2). In this case, the fetched module's configuration is being switched. The forcing, replacing, or non-replacing behaviors described previously determine the state of the objects after the fetching of the configuration completes.

  If the configuration is being fetched by specifying the command's options (-target, -path), or by using the current working directory, then the -replace option (default behavior) or -force option must be specified to allow the switching of the configuration in the work area. In addition, if the

configuration was previously fetched recursively, then it must be fetched recursively again in order for the base directory to switch between configurations.

If the configuration being fetched is a submodule reached as part of a recursive get operation, then any of the -force, -replace, or -noreplace options is valid and dictates what the state of the objects will be after the fetching completes.

- The configuration being fetched belongs to a different module altogether than the module configuration previously fetched (for example, ALU_VENDOR@V1.0 was fetched previously and now you want to update to ALU_NEW_VENDOR@V4.3).

  The hcm get command does not allow switching between modules created by using the configuration specified with the command's options (-target, -path) or by using the current working directory and the configuration previously fetched into the work area. The -replace and -force options are not relevant in this case.

  If the configuration being fetched is a submodule reached as part of a recursive get operation, then switching modules is allowed. This switch could occur if, for example, an href to a submodule is removed (by the hcm rmhref command) and another href to a different module which uses the same relative path is added to a configuration. Any of the -force, -replace, or -noreplace options is valid and dictates what the state of the objects will be after the fetching completes.

When you use hcm get with the -recursive option, an href may be followed to a submodule that was previously fetched, but the relative path (defined with the href from the upper level module to the submodule) has changed. Thus, the base directory where the configuration's data is to be fetched has changed. In this case, the old base directory (which still is part of your local data but which is no longer part of the design hierarchy) is acted on by following the same basic rules for -force, -replace, and -noreplace behavior.

- -force effectively removes the old base directory (leaving intact local unmanaged objects, if any, and keeping the necessary directories to hold these local unmanaged objects). Any local modifications to managed objects within the old base directory are lost.

- -replace tries to remove the old base directory too, but leaves local unmanaged objects as well as managed objects with local modifications intact. Warnings are output for the objects that could not be removed. The -replace option is turned on by default.

- -noreplace leaves the entire old base directory intact and outputs a warning about not removing the old base directory without using the -replace or -force options.

Regardless of what happens to the old base directory, the

configuration's data is fetched into the newly defined base directory. The new base directory may already exist, and it may even have managed objects within it. That is, it may be populated with DesignSync objects but not HCM module configurations at this point. In any case, the fetching obeys the forcing, replacing, and non-replacing behavior with respect to the objects that may already exist in the new base directory.

A similar situation occurs if an href was removed from a configuration (by the hcm rmhref command) after you have previously fetched the configuration into your work area. In this case, a submodule that is no longer referenced has been previously fetched into the work area. The hcm get command detects the href removal when it fetches the newer configuration that doesn't have the href that it had before. In this case, the base directory holding the configuration that's no longer referenced is acted on by the basic rules for -force, -replace, and -noreplace options as described above. (To determine the status of hrefs in your work area as compared to the module hierarchy defined on the server, use the hcm showstatus command.)

A module configuration commonly has hierarchical references (hrefs) to other configurations such that these configurations reside within the parent configuration's directory "cone" (hierarchy of subdirectories) in your work area. However, a module can have hrefs whose relative paths place referenced configurations outside the parent configuration's directory cone. For example, the Chip@Dev module's hrefs to CPU and ADDER submodules could have relative paths that place CPU and ADDER configurations at the same directory level in your work area as the Chip@Dev module.

If a hierarchical reference to a configuration residing outside the parent directory's cone has been deleted (with hcm rmhref) and if no other hierarchical references to that configuration exist, the hcm get operation removes (purges) the configuration from your work area. Use of the -noreplace, -replace, or -force option determines the behavior of the hcm get operation in handling the removal of such configurations. See "Forcing, Replacing, and Non-Replacing Modes".

Note: The ability to purge configurations that reside outside the parent configuration's directory cone has added with DeveloperSuite V4.1. If your module predates this version, you must delete the parent configuration from your work area (using DesignSync rmfolder and rmfile commands) and then refetch the module configuration hierarchy (with hcm get -recursive) to use this functionality.

Module Cache Mode

In this mode, the hcm get command fetches the target configuration from the module cache instead of the server. Using this mode can help decrease fetch time and save disk space. (Note: To be fetched from the module cache, the module configuration must be an HCM release and must exist in the module cache. To determine if a

release exists in the module cache, use the hcm showmcache command.)

You specify module cache mode by using hcm get with the -mcachemode link or the -mcachemode copy option in combination with the -mcachepaths option. (Note: Module cache mode and module cache paths can each have a default setting, which is specified in the DesignSync registry. When default settings for these options have been specified, you do not have to include them with the hcm get command.) Specifying -mcachemode and -mcachepaths on the command line overrides their default registry values. For more information on default module cache paths and mode, see the "Setting the Default Module Cache Paths or Mode" topic in DesignSync Data Manager HCM User's Guide.

In module cache mode, the hcm get command searches the module cache(s) for releases contained in the hierarchy of the configuration being fetched. When it finds such a release, the get command uses the value of the -mcachemode option (or its default registry setting) to determine how to fetch the release. This option takes three values: link, copy, and server. Link creates a link from your work area to the base directory of the release in the module cache; copy copies the release's contents to your work area. Server fetches the release from the server instead of the module cache.

In module cache mode, the action of the hcm get command depends on the contents of your work area, the contents of the module cache, and the module cache mode you specify.

- If your work area contains previously fetched configurations, you cannot replace them with links to or copies of releases in the module cache, regardless of the -mcachemode value.

  To replace configurations in your work area with links to or copies of releases in the module cache, first use the DesignSync rmfolder -recursive command on the configuration base directories you want to replace. (Note: This command deletes work area directories and their contents. Use it with care.) Then in the same work area, use the hcm get command with the -mcachemode link option to fetch the configuration.

  Instead of replacing the work area content with links, you can create a new (empty) work area directory and then use hcm get with the -mcachemode link option to fetch the configuration you want.

- If you use hcm get with -mcachemode link or -mcachemode copy (or if DesignSync registry settings specify mcachemode link or copy), and for -path you specify a path that already exists in your work area, the hcm get command displays an informational message and fetches the releases from the server.

- If your work area contains links to the module cache, you can replace those links with copies of the releases from the cache. Use hcm get -mcachemode copy.

  You can also replace work area links to the module cache with releases fetched from the server. Use hcm get with the

-mcachemode server option. The hcm get command removes all links to the module cache and fetches the target configuration from the server.

- If a release does not exist in the module cache, the hcm get command fetches the release from the server.
- If you specify hcm get -recursive with -mcachemode link or -mcachemode copy (or if DesignSync registry settings specify mcachemode copy or link) and only the upper-level module of a release exists in the cache, the hcm get command fetches that release from the server. The hcm get command takes this action because the release in the module cache doesn't match the specified action of the hcm get command (-recursive).
- If you specify hcm get (without -recursive) with -mcachemode link or -mcachemode copy (or if DesignSync registry settings specify mcachemode copy or link) and a release's entire hierarchy exists in the module cache, the hcm get command fetches the upper-level module of that release from the server.
- If you specify -keys or -retain with -mcachemode copy or -mcachemode link, (or if DesignSync registry settings specify mcachemode copy or link), the hcm get command ignores the specified -keys or -retain option.

SYNOPSIS

```
hcm get  [-incremental | full] [-keys <key_mode>]  [-merge | edit]
         [-mcachemode <mcache_mode>] [-mcachepaths <path_list>]
         [-path <path>] [-recursive] [-target <configuration_url>]
         [-replace | -noreplace | -force] [-[no]retain]
```

OPTIONS

- -edit
- -force
- -full
- -incremental
- -keys
- -mcachemode
- -mcachepaths
- -merge
- -noreplace
- -path
- -recursive
- -replace
- -retain
- -target

-edit

-edit	Locks the objects contained in the module. This option is only applicable when working with legacy modules with legacy module mode. If -edit is not specified, DesignSync uses the default fetch state to determine whether local copies, links to the DesignSync cache, or DesignSync references are fetched. Notes: - The mirror fetch state is not supported. - You cannot edit releases. If you try to edit a release, the hcm get command fails. - If you run the hcm get command with both the -edit and -recursive options, it locks the objects associated with the upper-level module (i.e., the module configuration specified as the target of the command). It fetches the objects associated with submodules according to DesignSync's default fetch state. - When you lock the contents (objects) of the module, other users cannot lock any objects belonging to that module. This inability to lock applies to DesignSync commands as well HCM commands that attempt to lock objects, for example: co -lock, populate -lock. - The -edit option is mutually exclusive with -merge. - If you specify 'hcm get' with both the -edit and -incremental options, the get operation ignores the -incremental option and performs a full update of the directories in the work area.

-force

-force	Synchronizes your work area's managed data to match exactly the configuration(s) being fetched. See the Description section above for information about the different aspects of the -force option's behavior.

-full

-full	Performs a non-incremental update of the

configuration in your work area by processing all objects and folders whose corresponding vaults have been modified.

The get operation follows the registry setting for the populate operation (Perform incremental populate); this option overrides that registry setting. If you specify neither '-incremental' nor '-full' with the hcm get command and if no registry setting is specified, the get operation performs an incremental update.

The hcm get operation performs an incremental update by default whenever possible, although it automatically reverts to a full update when necessary, such as when the vault or selector changes. For example, if one of the target configuration's hierarchical references has changed, then the hcm get operation performs a full update of the referenced configuration, even though you specified 'hcm get -incremental' on the command line or have a registry setting of populate incremental.

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

-incremental

-incremental

Updates the directories in the work area if the corresponding DesignSync vault folders have been modified since the configuration was brought into the work area.

The get operation follows the registry setting for the populate operation (Perform incremental populate); this option overrides that registry setting. If you specify neither '-incremental' nor '-full' with the hcm get command and if no registry setting is specified, the get operation performs an incremental update.

The hcm get operation performs an incremental update by default whenever possible, although it automatically reverts to a full update when necessary, such as when the vault or selector changes. For example, if one of the target configuration's hierarchical references has changed, then the hcm get operation performs a full update of the referenced configuration, even though you specified 'hcm get

```
                    -incremental' on the command line or have a
                    registry setting of populate incremental.

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

                    Note: If you specify 'hcm get' with both the
                    -incremental and -edit options, the get
                    operation ignores the -incremental option and
                    performs a full update of the directories in
                    the work area.
```

-keys

```
 -keys <key_mode>
                    Controls processing of RCE revision-control
                    keywords.

                    Available modes are:

                    kkv - (keep keywords and values) The local
                    object contains both revision control keywords
                    and their expanded values; for example,
                    $Revision: 1.30$. This is the default
                    behavior.

                    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
                    your 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 hcm get command ignores -keys for
                    any release that is linked to or copied from
                    the module cache.
```

-mcachemode

-mcachemode <mcach_emode>	Specifies how the hcm get command fetches from the module cache. Available modes are: link - For each release it finds in the module cache, the hcm get command sets up a symbolic link from your work area to the base directory of the release 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 get operation fails. copy - For each release it finds in the module cache, the hcm get command copies the release to your work area. Note: This mode is the default mode on Windows platforms. server - Causes the hcm get command to fetch releases from the server, not from the module cache. This option overrides the default module cache mode registry setting. If -mcachemode is not specified, the hcm get 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. See the Description section above for information about the behavior of the "Module Cache Mode" section of the hcm get command operation in module cache mode.

-mcachepaths

-mcachepaths <path_list>	Identifies one or more module cache directories to be searched for releases referenced by the target configuration. Paths may be relative or absolute. 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" 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 get operation fetches releases from the server.

See the Description section above for information about the behavior of the -mcachepaths option.

Notes:
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 command searches the module caches in the order specified with the -mcachepaths option or in the default module cache paths registry setting if this option is absent.

-merge

-merge	Gets the Latest versions from the default configuration or specified branch configuration, and merges them with the current, locally modified versions. By definition, a merge expects a locally modified object, so the -force option is not required. Note: The -merge option is mutually exclusive with -edit.

-noreplace

-noreplace	States to not synchronize your work area's managed data to match the configuration(s) being fetched. It preserves local modifications you may have with the managed data and leaves intact any managed data that is already in the work area but is no longer part of the configuration being fetched. See the Description section above for information about the different aspects of the -replace option's behavior.

-path

-path <path>	Identifies the directory into which the configuration will be brought. If -path is not specified, the hcm get command fetches the configuration into the current directory ("./"). Note: If you use hcm get with -mcachemode link or -mcachemode copy to fetch a release from the module cache and the path already exists in your work area, the hcm get command displays an informational message and fetches the release from the server.

-recursive

-recursive	Fetches into your work area directory the entire design hierarchy of module configurations rooted by the configuration specified for the hcm get command. As noted in the Description section, recursively getting a module hierarchy will automatically create subdirectory structures in your work area holding the data of fetched submodules. The locations of these subdirectories are calculated by the relative path values kept with hierarchical references that define the module hierarchy. By default, the hcm get command does not operate recursively. Note: For hierarchical references with empty relative paths, the hcm get command does not fetch the associated objects.

-replace

-replace	Synchronizes your work area's managed data to match the configuration(s) being fetched but preserves any local modifications you may have with this managed data. See the Description section above for information about the different aspects of the -replace option's behavior.

Note: The hcm get command runs with the -replace option by default.

-retain

-[no]retain	Retains the 'last modified' timestamp of the checked-out objects as recorded when the object was checked into the vault. If you do not specify '-retain' or -noretain' option, the hcm get 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 ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide. Note: The hcm get command ignores -retain if you use it with -mcachemode copy or -mcachemode link (or if DesignSync registry's settings specify mcachemode copy or link).

-target

-target <configuration_url>	Identifies the configuration to bring into your work area. o To specify the default configuration of the module, use the following syntax: 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 vault directory in which the module's data resides. o To specify a configuration other than the default configuration, use the following syntax: sync[s]://<host>[:<port>]/<vaultPath>@<config> where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, and <vaultPath> identifies the vault directory in which the module's data resides and <config> identifies the specific configuration. Note: The specified configuration does not

```
have to be an HCM configuration. You can also
use this option to fetch a DesignSync vault
to your work area.

If the -target option is not specified, the
hcm get command fetches the configuration of
the module associated with the directory
specified by the -path option or the current
working directory (if the -path option is not
specified). The configuration previously
fetched into the directory is fetched again,
possibly updating the work area's objects if
the configuration has changes since the last
time it was fetched. For more information, see
the "Identifying the Configuration to Fetch"
section.

Notes:
- When you set up your work area, you need to
  run the hcm get command with the -target
  option so that HCM stores the target
  information in your work area. The stored
  target information associates the fetched
  configuration with the directory into which
  it was fetched.
- The configuration specified by the -target
  option must exist. If you try to get a
  configuration that does not exist, the hcm
  get command fails.
- If you want to check which configuration was
  fetched into your work area, run the hcm
  showstatus command.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
ci, hcm addhref, hcm release, hcm rmhref, hcm showhrefs,
hcm showstatus, hcm showmcache, keywords, ls, populate, tag
,
```

EXAMPLES

- Initial Fetch of the Upper-Level Module
- Fetching the Entire Hierarchy
- Fetching from the Module Cache
- Updating a Work Area Example Overview
- Example of Updating the Upper-Level Module

- Example of Updating the Entire Hierarchy for a Changed Configuration
- Example of Updating the Entire Hierarchy for a Changed Submodule

Initial Fetch of the Upper-Level Module

```
The following examples guides you through the set up of the following
hierarchy in your work area:

Chip
   Cpu
   Alu@R1
     Dpath@R2

This example fetches the default configuration of the Chip module
into the current directory, Designs/Chip.

Note:
- Because this is the first time you are setting up your work area, you
  need to run the hcm get command with the -target option.

dss> hcm get -target sync://chip.ABCo.com:2647/Projects/Chip

By default, the hcm get command is not recursive; that is, it only
fetches the objects associated with the upper-level module. In this
example, the hcm get command fetched the objects associated with the
Chip module's default configuration into your work area and stored the
target information. To check which configuration was fetched into your
work area, run the hcm showstatus command.
```

Fetching the Entire Hierarchy

```
The following example recursively fetches the default configuration of
the Chip module into the current directory, Designs/Chip, creating the
additional levels of hierarchy and fetching the objects associated with
the Chip module and referenced submodules, Cpu, Alu, and Dpath.

Note:
- If this is the first time you are setting up your work area, you need
  to run the hcm get command with the -target option. If you previously
  fetched the Chip module into the current directory, as in Example 1a,
  you do not need to specify the -target option.

dss> hcm get -recursive \
             -target sync://chip.ABCo.com:2647/Projects/Chip

In this example, the hcm get command fetches Cpu into the relative
path Cpu and Alu@R1 into the relative path Alu, which in turn fetches
Dpath@R2 into Alu/Dpath in your work area.

The following hierarchy now exists in your work area:

Chip                kept in directory Designs/Chip
```

```
   Cpu              kept in directory Designs/Chip/Cpu
   Alu@R1           kept in directory Designs/Chip/Alu
     Dpath@R2       kept in directory Designs/Chip/Alu/Dpath
```

To view the directory structure created from the relative paths, run the hcm showhrefs command.

Fetching from the Module Cache

The following example recursively fetches the default configuration of the Chip module into the current directory, Designs/Chip. Modules are fetched from the HCM server, except for the Alu@R1 release, which is fetched from the module cache that has been set up at /home/ChipDev/cache1.

Note: This example shows a first time setup of a work area. The get operation does not replace work area content with links to or copies of releases in the module cache.

```
dss> hcm get -recursive \
             -target sync://chip.ABCo.com:2647/Projects/Chip \
             -mcachemode copy \
             -mcachepaths /home/ChipDev/cache1
```

In this example, the hcm get command searches the module cache for any releases that match the specified target or that are submodules of the target. The entire hierarchy of the Alu@R1 release exists in the cache, so the get command copies that release's hierarchy to the work area. Other submodules (those that are not releases or that are releases but not in the module cache) are fetched from the HCM server.

Updating a Work Area Example Overview

Since you initially fetched the Chip module into your work area, the ALU module has been finalized. The following examples guide you through updating the hierarchy in your work area to match this hierarchy:

```
Original Hierarchy   Hierarchy for 2a and 2b   Hierarchy for 2c
Chip                 Chip                      Chip
   ALU@Silver           ALU@Golden                ALU@Golden
     DPATH@Golden         DPATH@Golden              DPATH2@TEST
```

The Silver configuration of ALU contains the following files: File1, File2, and File3, of which File1 and File2 have been locally modified. The Golden configuration of ALU contains the following files: FileA, FileB, and FileC.

The Golden configuration of DPATH contains the following files: FileD1 and FileD2, of which FileD1 has been locally modified. The TEST configuration of DPATH2 contains the following files: FileDA and FileDB.

Example of Updating the Upper-Level Module

```
The following example updates the default configuration of the Chip
module.

Note:
- As you previously fetched the Chip module into the current directory,
  you do not need to specify the -target or -path option. If the
  current working directory was something other than Designs/Chip, use
  the -path option to specify Designs/Chip as the location to fetch
  the configuration's data.

dss> hcm get

By default, the hcm get command is not recursive; that is, it only
updates the files associated with the upper-level module. In this
example, the hcm get command updated the files associated with the
Chip module into your work area. It did not update the files
associated with the submodules.

To check that the relative paths have not changed, run the hcm
showhrefs command.
```

Example of Updating the Entire Hierarchy for a Changed Configuration

```
The following example recursively updates the default configuration of
the Chip module in the current directory, Designs/Chip; that is, the
files associated with the Chip module and referenced submodules ALU and
DPATH are updated.

Notes:
- As you previously fetched the Chip module into the current directory,
  you do not need to specify the -target or -path options.
- The design hierarchy being fetched now has the default configuration
  of Chip referencing ALU@Golden instead of referencing ALU@Silver.
  ALU@Golden still contains a reference to DPATH@Golden - which is
  already fetched into the work area. Assume that the default
  configuration of Chip was updated by someone else on your team to
  remove the href to ALU@Silver (via the hcm rmhref command) and add
  an href to ALU@Golden (via the hcm addhref command).

dss> hcm get -recursive

In this example, the hcm get command updates the files associated with
the Chip module and the referenced submodules. The hcm get command
modifies files in the ALU subdirectory as follows:
- Deletes File3
- Adds FileA, FileB, and FileC
- Leaves File1 and File2

By default, the hcm get command does not remove files that have been
```

locally modified. This behavior is due to the default -replace option being turned on. As File1 and File2 were locally modified, the hcm get command does not remove these files from the ALU subdirectory even though they are not associated with the Golden configuration of the ALU module. If you want the hcm get command to remove files that have been locally modified, you need to specify the -force option.

There were no files updated for the DPATH module in the work area. This is because DPATH@Golden was already fetched into the work area and the -force option was not specified. If you want the hcm get command to update the file, FileD1, that was locally modified, you need to specify the -force option.

Note: You will never lose your local modifications if you run hcm get with the default update behavior (-replace option) or with the -noreplace option. Only the -force option is destructive.

To check that the relative paths are as expected, run the hcm showhrefs command. To check the status of the files in your work area as compared to the server, run the ls command.

Example of Updating the Entire Hierarchy for a Changed Submodule

The following example recursively updates the default configuration of the Chip module in the current directory, Designs/Chip; that is, the files associated with the Chip module and referenced submodule ALU are updated. It also creates the DPATH2 subdirectory and fetches the files associated with the DPATH2 submodule.

Notes:
- As you previously fetched the Chip module into the current directory, you do not need to specify the -target or -path options.
- Before the hcm get command can fetch the updated hierarchy into your work area, the hierarchical reference between the ALU@Golden configuration and the DPATH@Golden module needs to be removed and the hierarchical reference between the ALU@Golden module and the DPATH2@TEST configuration needs to be added. This reference to DPATH2@TEST has a relative path of "./DPATH2". Assume for this example that someone else has taken these steps with the hcm rmhref and hcm addhref commands.

```
dss> hcm get -recursive
```

In this example, the hcm get command updates the files associated with the Chip module and the referenced submodules, and creates the Designs/Chip/ALU/DPATH2 subdirectory. The hcm get command updates the Designs/Chip/ALU/DPATH subdirectory as follows:
- Deletes FileD2
- Leaves FileD1

Because there is still a file in the Designs/Chip/ALU/DPATH subdirectory, the hcm get command does not remove the directory. Finally, it fetches files FileDA and FileDB into the Designs/Chip/ALU/DPATH2 directory.

```
The following hierarchy now exists in your work area:

Chip                kept in directory Designs/Chip
   ALU@Golden       kept in directory Designs/Chip/ALU
     DPATH2@TEST    kept in directory Designs/Chip/ALU/DPATH2

By default, the hcm get command does not remove files that have been
locally modified. As FileD1 was locally modified, the hcm get command
did not remove this file from the Designs/Chip/ALU/DPATH subdirectory
nor did it remove this subdirectory even though DPATH is not associated
with the Golden configuration of the ALU module. If you want the hcm get
command to remove files that have been locally modified, you need to
specify the -force option.

To check that the relative paths are as expected, run the hcm showhrefs
command. To check the status of the files in your work area as
compared to the server, run the ls command.
```

hcm mkalias

hcm mkalias Command

NAME

```
hcm mkalias          - Creates an alias for a release of a module
                       (legacy)
```

DESCRIPTION

```
This command creates an alias (a symbolic name) or modifies an existing
alias for a release. An alias can be moved from one release to another.
For example, "Golden" can initially refer to release R1. Later, after
release R2 has been thoroughly tested and approved, the alias can be
changed to refer to R2. Aliases can be used in a Synchronicity URL. For
example, you can create a hierarchical reference to an alias of a
submodule. When an HCM command processes an alias, it resolves the
alias to the release to which it points and then operates on the
release.

Notes:
- This command obeys the ProjectSync CreateConfig and ModifyConfig
  access controls. If you are trying to create an alias and you do not
  have permission to create a configuration, this command returns an
  error. If you are trying to modify an existing alias and you do not
  have permission to modify a configuration, this command returns an
  error.
-  For non-legacy module, the functionality of aliases is available
   through tags.

For this command to be successful, the following must be true:
```

```
- The target specified with the -target option must exist and must
  correspond to a release.
- The server on which the release resides must be available.
- The name of the alias must be unique within the scope of the module.
  o When creating aliases, the name of the alias cannot match the name
    of an existing alias, branch configuration, selector configuration,
    or release for the module.
  o When changing aliases, the name of the alias must match the name of
    an existing alias for one of the module's releases.
```

SYNOPSIS

```
hcm mkalias [-description <description>] -name <alias_name>
            -target <release_url>
```

OPTIONS

- -description
- -name
- -target

-description

```
-description           Provides a text description of the alias. To
 <description>         specify a description, enclose the text
                       within double quotation marks (""). The
                       default description is "HCM alias".

                       Note: You can use ProjectSync to change the
                       description for your alias.
```

-name

```
-name <alias_name>     Specifies the name of the alias.

                       Notes:
                       - When you create an alias, HCM attaches the
                         ProjectSync notes attached to the release to
                         new alias.
                       - To change the release to which an alias
                         refers, run the hcm mkalias command and
                         specify a new target. The hcm mkalias
                         command changes the release that the alias
                         references.
                       - When you change the release to which an alias
                         refers, HCM updates the ProjectSync note
                         attachments accordingly.
```

```
                        Rules for Alias Names:
                        - They must be unique within the scope of the
                          module. For new aliases, names cannot match
                          the name of an alias, branch configuration,
                          selector configuration, or release for the
                          module. For existing aliases, names must match
                          the name of an alias for one of the module's
                          releases.
                        - They can contain letters, numbers,
                          underscores (_), periods (.), and
                          hyphens (-). Other characters, including
                          whitespace, are prohibited.
                        - They 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 alias name and version/branch
                          dot-numeric identifiers.
                        - They 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.
                        - They should not start with 'Sync'(case-
                          insensitive) because Synchronicity may
                          define new keywords in the future using that
                          naming convention.
                        - They cannot end with '--R'. This tag is
                          reserved for use by HCM.
```

-target

```
 -target <release_url>  Identifies the release for which you want to
                        create an alias.

                        Specify the URL as follows:
                  sync[s]://<host>[:<port>]/<vaultPath>@<ReleaseName>
                        where <host> is the SyncServer on which the
                        module resides, <port> is the SyncServer port
                        number, and <vaultPath>@<ReleaseName> identifies
                        the release for which you want to create an
                        alias.
```

RETURN VALUE

```
 This command does not return Tcl values.
```

SEE ALSO

```
 hcm mkconf, hcm showconfs, hcm release, hcm rmalias
,
```

EXAMPLES

```
 The following example creates an alias 'Golden' to the released
 configuration of Chip@R1.

 "Creating, Fetching, and Releasing Module Hierarchies" in the hcm
 Example topic includes the steps for creating the Chip module and the
 R1 release.

 dss> hcm mkalias -target sync://chip.ABCo.com:2647/Projects/Chip@R1 \
                  -name Golden \
                  -description "Ready for distribution to other teams"

 The following example changes the release to which the alias 'Golden'
 points from the R1 configuration to the R2 configuration.

 dss> hcm mkalias -target sync://chip.ABCo.com:2647/Projects/Chip@R2 \
                  -name Golden \
                  -description "Ready for distribution to other teams"
```

hcm mkconf

hcm mkconf Command

NAME

```
 hcm mkconf          - Creates a legacy configuration for a module on
                       a server (legacy)
```

DESCRIPTION

```
 This command creates a configuration of an HCM module on a server and
 a ProjectSync configuration. An HCM configuration is a group of managed
 objects, each at a particular version.

 For non-legacy modules, the functionality of configurations is now
 available through tags on module versions and branches.

 There are four types of HCM configurations, depending on how the
 configuration was created. Two types (branch and selector) are
 created with hcm mkconf; two types (default and release) are
 created with other HCM commands.

 o Branch configuration. A branch configuration is a group of
   managed files that have a selector of <branch_name>:Latest.
```

A branch configuration is created by using the hcm mkconf command with the -branch option. This type of configuration can be removed from the server by using the hcm rmconf command.

o Selector configuration. A selector configuration is a group of managed files, each at a particular version. Selector configurations enable the full power of DesignSync selector lists. In addition, ProjectSync configurations are represented in HCM as selector configurations. A selector configuration is created by using the hcm mkconf command with the -selector option. This type of configuration can be removed from the server by using the hcm rmconf command.

o Default configuration. A default configuration is a branch configuration in which files have the selector Trunk:Latest (a default branch selector of Trunk and version selector of Latest). A default configuration is associated with a module by default. A default configuration is created by using the hcm mkmod command. This type of configuration can be removed from the server by using the hcm rmmod command. (Note: In HCM operations, when you specify a module without a configuration name, the HCM operation uses the module's default configuration.)

o Release. A release is a group of managed files, fixed at a particular version, that cannot be modified. A release is created with the hcm release command. (Note: Although the contents of a release cannot be changed, the release can be removed from the server with the hcm rmmod operation. This operation removes a module's configuration definitions as well as the module itself.)

Note:
- This command obeys the ProjectSync CreateConfig access control; if you do not have permission to create a configuration, this command returns an error.

For this command to be successful, the following must be true:
- The module specified by -target must exist.
- The server on which the module resides must be available.
- The name of the configuration must be unique within the scope of the module. It cannot match the name of an alias or release for that module.

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

SYNOPSIS

```
hcm mkconf [-description <description>] -name <configuration_name>
           [-selector <selector_list> | -branch <DS_branch_name>]
            -target <module_url>
```

OPTIONS

- -branch
- -description
- -name
- -selector
- -target

-branch

-branch <DS_branch_name>	Indicates that a branch configuration should be created. Specifies a DesignSync branch name that identifies the contents of the configuration. Note: - HCM automatically adds ':Latest' to the end of the supplied branch tag. - Branch names cannot end with "--R".

-description

-description <description>	Provides a text description of the configuration. To specify a description, enclose the text within double quotation marks (""). The default description is "HCM configuration". Note: You can use ProjectSync to change the description for your configuration.

-name

-name <configuration_name>	Specifies the name of the configuration. Names of Configurations: - Must be unique within the scope of the module. They cannot match the name of an alias or release for that module. - Can contain letters, numbers, underscores (_), periods (.), and hyphens (-). 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 configuration 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.
                        - Should not start with 'Sync'(case-
                          insensitive) because Synchronicity may
                          define new keywords in the future using that
                          naming convention.
                        - Cannot end with '--R'. This tag is
                          reserved for use by HCM.

                        Note:
                        - If the -branch and -selector options are
                          absent, HCM creates a selector configuration
                          with the value of the selector being the same
                          as the configuration name.
```

-selector

```
 -selector              Indicates that a selector configuration should
  <selector_list>       be created. Provides a comma-separated list of
                        selectors that identifies the contents of the
                        configuration. For information about selector
                        lists, see the "selectors" help topic.

                        To specify a selector configuration, use the
                        following syntax:
                          <branch>[:<version>]

                        Notes:
                        - If the selector value contains '--R', HCM
                          considers the configuration a release or an
                          alias. Using the hcm mkconf command to create
                          a release or an alias is not good practice.
                          Instead, use the hcm release and hcm mkalias
                          commands, respectively.
                        - If you specify -selector <branch>:Latest, the
                          mkconf operation creates a branch
                          configuration instead of a selector
                          configuration.
```

-target

```
 -target <module_url>   Identifies the module for which you want to
                        create a configuration.

                        Specify the URL as follows:
                         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
                        for which you want to create a configuration.
```

```
                    Note:
                    -target must resolve to a module. Specifying
                    just the SyncServer or specifying a
                    configuration will result in an error.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
ci, co, hcm addhref, hcm mkalias, hcm mkmod, hcm showconfs,
hcm release, hcm rmconf, mkbranch, selectors, setselector, tag
,
```

EXAMPLES

- Creating a Branch Configuration
- Creating a Selector Configuration

Creating a Branch Configuration

```
This example creates a DEV configuration of the Mem module.

"Creating and Removing Module Configurations" in
the hcm Example topic includes the steps for creating the
Mem module and DEV branch.

dss> hcm mkconf -branch DEV \
                -name DEV \
                -target sync://srvr1.ABCo.com:2647/Projects/Mem \
                -description "Config. for Mem development work"
```

Creating a Selector Configuration

```
This example creates the T103 configuration of the IO module,
which contains files tagged with "TEST" or "Dev1.03" tags.

"Creating and Removing Module Configurations" in the hcm Example
topic includes the steps for creating the IO module and tagging files
with a "TEST" tag.

dss> hcm mkconf -name T103 -selector Trunk:TEST,Trunk:Dev1.03 \
                -target sync://srvr1.ABCo.com:2647/Projects/IO \
                -description "TEST and Dev1.03 files"
```

hcm mkmod

hcm mkmod Command

NAME

```
hcm mkmod           - Creates an HCM module on a server (legacy)
```

DESCRIPTION

- Controlling the Ability to Create Modules

```
This command creates an legacy (HCM) module on a server. An HCM
module is data that represents a level of the design hierarchy. The
data that makes up a module includes its own objects and references
to other modules, but not the contents of the referenced modules. For
help creating non-legacy modules, see the mkmod command.

When you create your module, you can associate it with existing vault
data. If you do not associate the module with existing vault data, the
mkmod operation creates the vault directory. The appropriate vault
subdirectories are created when you first check data into the vault.

Note: The hcm mkmod command does not allow you to embed one module
within another. The command does not allow the creation of a new
module if:
o The new module's vault lies within the vault of an existing module.
o The new module's vault contains the vault of an existing module.

When you use the hcm mkmod command on a vault folder, in addition to
creating an HCM module from the specified vault folder, the hcm mkmod
command:
- Updates existing email subscriptions associated with the vault path
  that is being converted into a module. As the email subscriptions
  only apply to the newly created module, not its submodules, HCM sets
  the Scope field of the subscriptions to "This object only". HCM also
  sets the Scope field to "This object only" for existing
  subscriptions to any configurations of the module.
- For each existing subscription associated with the vault path,
  creates a new subscription covering the entire vault structure below
  the vault folder that is being converted into a module.

For information about email subscriptions, see the HCM Help topic, "The
CPU Team Subscribes to Email on a Hierarchy."

If the hcm mkmod command encounters a DesignSync REFERENCE line from
pre-HCM use, the output identifies the REFERENCE, and if the criteria
for success were satisfied, the command creates the module.

For this command to be successful, the following must be true:
```

```
- The server on which the module will be created must be available.
```

Controlling the Ability to Create Modules

```
The ability to create a module can be access-controlled. By
default, any user can create a module. If this policy is not
acceptable for your project, you (as the administrator or project
leader) can restrict or deny this capability for your site or
individual SyncServers. For more information, see the Access
Control Guide Help. See also information on the access allow and
access deny commands.

Note:
- This command obeys both DesignSync's MakeFolder access control and
  ProjectSync's AddProject access control; if you do not have
  permission to make a folder and add a project, this command displays
  an error.
```

SYNOPSIS

```
hcm mkmod [-description <description>] -target <module_url>
```

OPTIONS

- -description
- -target

-description

```
-description          Provides a text description of the module.
 <description>        To specify a description, enclose the text
                      within double quotation marks ("). The
                      default description is "HCM module".

                      Notes:
                      - If you specify -description and your module
                        resides in /Projects/<moduleName>, the
                        ProjectSync project that HCM creates for the
                        module will include the description.
                      - The hcm mkmod command does not overwrite
                        existing descriptions.
                      - You can use ProjectSync to change the
                        description for your module.
```

-target

```
-target <module_url>   Specifies the URL of the server vault hierarchy
                       where the module will be created.

                       Specify the Synchronicity URL as follows:
                        sync[s]://<host>[:<port>]/<vaultPath>
                       where <host> is the SyncServer on which the
                       module will reside, <port> is the SyncServer
                       port number, and <vaultPath> identifies either
                       an existing vault directory in which objects
                       associated with the module reside or a
                       new vault directory that hcm mkmod will create.

                       Vault paths:
                       - Should be /Projects/<moduleName>
                         where <moduleName> can contain letters,
                         numbers, underscores (_), periods (.), and
                         hyphens (-). All other characters, including
                         whitespace, are prohibited.

                         It is recommended that you place modules in
                         the Projects directory. Locating modules in
                         this directory enables the greatest use of
                         ProjectSync features. When your vault path is
                         of the form /Projects/<moduleName>, HCM
                         creates a ProjectSync project for the module.

                       - Cannot be /Projects. Modules can be created
                         in the Projects directory, for example,
                         sync://chip.ABCo.com:2647/Projects/Chip.

                       - Cannot be or be within any of the following:
                         o /Backup.sync
                         o /Export.sync
                         o /Import.sync
                         o /Modules
                         o /Partitions

                       - Must be unique with respect to the server. You
                         can have multiple modules on the same server
                         with the same name, with different vault
                         paths. For example, the following are unique
                         vault paths:
                         o sync://chip.ABCo.com:2647/Projects/Chip
                         o sync://chip.ABCo.com:2647/Chip

                       - Cannot specify creation of embedded
                         modules. The hcm mkmod command does not
                         create a module if:
                         o The new module's vault lies within the
                           vault of an existing module. For example,
                           the command fails if you specify
                           sync://cpu.ABCo.com:2647/Projects/Cpu/Ram
                           as the -target and the Cpu module already
                           exists.
                         o The new module's vault contains the
                           vault of an existing module. For example,
```

```
                        the command fails if you specify
                        sync://cpu.ABCo.com:2647/Projects/Cpu/Alu
                        as the -target and the Alu module
                        already exists.

                    This command derives the name of the module from
                    the leaf of the vault path. For example, if you
                    specify
                    sync://alu.ABCo.com:2647/Projects/Alu for
                    -target, with Projects/ALU being the vault path,
                    the command creates a module named Alu.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
hcm mkconf, hcm release, hcm rmmod, hcm showmods
,
```

EXAMPLES

```
This example creates a Chip module.

dss> hcm mkmod -target sync://chip.ABCo.com:2647/Projects/Chip \
               -description "HCM Module for Chip Block"

To view this command in the context of an extended example, refer
to "Example 1: Creating, Fetching, and Releasing Module Hierarchies"
in the hcm Example topic.
```

hcm put

hcm put Command

NAME

```
hcm put             - Checks in a configuration (legacy)
```

DESCRIPTION

- Understanding Handling of Symbolic Links
- Controlling the Ability to Check In Configurations

This command checks in a configuration; specifically it checks in all locally modified, managed objects associated with the configuration. If you run this command recursively, the command checks in all of the objects associated with the configuration and with all of the configuration's hierarchical references.

For non-legacy modules, the functionality of configurations is available through tags on module versions and branches and to check in module changes, you use the ci command, as for any other design data. For more information, see ci.

By default, this command removes the locks from locally modified objects when they were checked in and from unmodified, managed objects locks that were previously locked. When run with the -lock option, this command:
- Retains the locks for all objects that were locked for edit prior to check in.
- Adds locks to locally modified objects that were not previously locked but were checked in.

This command also generates a RevisionControl note for each configuration that is checked in. The generated RevisionControl note is attached to both the configuration and the owning module. The RevisionControl note identifies both the objects that were successfully checked in and any objects for which check in failed.

The hcm put command will not check in the following:
- Referenced configurations that are releases, aliases, or IP Gear deliverables.
- Referenced configurations for which the 'Put' access control check fails.
- Referenced configurations that do not exist on the server. If a configuration was removed from the server after you fetched it to your work area, you will be unable to check in objects associated with it.
- Referenced configurations for which none of the associated objects have been locally modified.

The hcm put command shows the check-in results for each configuration and summarizes the check-in failures after it completes. This summary allows you to identify why a check in for a given object failed (for example, attempt to check in a file that is locked by another user) and correct the situation. When you run the hcm put command recursively, the summary identifies failed check ins for all levels of the hierarchy. For information on the ci command, refer to the ci command topic.

For this command to be successful, the following must be true:
- The configuration to be checked in must
  - Exist on the server
  - Be a default, branch, or selector configuration
- Locally modified, managed objects must be either locked for edit or, if not locked for edit, the latest version on a branch.
- The server(s) on which the configurations reside must be available and must have HCM 1.1 or higher installed.
- The mirror fetch state is not set.

Notes:
- To add an unmanaged object to a configuration, check the object into the vault associated with the configuration by running the ci command with the -new option.
- Whether or not a checked-in object is kept locally depends on your DesignSync default fetch state. For more information, refer to the "fetch preference" topic.
- If you fetched the objects belonging to a selector configuration with a version tag, you need to move the tags associated with earlier versions to the newly checked-in versions to update the configuration with the newer versions. When the hcm put command finishes checking in the objects associated with the configuration, it provides a list of the selector configurations that were checked in. You can use this list to identify those configurations for which the version tags may need to be moved.
- The hcm put command skips objects that were removed from the work area with the rmfile command and recreated outside of DesignSync (that is, the new file is not checked in with DesignSync using the ci command).

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

Understanding Handling of Symbolic Links

The hcm put command checks in symbolic links if the following conditions are met:
- The link is already a managed object.
- The link has been modified.

If these conditions are met, the hcm put command checks in the link.

The hcm put command does not traverse directory links. Directory links could point to directories containing objects that do not live under the vault folder of the configuration, preventing correct maintenance of the configuration content.

The hcm put command, when operating on symbolic links to files and if dereference links setting is used, replaces the managed link with the contents of the files to which it points. When operating on symbolic links to directories, the command does not obey the dereference links setting. For more information, see the ENVOIA Synchroncity DesignSync Data Manager HCM User's Guide topic, "Overview of Administration Tasks."

Controlling the Ability to Check In Configurations

The ability to check in configurations can be access-controlled. By default, all users can check in configurations. If this policy is not acceptable for your project, you (as the administrator or project leader) can control this capability for your site or

```
individual SyncServers. For more information, see the Synchronicity
Access Control Guide. See also information on the access allow and
access deny commands.

Notes:
- This command obeys the HCM 'Put' access control. If you are
  trying to check in a configuration and you do not have permission to
  do so, this command does not check in the objects belonging to that
  configuration, continues processing the remaining configurations,
  and reports all check in failures after completion.
- This command performs an access control check for each configuration
  checked in.
- When you check in a configuration, the hcm put command uses the ci
  command to check in the objects associated with the configuration.
  The ci command obeys the 'Checkin' access control and applies the
  control to each object being checked in. If you are trying to check
  in an object and you do not have permission to do so, that object
  is not checked in and the hcm put command reports the failure.
```

SYNOPSIS

```
hcm put [-comment "<text>" | -nocomment] [-keys {kkv | kk | kv | ko}]
        [-lock] [-path <path>] [-recursive] [-retain]
```

OPTIONS

- -comment
- -keys
- -lock
- -nocomment
- -path
- -recursive
- -retain

-comment

```
-comment "<text>"      Checks in the specified objects with a
                       description attached. Enclose the description
                       in double quotes if it contains spaces. This
                       comment obeys the DesignSync minimum comment-
                       length settings. For more information, see
                       the SyncAdmin Help topic, "The General Tab -
                       Administrator View."

                       If you do not specify either -comment or
                       -nocomment, the hcm put command prompts you to
                       enter a check-in comment.
```

-keys

-keys <key_mode>	Controls processing of RCE revision-control keywords. Available modes are: kkv - (keep keywords and values) The local object contains both revision control keywords and their expanded values; for example, $Revision: 1.30$. This is the default behavior. 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 your 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: - To run the hcm put command with the -keys option, you must either: o Also specify the -lock option, or o Have DesignSync default fetch state set to 'get'. For more information on fetch state, refer to the "fetch preference" topic.

-lock

-lock	Locks all the objects associated with the configuration that were successfully checked in. Retains locks on the objects for which locks are already held, whether those objects have been locally modified or not. If the hcm put command is run without the -lock option, the command removes locks from o Locally modified objects that were checked in

o Unmodified objects that were previously locked

-nocomment

-nocomment	Checks in the specified objects without a description of changes. If the object was checked out with comments ("co -comment"), the check-out comments are retained during check in. If you do not specify either -comment or -nocomment, the hcm put command prompts you to enter a check-in comment.

-path

-path <path>	Identifies the directory from which to begin checking in objects. If the directory specified is not the top-level directory containing the configuration's data (also called the base directory), the command walks up the directory hierarchy chain to find the base directory. If the hcm put command fails to find a base directory, it stops processing and displays an error. If -path is not specified, the hcm put command begins checking in objects from the current directory ("./").

-recursive

-recursive	Checks in the objects associated with the configuration and with all of the configuration's hierarchically referenced components. By default, the hcm put command does not operate recursively. Notes: - The hcm put command retrieves the hierarchical references from the local metadata, not from the server. As a result, it checks in the configuration hierarchy that was fetched by the hcm get command. - When run recursively, if the hcm put command encounters an error anywhere within the hierarchy, it displays an error and continues processing the rest of the hierarchy.

-retain

```
-retain                Retains the 'last modified' timestamp of the
                       objects that remain in your work area. By
                       default (without -retain), the hcm put command
                       updates the timestamp of the local object to
                       the check-in time.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
access allow, access deny, cancel, ci, hcm get, hcm release, unlock,
tag
,
```

EXAMPLES

```
This example checks in changes to the Block1 file into the default
configuration of the Block module.

The "Creating, Modifying, and Putting Configurations" example in the
hcm Example topic includes the steps for creating the Block module
and the Block1 file.

dss> hcm put -path /home/jsmith/Designs/Block/ \
             -comment "Checking changes into Block Module"
```

Description

hcm release Command

NAME

```
hcm release          - Creates a configuration that cannot be modified
                       (legacy)
```

DESCRIPTION

- Controlling the Ability to Release Configurations

- Understanding the State of Your Work Area State After a Release
- Releasing Submodules
- Preparing to Release Your Work Area

This command creates a configuration that cannot be modified (a release) for one or more modules. This command performs recursively; that is, it creates configurations for the upper-level module (module associated with the current directory or the work area specified by the -path option) and its submodules. If you want to release just the upper-level module, run the hcm release command with the -norecursive option. If you perform a nonrecursive release, the hcm release command creates a release that has no hierarchical references.

For non-legacy modules, the functionality of configurations is now available through tags on module versions and branches. For more information, see tags.

Notes:
- Only configurations that exist in your work area can be released.
- Release names must be unique within the scope of the module. If the module has an alias, branch configuration, selector configuration, or release with the same name as specified by -name, the release operation will fail.
- The hcm release command fails if it is operating recursively and a lower-level module does not exist in the work area.
- Released configurations cannot be changed. For example, you cannot run hcm get with the -edit option or run hcm addhref on releases. (Note: Although the contents of a release cannot be changed, the release can be removed from the server with the hcm rmmod operation. This operation removes a module's configuration definitions as well as the module itself. See the hcm rmmod command topic.)
- The execution of the hcm release command could fail to complete; for example, if a referenced server is unavailable or the release of a submodule failed. HCM recognizes that the release was not completed, notifies you, and permits you to rerun the command.

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

Controlling the Ability to Release Configurations

The ability to create a release can be access-controlled. By default, any user can create a release. If this policy is not acceptable for your project, you (as the administrator or project leader) can restrict or deny this capability for your site or individual SyncServers. For more information, see the Synchronicity Access Control Guide. See also information on the access allow and access deny commands.

Understanding the State of Your Work Area State After a Release

The hcm release command does not alter the state of your work area in any way. If you fetched a branch configuration of a module into your work area and then release it, your work area still contains the branch configuration of the module. This behavior is desirable in that you probably want to continue making changes to the configuration and then to make subsequent releases of the changes.

If you want to switch your work area to reflect the release instead of the branch configuration, run the hcm get command to fetch a release and replace the branch configuration. The hcm get command allows you to switch the configuration of a module that already has been fetched into your work area. "Example 1: Creating, Fetching, and Releasing Module Hierarchies" in the hcm Example topic shows how you can switch your work area.

Releasing Submodules

By default, the hcm release command performs a recursive release. When a recursive release operation encounters a submodule, it does the following:

- For submodules that have been released, it creates a new reference. The new reference is from the new release of the upper-level module to the referenced release configuration of the submodule.

- If the submodule is referenced from the upper-level module using an alias (for example, Mem2 contains a hierarchical reference to DRAM@Silver and Silver is an alias to the DRAM@R1 configuration), a new release of DRAM is not created. However, the hierarchical reference created in the release of Mem2 will resolve the alias. Thus, the release of Mem2 will contain a reference to DRAM@R1 and not DRAM@Silver. ("Example 2: Understanding the Necessity of Fetching All Configurations" in this topic illustrates the release of an aliased configuration.)

- For branch or selector configurations, the release operation descends into the submodule and releases it. It creates a new hierarchical reference from the released upper-level module to the released submodule.

- During a recursive release operation, if a hierarchical reference is encountered that points to a vault path that has not been defined as an HCM module (by using the hcm mkmod command), this situation is treated as though the reference is to a branch configuration. Thus, the hcm release command descends into the vault path, implicitly defines the path as an HCM module, and creates a release for the module.

- The submodule that is released is not required to exist on the same server that contains the upper-level module that references the submodule. The hcm release command handles the communication and coordination of releasing submodules on any number of servers. The release of the submodule is created on the same server that contains

```
  the branch configuration of the submodule.

- The release of the submodule must have a name (as does any
  configuration). To name the submodule's release uniquely, a unique
  configuration name is generated. This unique name is formed by
  concatenating the names of all the upper-level modules and prefixing
  the name produced to the name specified by the -name option. For
  example, consider the following hierarchy and the effect produced by
  the invocation of hcm release:

    sync://chip.ABCo.com:2647/Projects/Chip@DEV
      sync://cpu.ABCo.com:2647/Projects/CPU@DEV
        sync://alu.ABCo.com:2647/Projects/ALU@DEV

    dss> hcm release -name R1

  The following releases are created:

     sync://chip.ABCo.com:2647/Projects/Chip@R1
       sync://cpu.ABCo.com:2647/Projects/CPU@Chip.R1
         sync://alu.ABCo.com:2647/Projects/ALU@Chip.CPU.R1

For information on how HCM handles hierarchical references to non-HCM
servers, see the HCM Help topic, "How the Release Operation Works."
```

Preparing to Release Your Work Area

```
Before executing the hcm release command, you must do the following:

- If you are going to perform a recursive release, fetch all the
  configurations in the design hierarchy that you are going to release
  into your work area. The hcm release command releases the
  configurations only in your work area.
- If you have locally modified objects and you want those changes to be
  included in the release, check in those objects. If you do not check
  in the objects, the release includes the versions of the objects that
  were previously fetched into the work area. To identify locally
  modified or locked objects, run the DesignSync ls command.
- If you have unmanaged objects in your work area and you want those
  objects to be included in the configuration, check in those objects.
```

SYNOPSIS

```
hcm release  [-description <description>] -name <release_name>
             [-path <path>] [-recursive | -norecursive]
```

OPTIONS

- -description
- -name

- -path
- -[no]recursive

-description

-description <description>	Provides a text description of the release. To specify a description, enclose the text within double quotation marks (""). The default description is "HCM release". Note: You can use ProjectSync to change the description for your release configuration.

-name

-name <release_name>	Specifies the name of the release. Rules for Release Names: - They must be unique within the scope of the module. Names cannot match the name of an existing alias, branch configuration, selector configuration, or release for the module. - They can contain letters, numbers, underscores (_), periods (.), and hyphens (-). All other characters, including whitespace, are prohibited. - They 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 release name and version/branch dot-numeric identifiers. - They 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. - They should not start with 'Sync'(case-insensitive) because Synchronicity may define new keywords in the future using that naming convention.

-path

-path <path>	Indicates the work area directory that contains the configuration for which you want to create a release. If you do not specify a path, HCM releases the module associated with the current

directory. If you specify a work area directory that does not contain a configuration but that is a subdirectory of a configuration's base directory, the hcm release command proceeds up the directory hierarchy until it finds a configuration.

-[no]recursive

-[no]recursive

-recursive performs the release operation on the upper-level module (the module associated with the current directory or the work area specified by the -path option) and each submodule in its hierarchy. This is the default behavior.

Notes:
- All configurations that you want to release must be in your work area.
- When you run hcm release with -recursive, it may contact multiple servers. When it releases submodules, it creates the release on the server on which the submodule resides.
- During a release operation on a module hierarchy, if a submodule needs to be released, then the name of the submodule release is uniquely determined, based on the hierarchy. See the Description section for details on the naming convention used to define the release name for the submodule.
- When you run the hcm release command recursively, the summary identifies any submodules within the hierarchy that were not successfully released.

-norecursive performs the release operation on the configuration associated with the current directory or the work area specified by the -path option.

Note:
- If you perform a nonrecursive release, the hcm release command creates a release that has no hierarchical references to submodules, even if the configuration being released has hierarchical references defined for it.

RETURN VALUE

This command does not return Tcl values.

SEE ALSO

```
hcm addhref, hcm get, hcm mkalias, hcm mkconf, hcm rmmod,
hcm showhrefs, hcm showstatus, ls, tag
,
```

EXAMPLES

- Example of Recursively Releasing a Module
- Understanding the Necessity of Fetching All Configurations

Example of Recursively Releasing a Module

```
This example assumes the following hierarchy in your work area:

Chip               <= uses the default configuration
   Cpu             <= uses the default configuration
   Alu@R1          <= R1 is already released
     Dpath@R2      <= R2 is already released

This example creates an R1 configuration that cannot be modified for
the Chip module and its submodules. It is assumed that the default
configuration of Chip has previously been fetched recursively to the
work area directory /home/username/Designs/Chip using the -recursive
option to the hcm get command.

The "Creating, Fetching, and Releasing Module Hierarchies" example in
the hcm Example topic includes the steps for creating the modules
and configurations that are presented in this example.

dss> hcm release -name R1 -recursive \
                 -description "Initial release of Chip"

This command displays the following output:

---------------------------------------------------------------------
Starting release of module at
        file:///home/username/Designs/Chip

Verifying Release Access for sync:///Projects/Chip

Starting Tagging of files
Skipping /home/username/Designs/Chip/Cpu (excluded since it is the
 base directory of a lower level module)
Skipping /home/username/Designs/Chip/Alu (excluded since it is the
 base directory of a lower level module)
Tagging: sync:///Projects/Chip/Chip1;1.1

Added tag 'R1-1021560986--R' to version '1.1'


Hierarchical reference added on server sync://alu.ABCo.com:2647
sync://chip.ABCo.com:2647/Projects/Chip@R1 ->
```

```
sync://alu.ABCo.com:2647/Projects/Alu@R1 : relative path = "Alu"

------------------------------------------------------------------
Starting release of lower-level module at
        file:///home/username/Designs/Chip/Cpu

Verifying Release Access for sync:///Projects/Cpu

Starting Tagging of files
Tagging: sync:///Projects/Cpu/Cpu1;1.1

Added tag 'Chip.R1-1021560986--R' to version '1.1'


Added Config Chip.R1 -> Chip.R1-1021560986--R.
Hierarchical reference added on server sync://cpu.ABCo.com:2647
sync://chip.ABCo.com:2647/Projects/Chip@R1 ->
sync://cpu.ABCo.com:2647/Projects/Cpu@Chip.R1 : relative path = "Cpu"
Added Config R1 -> R1-1021560986--R.
```

After the command finishes, the work area still contains the default configurations for Chip and Cpu (that is, the releases are created on the servers that contain Chip and Cpu but the work area is not updated). The release hierarchy that now exists on the server(s) is:

```
Chip@R1
  Cpu@Chip.R1
  Alu@R1
    Dpath@R2
```

If you want to get the released hierarchy into your work area, simply run the hcm get command recursively on Chip@R1. The get operation allows for the work area directory containing a module to switch configurations (that is, to switch the default configuration of Chip to be Chip@R1).

```
dss> hcm get -recursive \
       -target sync://chip.ABCo.com:2647/Projects/Chip@R1
```

This get operation should be relatively fast because there are probably no objects to update for the Chip and CPU modules.

Understanding the Necessity of Fetching All Configurations

This example illustrates the importance of fetching all the configurations you want to release into your work area.

The "Creating, Modifying, and Putting Configurations" example in the hcm Example command topic includes the steps for creating the modules, configurations, and hierarchical references presented in this example.

This example assumes the following hierarchy on the server:
Block

```
    BIST@TEST
    Mem2
       DRAM@Silver
       Collar@DEV

It assumes the following hierarchy in your work area:
Block

When you fetched the Block module, it was not fetched recursively.
Therefore, the BIST@TEST and Mem2 submodules do not exist in your
work area.

You want to create an R2 release for the Block module and its
submodules. If you release Block without fetching the submodules,
BIST@TEST and Mem2 will not be released.

dss> hcm release -name R2 -recursive \
                 -path /Designs/Block \
                 -description "7.03 release of Block and submodules"

This command displays the following output:

-------------------------------------------------------------------
Starting release of module at
        file:///Designs/Block

Verifying Release Access for sync:///Projects/Block

Starting Tagging of files
Tagging: sync:///Projects/Block/Block1;1.1
 : Added tag 'R2-1021991824--R' to version '1.1'


Added Config R2 -> R2-1021991824--R.

This example creates only the Block@R2 configuration on the server.
Because the BIST and Mem2 submodules are not in your work area, only
the Block@R2 configuration is created.

By issuing the following hcm get command:

dss> hcm get -target sync://srvr2.ABCo.com:2647/Projects/Block \
             -recursive

The following hierarchy now exists in your work area:
Block
    BIST@TEST
    Mem2
       DRAM@Silver
       Collar@DEV

dss> hcm release -name R3 -recursive -path /Designs/Block \
                 -description "Block and all submodules"

This command displays the following output:

-------------------------------------------------------------------
```

```
Starting release of module at
        file:///home/jsmith/Designs/Block

Verifying Release Access for sync:///Projects/Block

Starting Tagging of files
Skipping /home/jsmith/Designs/Block/BIST
 (excluded since it is the base directory of a lower level module)
Skipping /home/jsmith/Designs/Block/Mem2
 (excluded since it is the base directory of a lower level module)
Tagging: sync:///Projects/Block/Block1;1.3
 : Added tag 'R3-1022015505--R' to version '1.3'


------------------------------------------------------------------
Starting release of lower-level module at
        file:///home/jsmith/Designs/Block/BIST

Verifying Release Access for sync:///Projects/BIST

Starting Tagging of files
Tagging: sync:///Projects/BIST/BIST1;1.1
 : Added tag 'Block.R3-1022015505--R' to version '1.1'


Added Config Block.R3 -> Block.R3-1022015505--R.
Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Block@R3 ->
sync://srvr2.ABCo.com:2647/Projects/BIST@Block.R3 :
 relative path = "BIST"

------------------------------------------------------------------
Starting release of lower-level module at
        file:///home/jsmith/Designs/Block/Mem2

Verifying Release Access for sync:///Projects/Mem2

Starting Tagging of files
Skipping /home/jsmith/Designs/Block/Mem2/DRAM
 (excluded since it is the base directory of a lower level module)
Skipping /home/jsmith/Designs/Block/Mem2/Collar
 (excluded since it is the base directory of a lower level module)
Tagging: sync:///Projects/Mem2/Mem21;1.1
 : Added tag 'Block.R3-1022015505--R' to version '1.1'


Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Mem2@Block.R3 ->
sync://srvr2.ABCo.com:2647/Projects/DRAM@R1 : relative path = "DRAM"

------------------------------------------------------------------
Starting release of lower-level module at
        file:///home/jsmith/Designs/Block/Mem2/Collar

Verifying Release Access for sync:///Projects/Collar

Starting Tagging of files
```

```
Tagging: sync:///Projects/Collar/Collar1;1.1     :
 Added tag 'Block.Mem2.R3-1022015505--R' to version '1.1'


Added Config Block.Mem2.R3 -> Block.Mem2.R3-1022015505--R.
Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Mem2@Block.R3 ->
sync://srvr2.ABCo.com:2647/Projects/Collar@Block.Mem2.R3 :
 relative path = "Collar"
Added Config Block.R3 -> Block.R3-1022015505--R.
Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Block@R3 ->
sync://srvr2.ABCo.com:2647/Projects/Mem2@Block.R3 :
 relative path = "Mem2"

Added Config R3 -> R3-1022015505--R.

This release command creates the Block@R3 configuration, the
BIST@Block.R3 configuration, the Mem2@Block.R3 configuration, the
Collar@Block.Mem2.R3 configuration, and the DRAM@Block.Mem2.R3 on
the DesignSync server sync://srvr2.ABCo.com.
```

hcm release

hcm release Command

NAME

```
hcm release         - Creates a configuration that cannot be modified
                      (legacy)
```

DESCRIPTION

- Controlling the Ability to Release Configurations
- Understanding the State of Your Work Area State After a Release
- Releasing Submodules
- Preparing to Release Your Work Area

```
This command creates a configuration that cannot be modified (a
release) for one or more modules. This command performs recursively;
that is, it creates configurations for the upper-level module (module
associated with the current directory or the work area specified by the
-path option) and its submodules. If you want to release just the
upper-level module, run the hcm release command with the -norecursive
option. If you perform a nonrecursive release, the hcm release command
creates a release that has no hierarchical references.

For non-legacy modules, the functionality of configurations is now
available through tags on module versions and branches. For more
information, see tags.
```

Notes:
- Only configurations that exist in your work area can be released.
- Release names must be unique within the scope of the module. If the module has an alias, branch configuration, selector configuration, or release with the same name as specified by -name, the release operation will fail.
- The hcm release command fails if it is operating recursively and a lower-level module does not exist in the work area.
- Released configurations cannot be changed. For example, you cannot run hcm get with the -edit option or run hcm addhref on releases. (Note: Although the contents of a release cannot be changed, the release can be removed from the server with the hcm rmmod operation. This operation removes a module's configuration definitions as well as the module itself. See the hcm rmmod command topic.)
- The execution of the hcm release command could fail to complete; for example, if a referenced server is unavailable or the release of a submodule failed. HCM recognizes that the release was not completed, notifies you, and permits you to rerun the command.

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

Controlling the Ability to Release Configurations

The ability to create a release can be access-controlled. By default, any user can create a release. If this policy is not acceptable for your project, you (as the administrator or project leader) can restrict or deny this capability for your site or individual SyncServers. For more information, see the Synchronicity Access Control Guide. See also information on the access allow and access deny commands.

Understanding the State of Your Work Area State After a Release

The hcm release command does not alter the state of your work area in any way. If you fetched a branch configuration of a module into your work area and then release it, your work area still contains the branch configuration of the module. This behavior is desirable in that you probably want to continue making changes to the configuration and then to make subsequent releases of the changes.

If you want to switch your work area to reflect the release instead of the branch configuration, run the hcm get command to fetch a release and replace the branch configuration. The hcm get command allows you to switch the configuration of a module that already has been fetched into your work area. "Example 1: Creating, Fetching, and Releasing Module Hierarchies" in the hcm Example topic shows how you can switch your work area.

Releasing Submodules

By default, the hcm release command performs a recursive release. When a recursive release operation encounters a submodule, it does the following:

- For submodules that have been released, it creates a new reference. The new reference is from the new release of the upper-level module to the referenced release configuration of the submodule.

- If the submodule is referenced from the upper-level module using an alias (for example, Mem2 contains a hierarchical reference to DRAM@Silver and Silver is an alias to the DRAM@R1 configuration), a new release of DRAM is not created. However, the hierarchical reference created in the release of Mem2 will resolve the alias. Thus, the release of Mem2 will contain a reference to DRAM@R1 and not DRAM@Silver. ("Example 2: Understanding the Necessity of Fetching All Configurations" in this topic illustrates the release of an aliased configuration.)

- For branch or selector configurations, the release operation descends into the submodule and releases it. It creates a new hierarchical reference from the released upper-level module to the released submodule.

- During a recursive release operation, if a hierarchical reference is encountered that points to a vault path that has not been defined as an HCM module (by using the hcm mkmod command), this situation is treated as though the reference is to a branch configuration. Thus, the hcm release command descends into the vault path, implicitly defines the path as an HCM module, and creates a release for the module.

- The submodule that is released is not required to exist on the same server that contains the upper-level module that references the submodule. The hcm release command handles the communication and coordination of releasing submodules on any number of servers. The release of the submodule is created on the same server that contains the branch configuration of the submodule.

- The release of the submodule must have a name (as does any configuration). To name the submodule's release uniquely, a unique configuration name is generated. This unique name is formed by concatenating the names of all the upper-level modules and prefixing the name produced to the name specified by the -name option. For example, consider the following hierarchy and the effect produced by the invocation of hcm release:

```
sync://chip.ABCo.com:2647/Projects/Chip@DEV
  sync://cpu.ABCo.com:2647/Projects/CPU@DEV
    sync://alu.ABCo.com:2647/Projects/ALU@DEV

dss> hcm release -name R1
```

The following releases are created:

```
     sync://chip.ABCo.com:2647/Projects/Chip@R1
       sync://cpu.ABCo.com:2647/Projects/CPU@Chip.R1
         sync://alu.ABCo.com:2647/Projects/ALU@Chip.CPU.R1

For information on how HCM handles hierarchical references to non-HCM
servers, see the HCM Help topic, "How the Release Operation Works."
```

Preparing to Release Your Work Area

```
Before executing the hcm release command, you must do the following:

- If you are going to perform a recursive release, fetch all the
  configurations in the design hierarchy that you are going to release
  into your work area. The hcm release command releases the
  configurations only in your work area.
- If you have locally modified objects and you want those changes to be
  included in the release, check in those objects. If you do not check
  in the objects, the release includes the versions of the objects that
  were previously fetched into the work area. To identify locally
  modified or locked objects, run the DesignSync ls command.
- If you have unmanaged objects in your work area and you want those
  objects to be included in the configuration, check in those objects.
```

SYNOPSIS

```
hcm release  [-description <description>] -name <release_name>
             [-path <path>] [-recursive | -norecursive]
```

OPTIONS

- -description
- -name
- -path
- -[no]recursive

-description

```
 -description          Provides a text description of the
  <description>        release. To specify a description, enclose
                       the text within double quotation marks
                       (""). The default description is "HCM
                       release".

                       Note: You can use ProjectSync to change the
                       description for your release configuration.
```

-name

-name <release_name>	Specifies the name of the release. Rules for Release Names: - They must be unique within the scope of the module. Names cannot match the name of an existing alias, branch configuration, selector configuration, or release for the module. - They can contain letters, numbers, underscores (_), periods (.), and hyphens (-). All other characters, including whitespace, are prohibited. - They 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 release name and version/branch dot-numeric identifiers. - They 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. - They should not start with 'Sync'(case-insensitive) because Synchronicity may define new keywords in the future using that naming convention.

-path

-path <path>	Indicates the work area directory that contains the configuration for which you want to create a release. If you do not specify a path, HCM releases the module associated with the current directory. If you specify a work area directory that does not contain a configuration but that is a subdirectory of a configuration's base directory, the hcm release command proceeds up the directory hierarchy until it finds a configuration.

-[no]recursive

-[no]recursive	-recursive performs the release operation on the upper-level module (the module associated with the current directory or the work area specified by the -path option) and each

```
submodule in its hierarchy. This is the default
behavior.

Notes:
- All configurations that you want to release
  must be in your work area.
- When you run hcm release with -recursive, it
  may contact multiple servers. When it
  releases submodules, it creates the release
  on the server on which the submodule
  resides.
- During a release operation on a module
  hierarchy, if a submodule needs to be
  released, then the name of the submodule
  release is uniquely determined, based on the
  hierarchy. See the Description section for
  details on the naming convention used to
  define the release name for the submodule.
- When you run the hcm release command
  recursively, the summary identifies any
  submodules within the hierarchy that were
  not successfully released.

-norecursive performs the release operation on
the configuration associated with the current
directory or the work area specified by the
-path option.

Note:
- If you perform a nonrecursive release, the
  hcm release command creates a release that
  has no hierarchical references to submodules,
  even if the configuration being released has
  hierarchical references defined for it.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
hcm addhref, hcm get, hcm mkalias, hcm mkconf, hcm rmmod,
hcm showhrefs, hcm showstatus, ls, tag
'
```

EXAMPLES

- Example of Recursively Releasing a Module
- Understanding the Necessity of Fetching All Configurations

Example of Recursively Releasing a Module

```
This example assumes the following hierarchy in your work area:

Chip               <= uses the default configuration
   Cpu             <= uses the default configuration
   Alu@R1          <= R1 is already released
     Dpath@R2      <= R2 is already released

This example creates an R1 configuration that cannot be modified for
the Chip module and its submodules. It is assumed that the default
configuration of Chip has previously been fetched recursively to the
work area directory /home/username/Designs/Chip using the -recursive
option to the hcm get command.

The "Creating, Fetching, and Releasing Module Hierarchies" example in
the hcm Example topic includes the steps for creating the modules
and configurations that are presented in this example.

dss> hcm release -name R1 -recursive \
                 -description "Initial release of Chip"

This command displays the following output:

---------------------------------------------------------------
Starting release of module at
        file:///home/username/Designs/Chip

Verifying Release Access for sync:///Projects/Chip

Starting Tagging of files
Skipping /home/username/Designs/Chip/Cpu (excluded since it is the
 base directory of a lower level module)
Skipping /home/username/Designs/Chip/Alu (excluded since it is the
 base directory of a lower level module)
Tagging: sync:///Projects/Chip/Chip1;1.1

Added tag 'R1-1021560986--R' to version '1.1'


Hierarchical reference added on server sync://alu.ABCo.com:2647
sync://chip.ABCo.com:2647/Projects/Chip@R1 ->
sync://alu.ABCo.com:2647/Projects/Alu@R1 : relative path = "Alu"

---------------------------------------------------------------
Starting release of lower-level module at
        file:///home/username/Designs/Chip/Cpu

Verifying Release Access for sync:///Projects/Cpu

Starting Tagging of files
Tagging: sync:///Projects/Cpu/Cpu1;1.1

Added tag 'Chip.R1-1021560986--R' to version '1.1'
```

```
Added Config Chip.R1 -> Chip.R1-1021560986--R.
Hierarchical reference added on server sync://cpu.ABCo.com:2647
sync://chip.ABCo.com:2647/Projects/Chip@R1 ->
sync://cpu.ABCo.com:2647/Projects/Cpu@Chip.R1 : relative path = "Cpu"
Added Config R1 -> R1-1021560986--R.
```

After the command finishes, the work area still contains the default configurations for Chip and Cpu (that is, the releases are created on the servers that contain Chip and Cpu but the work area is not updated). The release hierarchy that now exists on the server(s) is:

```
Chip@R1
  Cpu@Chip.R1
  Alu@R1
    Dpath@R2
```

If you want to get the released hierarchy into your work area, simply run the hcm get command recursively on Chip@R1. The get operation allows for the work area directory containing a module to switch configurations (that is, to switch the default configuration of Chip to be Chip@R1).

```
dss> hcm get -recursive \
       -target sync://chip.ABCo.com:2647/Projects/Chip@R1
```

This get operation should be relatively fast because there are probably no objects to update for the Chip and CPU modules.

Understanding the Necessity of Fetching All Configurations

This example illustrates the importance of fetching all the configurations you want to release into your work area.

The "Creating, Modifying, and Putting Configurations" example in the hcm Example command topic includes the steps for creating the modules, configurations, and hierarchical references presented in this example.

This example assumes the following hierarchy on the server:

```
Block
    BIST@TEST
    Mem2
       DRAM@Silver
       Collar@DEV
```

It assumes the following hierarchy in your work area:

```
Block
```

When you fetched the Block module, it was not fetched recursively. Therefore, the BIST@TEST and Mem2 submodules do not exist in your work area.

You want to create an R2 release for the Block module and its

```
submodules. If you release Block without fetching the submodules,
BIST@TEST and Mem2 will not be released.

dss> hcm release -name R2 -recursive \
                 -path /Designs/Block \
                 -description "7.03 release of Block and submodules"

This command displays the following output:

-------------------------------------------------------------------
Starting release of module at
        file:///Designs/Block

Verifying Release Access for sync:///Projects/Block

Starting Tagging of files
Tagging: sync:///Projects/Block/Block1;1.1
 : Added tag 'R2-1021991824--R' to version '1.1'


Added Config R2 -> R2-1021991824--R.

This example creates only the Block@R2 configuration on the server.
Because the BIST and Mem2 submodules are not in your work area, only
the Block@R2 configuration is created.

By issuing the following hcm get command:

dss> hcm get -target sync://srvr2.ABCo.com:2647/Projects/Block \
             -recursive

The following hierarchy now exists in your work area:
Block
    BIST@TEST
    Mem2
       DRAM@Silver
       Collar@DEV

dss> hcm release -name R3 -recursive -path /Designs/Block \
                 -description "Block and all submodules"

This command displays the following output:

-------------------------------------------------------------------
Starting release of module at
        file:///home/jsmith/Designs/Block

Verifying Release Access for sync:///Projects/Block

Starting Tagging of files
Skipping /home/jsmith/Designs/Block/BIST
 (excluded since it is the base directory of a lower level module)
Skipping /home/jsmith/Designs/Block/Mem2
 (excluded since it is the base directory of a lower level module)
Tagging: sync:///Projects/Block/Block1;1.3
 : Added tag 'R3-1022015505--R' to version '1.3'
```

```
------------------------------------------------------------------
Starting release of lower-level module at
        file:///home/jsmith/Designs/Block/BIST

Verifying Release Access for sync:///Projects/BIST

Starting Tagging of files
Tagging: sync:///Projects/BIST/BIST1;1.1
 : Added tag 'Block.R3-1022015505--R' to version '1.1'


Added Config Block.R3 -> Block.R3-1022015505--R.
Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Block@R3 ->
sync://srvr2.ABCo.com:2647/Projects/BIST@Block.R3 :
 relative path = "BIST"

------------------------------------------------------------------
Starting release of lower-level module at
        file:///home/jsmith/Designs/Block/Mem2

Verifying Release Access for sync:///Projects/Mem2

Starting Tagging of files
Skipping /home/jsmith/Designs/Block/Mem2/DRAM
 (excluded since it is the base directory of a lower level module)
Skipping /home/jsmith/Designs/Block/Mem2/Collar
 (excluded since it is the base directory of a lower level module)
Tagging: sync:///Projects/Mem2/Mem21;1.1
 : Added tag 'Block.R3-1022015505--R' to version '1.1'


Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Mem2@Block.R3 ->
sync://srvr2.ABCo.com:2647/Projects/DRAM@R1 : relative path = "DRAM"

------------------------------------------------------------------
Starting release of lower-level module at
        file:///home/jsmith/Designs/Block/Mem2/Collar

Verifying Release Access for sync:///Projects/Collar

Starting Tagging of files
Tagging: sync:///Projects/Collar/Collar1;1.1     :
 Added tag 'Block.Mem2.R3-1022015505--R' to version '1.1'


Added Config Block.Mem2.R3 -> Block.Mem2.R3-1022015505--R.
Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Mem2@Block.R3 ->
sync://srvr2.ABCo.com:2647/Projects/Collar@Block.Mem2.R3 :
 relative path = "Collar"
Added Config Block.R3 -> Block.R3-1022015505--R.
Hierarchical reference added on server sync://srvr2.ABCo.com:2647
sync://srvr2.ABCo.com:2647/Projects/Block@R3 ->
sync://srvr2.ABCo.com:2647/Projects/Mem2@Block.R3 :
```

```
 relative path = "Mem2"

Added Config R3 -> R3-1022015505--R.

This release command creates the Block@R3 configuration, the
BIST@Block.R3 configuration, the Mem2@Block.R3 configuration, the
Collar@Block.Mem2.R3 configuration, and the DRAM@Block.Mem2.R3 on
the DesignSync server sync://srvr2.ABCo.com.
```

hcm rmalias

hcm rmalias Command

NAME

```
hcm rmalias         - Removes an alias for a release (legacy)
```

DESCRIPTION

- Controlling the Ability to Remove Aliases

```
This command removes an alias (symbolic name) for a release and
detaches any ProjectSync notes that may have been attached to the
alias. The rmalias operation also generates a RevisionControl note and
attaches it to both the owning module and the release to which the
alias points. Once an alias has been removed, you can reuse its name.

For non-legacy modules, the functionality of aliases is available
through tags.  For more information, see tag help.

Notes:
- The hcm rmalias command affects only the server; it does not affect
  your work area. If you fetched the release referenced by the alias
  into your work area, that referenced release still exists in
  your work area.
- The hcm rmalias command removes the specified alias; it does not
  remove the release to which the alias refers or hierarchical
  references to the alias.

For this command to be successful, the following must be true:
- The Synchronicity URL specified for -target must exist and must
  correspond to an alias.
- The server on which the alias resides must be available and must
  have HCM 1.1 or higher installed.

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

Controlling the Ability to Remove Aliases

```
The ability to remove aliases can be access-controlled. By default,
no users can remove aliases. If this policy is not acceptable for
your project, you (as the administrator or project leader) can
control this capability for your site or individual
SyncServers. For more information, see the Synchronicity Access
Control Guide. See also information on the access allow and access
deny commands.

Note:
- This command obeys the HCM 'Rmalias' access control. If you are
  trying to remove an alias and you do not have permission to do so,
  this command displays an error.
```

SYNOPSIS

```
hcm rmalias [-notes] -target <alias_url>
```

OPTIONS

- -notes
- -target

-notes

```
-notes                  Deletes those notes that were detached from the
                        alias and not attached to other objects.

                        Note:
                        - Notes attached to the alias you are removing
                          may not be deleted, even though you specify
                          the -notes option. When you add a note to an
                          alias, the installed hcmNoteAttach trigger
                          automatically attaches the note to the release
                          currently pointed to by the alias. Because the
                          note is attached to both the alias and the
                          release to which the alias points, the rmalias
                          operation cannot delete the note. For more
                          information about the hcmNoteAttach trigger,
                          see the DesignSync Data Manager HCM User's
                          Guide topic, "Adding the hcmNoteAttach
                          Trigger."
```

-target

```
-target <alias_url>     Identifies the alias you want to remove.

                        Specify the Synchronicity URL as follows:
```

```
                    sync[s]://<host>[:<port>]/<vaultPath>@<AliasName>
                      where <host> is the SyncServer on which the
                      module resides, <port> is the SyncServer port
                      number, and <vaultPath>@<AliasName> identifies
                      the alias to remove.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
access allow, access deny, hcm mkalias, hcm rmmod, hcm rmconf, tag
,
```

EXAMPLES

```
The following example removes the alias 'Golden' for the released
configuration of Chip@R1.

The "Creating, Fetching, and Releasing Module Hierarchies" example in
the hcm Example topic includes the steps for creating the Chip
module, R1 release, and Golden alias.

dss> hcm rmalias -target sync://chip.ABCo.com:2647/Projects/Chip@Golden
```

hcm rmconf

hcm rmconf Command

NAME

```
hcm rmconf          - Removes a legacy configuration from a server
                      (legacy)
```

DESCRIPTION

- Controlling the Ability to Remove Configurations

```
This command removes an HCM configuration from a server. It removes all
hierarchical references from the deleted configuration and detaches any
ProjectSync notes that may have been attached to the configuration.
This command also generates a RevisionControl note for the
```

configuration and attaches the note to the owning module.

For non-legacy modules, the functionality of configurations has been replaced by tags on module versions and branches.

Using this command, you can remove either selector configurations or branch configurations that are not default configurations. Default configurations are only removed when the module is removed, see the hcm rmmod command. You can remove an alias with the hcm rmalias command. You cannot remove a release (unless you remove the entire module with the hcm rmmod command).

Notes:
- The hcm rmconf command affects only the server; it does not affect your work area. If you fetched the configuration into your local work area, that configuration still exists in your local work area.
- When you remove a configuration, the rmconf operation does not remove the selector(s) that identify objects in a vault as members of that configuration. If you create a new configuration (using the hcm mkconf command) with the same name as a deleted configuration, the new configuration contains the contents of the old configuration.
- This command is not recursive, it does not follow the hierarchical references of the configuration and delete the referenced modules. To remove an entire module hierarchy, you must remove each configuration separately in a bottom up fashion.
- The hcm rmconf command removes the specified configuration; it does not remove hierarchical references to the configuration.

For this command to be successful, the following must be true:
- The Synchronicity URL specified for -target must exist and must correspond to a selector configuration or a branch configuration that is not a default configuration.
- The server on which the configuration resides must be available and must have HCM 1.1 or higher installed.

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

Controlling the Ability to Remove Configurations

The ability to remove configurations can be access-controlled. By default, no users can remove configurations. If this policy is not acceptable for your project, you (as the administrator or project leader) can control this capability for your site or individual SyncServers. For more information, see the Synchronicity Access Control Guide. See also information on the access allow and access deny commands.

Note:
- This command obeys the HCM 'Rmconf' access control. If you are trying to remove a configuration and you do not have permission to do so, this command displays an error.

SYNOPSIS

```
hcm rmconf [-notes] -target <configuration_url>
```

OPTIONS

- -notes
- -target

-notes

```
-notes                  Deletes those notes that were detached from the
                        configuration and not attached to other objects.

                        Note:
                        - Notes attached to the configuration you are
                          removing may not be deleted, even though
                          you specify the -notes option. This
                          behavior occurs when you use the ProjectSync
                          GUI, rather than some other means, such as a
                          server-side script. When you use the
                          DesignSync WebUI to attach a note to a
                          configuration, ProjectSync also attaches a
                          note to the project (module). The rmconf
                          operation does not delete the notes because
                          they are attached to the project as well as
                          the configuration.
```

-target

```
-target                 Identifies the configuration you want to remove.
 <configuration_url>
                        Specify the Synchronicity URL as follows:
                      sync[s]://<host>[:<port>]/<vaultPath>@<ConfigName>
                        where <host> is the SyncServer on which the
                        module resides, <port> is the SyncServer port
                        number, and <vaultPath>@<ConfigName> identifies
                        the configuration to be removed.

                        Note:
                        - -target must be either a branch configuration,
                          which is not a default configuration, or a
                          selector configuration. If you want to remove
                          an alias, use hcm rmalias. You cannot
                          remove a release.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
access allow, access deny, hcm mkconf, hcm rmmod, hcm rmalias, tag
,
```

EXAMPLES

```
The following example removes the TEST selector configuration from the
IO module. All of the ProjectSync notes attached to the configuration
are detached but not deleted.

Note:
  The "Creating and Removing Module Configurations" example in the
  hcm Example topic includes the steps for creating the module and
  configuration presented in this example.

dss> hcm rmconf -target sync://srvr1.ABCo.com:2647/Projects/IO@TEST
```

hcm rmhref

hcm rmhref Command

NAME

```
hcm rmhref          - Removes a hierarchical reference between
                      modules (legacy)
```

DESCRIPTION

```
This command removes a hierarchical reference (connection) or
references from the configuration of an upper-level module to any of
the following: a submodule configuration, a DesignSync vault, or an
IP Gear deliverable.

This command is for removing hierarchical references from legacy
modules only.  To remove hrefs from non-legacy modules, use the
rmhref command with no "hcm" prefix.

The -fromtarget, -totarget, and -relpath options are not all required by
the rmhref operation. Depending on whether you specify -fromtarget,
the -totarget or -relpath option may be required.
- If you do not specify -fromtarget, the hcm rmhref command uses the
  current work area to determine the upper-level module. To determine
```

```
  which hierarchical reference to remove, either -totarget or -relpath
  must be specified.
  o If -totarget is not specified, -relpath is required. The hcm rmhref
    command uses -relpath and the base directory of -fromtarget to
    determine which hierarchical reference to remove.
  o If -totarget is specified, -relpath is optional.
- If you specify -fromtarget, -totarget is required and -relpath is
  optional.

The ability to remove hierarchical references can be
access-controlled.  By default, any user can delete new
hierarchical references. If this policy is not acceptable for your
project, you (as the administrator or project leader) can restrict
or deny this capability for your site or individual SyncServers.
For more information, see the Synchronicity Access Control Guide.
See also information on the access allow and access deny commands.

Notes:
- The hcm rmhref command affects only the server definition of the
  hierarchical references. For example, if you fetched a configuration
  with hierarchical references into your work area and later a
  hierarchical reference is removed from the configuration on the
  server, the "deleted" hierarchical reference still exists in the
  local definition. To identify hierarchical references removed on the
  server, run the hcm showstatus command. To synchronize your
  work area with the server, run the hcm get command.
- If you run the hcm rmhref command using the default behavior for the
  -fromtarget, -totarget, or -relpath options, verify that your
  work area is current by running the hcm showstatus command.
- If you created a hierarchical reference (using hcm addhref) without
  specifying a port for the item to which you are connecting
  (identified by the -totarget option), the hcm rmhref command does
  not recognize the reference unless you specify the default port (2647
  for sync: addresses and 2679 for syncs: addresses). For example, if
  you created a hierarchical reference between the Chip module and the
  DEV configuration of the ALU module as follows:

   dss> hcm addhref -fromtarget sync://chip.ABCo.com:2647/Projects/Chip \
                    -totarget sync://alu.ABCo.com/Projects/ALU@DEV \
                    -relpath ALU

  Hierarchical reference added on server sync://chip.ABCo.com:2647
  sync://chip.ABCo.com:2647/Projects/Chip ->
  sync://alu.ABCo.com:2647/Projects/ALU@DEV: relative path = "ALU"

  If you try to remove the hierarchical reference without specifying
  the default port for the ALU@DEV module, HCM will not find it.

   dss> hcm rmhref -fromtarget sync://chip.ABCo.com:2647/Projects/Chip \
                   -totarget sync://alu.ABCo.com/Projects/ALU@DEV

   No matching hrefs found

  To successfully remove the hierarchical reference, run the following:

   dss> hcm rmhref -fromtarget sync://chip.ABCo.com:2647/Projects/Chip \
                   -totarget sync://alu.ABCo.com:2647/Projects/ALU@DEV
```

```
This command is subject to access controls on the server.
```

SYNOPSIS

```
hcm rmhref [-fromtarget <uppermod_url>] [-relpath <relative_path>]
           [-totarget <submod_url>]
```

OPTIONS

- -fromtarget
- -relpath
- -totarget

-fromtarget

```
-fromtarget              Specifies the URL of a configuration
 <uppermod_url>          corresponding to an upper-level module from
                         which you want to remove a connection.

                         o To specify the default configuration of the
                           upper-level module, use the following syntax:
                            sync[s]://<host>[:<port>]/<vaultPath>
                           where <vaultPath> identifies the module.

                         o To specify a configuration other than the
                           default configuration, use the following
                           syntax:
                           sync[s]://<host>[:<port>]/<vaultPath>@<config>
                           where <vaultPath> identifies the module and
                           <config> identifies the specific
                           configuration.

                         Note:
                         - If you do not specify -fromtarget, the
                           hcm rmhref command determines the upper-level
                           module from the current directory. If the
                           current directory is not a base directory of
                           a target, the hcm rmhref command fails.
```

-relpath

```
-relpath                 Indicates the path between the modules relative
 <relative_path>         to the top of the upper-level module. When
                         you specify -relpath, HCM only deletes the
                         connection between the upper-level module and
                         submodule along the specified path.
```

```
                        Notes:
                        - If you do not specify -relpath, all
                          hierarchical references between the
                          upper-level module, specified by -fromtarget,
                          and the submodule, specified by -totarget, are
                          deleted.
                        - If you do not specify -totarget, you must
                          specify -relpath.
```

-totarget

```
 -totarget              Specifies the URL of an IP Gear deliverable,
  <submod_url>          DesignSync vault, or a submodule configuration.

                        o To specify a default configuration of a
                          submodule, use the following syntax:
                           sync[s]://<host>[:<port>]/<vaultPath>
                          where <vaultPath> identifies the module.

                        o To specify a configuration other than the
                          default configuration, use the following
                          syntax:
                          sync[s]://<host>[:<port>]/<vaultPath>@<config>
                          where <vaultPath> identifies the module and
                          <config> identifies the specific
                          configuration.

                        o To specify an IP Gear deliverable, use the
                          following syntax:
                           sync[s]://<host>[:<port>]/Deliverable/<ID>
                          where <host> is the SyncServer on which the
                          module resides, <port> is the SyncServer port
                          number, and <ID> is the deliverable ID number;
                          for example,
                      sync://publisher.ipgsrvr1.com:2647/Deliverable/142.

                        Notes:
                        - If you do not specify -totarget, the
                          hcm rmhref command determines the object or
                          objects to which the upper-level module
                          connects from -fromtarget (or the current
                          directory if -fromtarget is not supplied) and
                          -relpath.
                        - If you specify -fromtarget, you must specify
                          -totarget. By specifying -totarget, you
                          ensure that you are removing the correct
                          hierarchical reference.

                        Using Wildcards to Remove References
                        ====================================
                        You can use pattern matching to remove
                        hierarchical references. Glob style pattern
                        matching is used.
```

```
- To delete all hierarchical references between
  the upper-level module and submodules, specify
  only '*'. "Example 2: Removing all
  Hierarchical References from a Configuration"
  illustrates this usage.

- To delete specific hierarchical
  references between the upper-level module and
  submodules specify a pattern and either '*' or
  '?'.
  o '*' indicates that hcm rmhref should match
    any number of characters.
  o '?' indicates that hcm rmhref should match
    one instance of any character.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
access allow, access define, hcm addhref, hcm get, hcm showhrefs,
hcm showstatus
,
```

EXAMPLES

- Removing a Hierarchical Reference
- Removing all Hierarchical References from a Configuration

Removing a Hierarchical Reference

```
The following example assumes this hierarchy:
Top                <= uses the default configuration
   Mem@DEV         <= DEV is a branch configuration
   IO@TEST         <= TEST is a selector configuration

The "Creating and Removing Module Configurations" example in the hcm
Example topic includes the steps for creating the modules,
configurations, and hierarchical references presented in this
example.

This example removes the connection from the Top module to the
TEST configuration of the IO module.

dss> hcm rmhref \
          -fromtarget sync://srvr1.ABCo.com:2647/Projects/Top \
          -totarget sync://srvr1.ABCo.com:2647/Projects/IO@TEST
```

Removing all Hierarchical References from a Configuration

```
The following example assumes this hierarchy:
Block               <= uses the default configuration
    IO2             <= uses the default configuration
    Mem2            <= uses the default configuration
       DRAM@Silver  <= Silver is an alias
       Collar@DEV   <= DEV is a branch configuration

The "Creating, Modifying, and Putting Configurations" example in the
hcm Example topic includes the steps for creating the modules,
configurations, and hierarchical references presented in this
example.

This example removes all the connections kept in the default
configuration of Mem2, including the connections to DRAM@Silver and
Collar@DEV.

dss> hcm rmhref \
          -fromtarget sync://srvr2.ABCo.com:2647/Projects/Mem2 \
          -totarget *
```

hcm rmmod

hcm rmmod Command

NAME

```
hcm rmmod           - Removes a legacy module and its configurations
                      from a server (legacy)
```

DESCRIPTION

- Controlling the Ability to Remove Modules

```
This command removes a legacy module (data that represents a level of
the design hierarchy) and its configurations from a server. Optionally,
this command, when run with the -vaultdata option, removes the vault
folder in which the module contents reside and deletes all objects in
in that vault. For more information, see the -vaultdata option
description.

This command is used to remove legacy modules in legacy module
mode, or both legacy and non-legacy modules when legacy mode is
not enabled.  For information on removing non-legacy modules, or
removing modules when not in legacy mode, see the rmmod help.

This command also detaches any ProjectSync notes that may have been
attached to the module and its configurations. Optionally, this
```

```
command, when run with the -notes option, deletes the detached
notes.

The rmmod operation generates a RevisionControl note for the
deleted module. The generated RevisionControl note is attached to
the module and includes the URLs (shown as paths relative to the
server vault) of the module and all of its configurations in the
Objects field. This note, unlike other notes, is not detached from
the vault folder associated with the module. Not detaching the
generated RevisionControl note ensures that when a ProjectSync note
query is run on that folder the note is retrieved, showing that the
folder once contained the data of a deleted module. The -notes and
-vaultdata options control when this note will be deleted. For more
information, refer to the -notes option description.

WARNING: With this command, you can delete releases. By default, the
         use of this command is restricted. It is recommended that
         administrators remove this restriction temporarily, if
         needed, and with extreme care.

Notes:
- The hcm rmmod command affects only the server; it does not affect
  your work area. If you fetched a configuration of the module into
  local work area, that configuration still exists in your local work
  area.
- This command is not recursive, it does not follow the hierarchical
  references of the module configurations and delete the referenced
  modules.  To remove an entire module hierarchy, you must remove
  each module separately.
- The hcm rmmod command removes the specified module; it does not
  remove hierarchical references to configurations of the deleted
  module (that is, configurations specified for -totarget option in
  an addhref operation).
- If you chose to remove a module without removing the vault folder
  (running the hcm rmmod command without the -vaultdata option),
  the hcm showconfs command continues to display the configurations for
  the deleted module. Additionally, if the module resides in the
  /Projects folder, the module and its configurations appear in
  ProjectSync as a ProjectSync project.

For this command to be successful, the following must be true:
- The Synchronicity URL specified for -target must exist.
- The server on which the module resides must be available and must
  have HCM 1.1 or higher installed.
```

Controlling the Ability to Remove Modules

```
The ability to remove modules can be access-controlled. By default,
no users can remove modules. If this policy is not acceptable for
your project, you (as the administrator or project leader) can
control this capability for your site or individual SyncServers.
For more information, see the Synchronicity Access Control
Guide. See also information on the access allow and access deny
```

```
commands.

Notes:
- This command obeys the HCM 'Rmmod' access control. If you are
  trying to remove a module and you do not have permission to do so,
  this command displays an error.
- If you run this command with the -vaultdata option, the command checks
  the 'Delete' access control before it removes the vault.
- This command does not check the 'Rmconf' or 'Rmalias' access controls
  when removing the module's configurations and aliases.
```

SYNOPSIS

```
hcm rmmod [-notes] -target <module_url> [-vaultdata]
```

OPTIONS

- -notes
- -target
- -vaultdata

-notes

```
-notes                Deletes those notes that were detached from the
                      module or its configurations and not attached
                      to other objects.

                      Note:
                      - When you run the hcm rmmod command, it
                        generates a RevisionControl note for the
                        deleted module. This note will be deleted if
                        both of the following conditions are met:
                        o You run the hcm rmmod command with both
                          the -notes and -vaultdata options.
                        o The default RmVaultKeepVid behavior has
                          been overridden and version information
                          for the removed objects is not retained.
```

-target

```
-target <module_url>  Identifies the module you want to remove.

                      Specify the Synchronicity URL as follows:
                       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 remove.
```

This command derives the name of the module from the leaf of the vault path. For example, if you specify sync://alu.ABCo.com:2647/Projects/ALU for -target, with Projects/ALU being the vault path, the command removes a module named ALU.

-vaultdata

-vaultdata

Removes the vault folder in which the module contents reside. This option deletes all objects, including directories and locked files, within the vault.

Notes:
- If -vaultdata is specified, the rmmod operation calls the rmfolder operation to delete the vault data. The rmfolder command obeys the RmVaultKeepVid client-side registry key to determine if version information for the deleted files is retained. For more information, see the "Advanced Registry Settings" topic in the DesignSync Help.
- If you chose not to remove the vault folder, the following occurs:
  - o hcm showconfs continues to display the configurations for the deleted module.
  - o if the module resides in the /Projects folder, the module and its configurations appear in ProjectSync as a ProjectSync project.

RETURN VALUE

This command does not return Tcl values.

SEE ALSO

access allow, access deny, hcm mkmod, hcm rmconf, hcm rmalias, rmfolder, rmvault
,

EXAMPLES

The following example removes the BIST module from the DesignSync server sync://srvr2.ABCo.com.

```
The "Creating, Modifying, and Putting Configurations" example in the
hcm Example topic includes the steps for creating the modules and
hierarchy presented in this example.

dss> hcm rmmod -target sync://srvr2.ABCo.com:2647/Projects/BIST \
               -vaultdata

In this example, the hcm rmmod command:
o Deletes the default and TEST configurations of the BIST module.
o Deletes vault data associated with module.
o Detaches all of the ProjectSync notes attached to the module and
  any of its configurations.
o Retains the latest version ID of all removed vaults (depending
  on the value of the DesignSync RmVaultKeepVid client-side
  registry setting).
```

hcm showconfs

hcm showconfs Command

NAME

```
hcm showconfs       - Displays module configurations (legacy)
```

DESCRIPTION

- Display of Configurations on the Server (Specified with -target)
- Understanding the Output of hcm showconfs -target
- Display of Configurations in a Work Area (Specified with -path)
- Understanding the Output of hcm showconfs -path

```
This command displays legacy module configurations on the server or
the legacy configurations in your work area, depending on whether you
specify the -target or the -path option:

o If you specify the -target option, the command displays the
  configurations of a legacy module on the server.

o If you specify the -path option, the command displays legacy
  configurations in a specified directory in your work area.

Note: The functionality of configurations is available through
tags on non-legacy module versions and branches. Module information
for non-legacy modules is available through the vhistory, tags, and
the hcm showmods commands.

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

Display of Configurations on the Server (Specified with -target)

```
When used with the -target option, this command displays the
configurations and aliases for a specified legacy module on the server.
You can run this command to determine the configurations available
for hcm addhref -totarget values.
```

Understanding the Output of hcm showconfs -target

```
By default, or if you run the hcm showconfs -target command with
the '-report normal' option, the command displays a list of
configurations associated with the legacy module in a user-friendly
format. The '-report normal' option also displays a row for the
default configuration at the top of the output.

If you run the hcm showconfs -target command with '-report script',
it returns a Tcl list in the following form:

    <config_name1> {<config_data_list1>}
     <config_name2> {<config_data_list2>}
     ...

The returned information lists the following properties:
o config_name       The name of the configuration
o config_data_list  A list of name/value pairs describing the
                    following properties of the configuration:

                    description - A description of the configuration
                    or an empty string if none exists.

                    name - The name of the configuration.

                    selector -
                    + If the configuration is an alias, this property
                      contains the name of the referenced release.
                    + If the configuration is a branch configuration or
                      a selector configuration, this property is the
                      selector list (tag) associated with the
                      configuration that identifies the versions of the
                      DesignSync data that are part of the
                      configuration.
                    + If the configuration is a release, the selector
                      is {}.

                    type - The type of configuration. Available values
                    are as follows:
                    + Alias: Identifies a symbolic name for a release.
                      Aliases are created with the hcm mkalias
                      command.
                    + Branch: Identifies the configuration as a branch
                      configuration. Branch configurations are created
```

```
  with the hcm mkconf command.
+ Release: Identifies the configuration as a
  release. Releases are created with the hcm
  release command.
+ Selector: Identifies the configuration as a
  selector configuration. Selector configurations
  are created with the hcm mkconf command.
  ProjectSync configurations are shown as selector
  configurations.

exposure - The list of team members (usernames)
associated with the configuration. If the member
list is the default of all users defined on the
SyncServer, the exposure list is empty. This list
is managed via ProjectSync's Edit Project
Configuration.

owner - The owner of the configuration. If owner is
the only property in which you are interested, use
the url owner command.
```

Display of Configurations in a Work Area (Specified with -path)

```
When used with the -path option, the hcm showconfs command displays
the configuration in the work area directory you specified with
-path. When used with the -path option and the -directoryrecursive
option, the command displays all of the configurations found in a
directory hierarchy, starting with the directory specified with
-path.

You can run the hcm showconfs -path command to determine:
o If a directory in your work area belongs to a configuration
o Where in your work area a configuration's base directory resides
o Which configurations reside in a directory hierarchy in your
  work area

Notes:
o You cannot use the -path option with the -target option.
o If you specify neither the -target nor the -path option, the command
  uses -path option with the current directory as its value.
o If you use -path with -directoryrecursive, the command searches
  the directory hierarchy, starting with the directory specified
  with -path, and displays all configurations found.
o The -directoryrecursive option follows the directory hierarchy,
  not the module configuration hierarchy.
o If you use -path without -directoryrecursive and the specified
  path does not contain a configuration, the command searches up
  the directory hierarchy for a configuration. If it finds a
  configuration, the command displays the absolute path to that
  configuration.
```

Understanding the Output of hcm showconfs -path

```
By default, or if you run the hcm showconfs -path command with the
'-report normal' option and the -directoryrecursive option, the
command displays, in table format, a list of configurations found
in the work area directory hierarchy specified with -path.

The returned table contains the following information:
o PATH              The path to the work area base directory
                    containing the configuration.
o TARGET            The URL of the configuration.
o TYPE              The type of configuration. Type can be:
                    - Alias: Identifies a symbolic name for a release.
                      Aliases are created with the hcm mkalias
                      command.
                    - Branch: Identifies the configuration as a branch
                      configuration. Branch configurations are created
                      with the hcm mkconf command.
                    - Deliverable: Identifies the configuration as
                      an IP Gear Deliverable.
                    - Release: Identifies the configuration as a
                      release. Releases are created with the hcm
                      release command. Note: When the type is
                      Release, (mcache) indicates that the
                      directory is a link to a release in the
                      module cache.
                    - Selector: Identifies the configuration as a
                      selector configuration. Selector configurations
                      are created with the hcm mkconf command.
                      ProjectSync configurations are shown as selector
                      configurations.
o HIERARCHY         Indicates whether the entire hierarchy is
                    present in the directory.

Note: If you run the hcm showconfs -path command with the '-report
normal' option and you do not specify -directoryrecursive, the
command displays information about the configuration in a list of
name-value pairs.

If you run the hcm showconfs -path command with '-report script',
it returns a Tcl list of name-value pairs in the form of:

{path <path1> target <target1> selector <selector1> hierarchical
<hierarchical1> type <type1> mcachelink <mcachelink1>}{path <path2>
target <target2> selector <selector2> hierarchical <hierarchical2>
type <type2> mcachelink <mcachelink2>}

The returned information includes the following:
o path              The path to the work area directory containing
                    the configuration.
o target            The URL of the configuration.
o selector          The selector used to create the configuration.
                    If the configuration is an alias, this value is
                    the name of the referenced release.
                    If the configuration is a branch configuration or
                    a selector configuration, this property is the
                    selector list (tag) associated with the
                    configuration.
```

```
                    If the configuration is a release, the selector
                    is {}.
 o hierarchical     A value indicating whether the entire hierarchy
                    is present in the directory. 1 indicates that
                    the entire hierarchy is present. 0 indicates
                    that only the upper-level module is present (no
                    hierarchy).
 o type             The type of configuration. Available values
                    are: Alias, Branch, Deliverable, Release, and
                    Selector.
                    Note: When the type is Release, (mcached)
                    indicates that directory is a link to a release
                    in the module cache.
 o mcachelink       A value indicating whether the configuration in
                    the work area is a link to the configuration's
                    base directory in the module cache. 1 indicates
                    that the work area configuration is a link to
                    the module cache. 0 indicates that the work area
                    configuration is not a link to the module cache.

Note: If you run the hcm showconfs -path command with '-report
script' and the path supplied is not the base directory of a
configuration, the command returns an empty list.
```

SYNOPSIS

```
hcm showconfs [-directoryrecursive]
              [-report {brief | normal | verbose | script}]
              [-target <module_url> | -path <path>]
```

OPTIONS

- -directoryrecursive
- -path
- -report
- -target

-directoryrecursive

```
 -directoryrecursive    Specifies that the command follow the
                        directory hierarchy (starting with the
                        directory specified by -path) and display all
                        configurations found in the directory
                        hierarchy.

                        Notes:
                        - This option follows the directory
                          hierarchy, not the module configuration
                          hierarchy.
                        - Use this option with the -path option but
```

not with -target.

-path

-path	Indicates the path to a work area directory where the command starts the search for configurations. The path may be relative or absolute. If the specified path is absolute, the command displays absolute paths for all configurations found. If the specified path is relative, the command displays relative paths for all configurations found. If -path is used without -directoryrecursive, the command reports an absolute path for the configuration found. If neither -target nor -path is specified, the command uses the -path option with the current directory as it value.

-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 script - Returns a Tcl list of config_name/property_list pairs. Note: If you specify a release with the -target option, '-report script' also returns the creation time as expressed in seconds. (For all other configuration types, this report format returns an empty string.) Notes: - hcm showconfs -target displays configurations alphabetically according to configuration name.

```
                        - If you run hcm showconfs -target with
                          -report normal, the output includes the
                          default configuration (first line of output).
                        - hcm showconfs -path displays configurations
                          sorted by path.
```

-target

```
 -target <module_url>   Identifies the module for which you want to
                        view the configurations.

                        Specify the URL as follows:
                         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 for
                        which you want to view the configurations.

                        Notes:
                        - -target must resolve to a DesignSync vault.
                          Specifying just the SyncServer will result in
                          an error.
                        - If you specify a URL corresponding to a
                          configuration, the hcm showconfs command
                          fails.
                        - If you do not have access to browse the server
                          on which the module is located (set via
                          DesignSync's BrowseServer access control), the
                          hcm showconfs command fails.
                        - Do not use -target with -directoryrecursive.
                        - If neither -target nor -path is specified,
                          the command uses the -path option with the
                          current directory as it value.
```

RETURN VALUE

```
 If you run the hcm showconfs command with the '-report script'
 option, it returns a list of Tcl lists. Each element in the main
 list is a configuration, and sublists include name/value pairs of
 information about each configuration. If you specify '-report
 normal', or if you do not specify the -report option, the command
 does not it does not return any Tcl values. For a description of
 the output, see the "Understanding the Output" section.
```

SEE ALSO

```
 vhistory, tag, hcm showmods, hcm addhref, hcm mkmod,
 url owner, url properties
,
```

EXAMPLES

- Example Showing Configurations and Aliases on the Server
- Example Showing the Configurations in the Workspace

Example Showing Configurations and Aliases on the Server

```
This example displays the configurations and aliases of the Chip
module on the server.

dss> hcm showconfs  -target sync://chip.ABCo.com:2647/Projects/Chip

This command displays the following output:

Configurations of module sync://chip.ABCo.com:2647/Projects/Chip

NAME       TYPE      OWNER     SELECTOR/ALIASED RELEASE
-----------------------------------------------------
<Default>  Branch    rsmith    Trunk:Latest
Golden     Alias     bjones    R1
R1         Release   jdoe

Notice that because Chip@R1 is a release, the SELECTOR/ALIASED
RELEASE value is empty.
```

Example Showing the Configurations in the Workspace

```
This example displays the configurations in the Cpu work area directory
hierarchy.

dss> hcm showconfs -path Cpu -directoryrecursive

PATH     TARGET                                          TYPE        HIERARCHY
-------------------------------------------------------------------------------
Cpu      sync://cpu.ABCo.com:2647/Projects/Cpu@R1  Release           yes
Cpu/Alu  sync://alu.ABCo.com:2647/Projects/Alu@R5  Release (mcached) yes
Cpu/IO   sync://alu.ABCo.com:2647/Projects/IO@TEST Selector          no
Cpu/130  sync://ipgsrvr1.IOCo.com:2647/Deliverable/130 Deliverable   no

In this example:
o Because a relative path was specified for the -path option, the hcm
  showconfs command shows the relative path for each directory.
o (mcached) indicates that the directory is a link to a release in the
  module cache.
```

hcm showhrefs

hcm showhrefs Command

NAME

```
hcm showhrefs       - Displays the hierarchy of a module
                      configuration (legacy)
```

DESCRIPTION

- Understanding the Output

```
This command displays the hierarchical references for a module
configuration. When run recursively, this command displays the
hierarchical references for the upper-level module configuration and
all referenced submodule configurations, IP Gear deliverables, or
DesignSync vaults.

This command should not be used for non-legacy modules.  For
non-legacy modules, use the showhrefs command with no hcm prefix. The
command syntax for hcm showhrefs is different than the showhref
command syntax.

Note:
- This command obeys DesignSync's BrowseServer access control; if you
  do not have permission to browse a server, this command returns an
  error.

For this command to be successful, the following must be true:
- If you specify -target, the server on which the configuration resides
  must be available and the configuration specified must exist.
- If you specify -path, the path specified must exist.
- If you do not specify -target the directory specified by -path must
  contain an HCM configuration. If either -path is not specified or it
  does not contain a configuration, a parent directory in the
  file system hierarchy must contain an HCM configuration. The
  hcm showhrefs command looks first for a configuration in the
  directory identified by -path. If that directory does not contain a
  configuration, the hcm showhrefs command searches up the file system
  hierarchy until it finds a parent directory that does contain a
  configuration or it reaches the root of the file system. If it
  reaches the root of the file system without finding a configuration,
  the command fails.
```

Understanding the Output

```
By default, or if you run the hcm showhrefs command with the '-report
normal' option, the command displays the status of the hierarchical
references for the configuration in a user-friendly format.

The hcm showhrefs command, by default, displays the following
information:
o Reference         The leaf of the vault path. For example, if you
                    specified sync://host2:port2/Projects/ALU for
```

```
                    -totarget, the value for Reference would be ALU.
o URL               The complete URL of an object including host,
                    port, vaultPath, and optionally '@<config>'. If
                    the object is a configuration and '@<config>'
                    is absent, the default configuration is assumed.
o Relative Path     The relative path from the base directory of the
                    upper-level module configuration (Parent) to the
                    submodule configuration (Target). This path is used
                    by the hcm get command when you fetch the module
                    into your work area.

If you run the hcm showhrefs command with '-report brief' it displays
the following information:
o Target            The upper-level module configuration (first line
                    of output) and the submodule configurations,
                    DesignSync vaults, or IP Gear deliverables, to
                    which the upper-level module is connected. If
                    '@' is absent, the default configuration
                    is assumed.
o URL               The complete URL of an object including host,
                    port, vaultPath, and optionally '@<config>'.
                    If the object is a configuration and '@<config>'
                    is absent, the default configuration is assumed.
                    URL information is displayed if you run the
                    hcm showhrefs command with the -target option.
o Path              The local directory in which the module or
                    or IP Gear deliverable reside. Path information
                    is displayed unless you specify the -target
                    option. Note: A path marked as '(mcached)'
                    indicates that the local directory is a link to
                    a release in the module cache.

If you run the hcm showhrefs command with '-report script', it returns
a Tcl list in the following form:

   {relpath <relative_path1>  target <full_URL_object1>}
    {relpath <relative_path2>  target <full_URL_object2>}
    ...

Notes:
o The order of relpath and target might be reversed in your output.
o The output of hcm showhrefs -report script does not include
  information about module cache links.

If you run the hcm showhrefs command with '-report command', it
displays to the screen a list of hierarchical references in the
following form:

   -fromtarget <full_URL_uppermod@config>
     -totarget <full_URL_submod> -relpath <relative_path>

   - fromtarget is the URL of the upper-level module configuration for
     which the hierarchical references are shown. If the upper-level
     module configuration is a default configuration, the '@config'
     will be absent.

   - totarget is the URL of an IP Gear deliverable, DesignSync vault,
```

```
      or a submodule configuration to which the upper-level module
      configuration connects. If the totarget is a configuration, the
      URL will include '@config' for branch or selector configurations,
      aliases, and releases. If the totarget is a default configuration,
      '@config' will not be present.

   - relpath is the relative path from the base directory of the upper-
     level module configuration to the submodule configuration, IP Gear
     deliverable, or DesignSync vault. This path is used by the hcm get
     command when you fetch the module into your work area.

Note:
- relpath can be empty. An empty path is specified if you do not want
  to fetch the contents of the submodule into your work area.
  This behavior may be desirable if you want to reference a ProjectSync
  project that tracks defects against a CAD tool.

As hcm showhrefs recurses a hierarchy, it counts any errors and
displays suitable error messages. At the end of the operation,
hcm showhrefs displays an error message which contains the number of
errors that occurred.
```

SYNOPSIS

```
hcm showhrefs  [-recursive]
               [-report {brief | normal |verbose | command | script}]
               [-target <configuration_url> | -path <path>]
```

OPTIONS

- -path
- -recursive
- -report
- -target

-path

```
-path <path>          Identifies the directory containing the
                      configuration for which the hierarchical
                      references will be shown.

                      If neither -path or -target is specified, the
                      hierarchical references for the default
                      configuration associated with the current
                      directory will be shown.
```

-recursive

-recursive	Displays the hierarchical references for the upper-level module configuration and all submodule configurations.

-report

-report	Indicates the format in which the output appears. Valid values are: o brief - Displays the output in a user-friendly format such that the hierarchy of submodules is visible. 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 - Displays to the screen a list of hierarchical references in a format that closely matches the usage of the hcm rmhref command. With this format you can easily cut and paste to remove a hierarchical reference. o script - Returns a Tcl list. Note: - If you run the hcm showhrefs command with the -path option and the '-report brief' option, the command does not detect errors with referenced objects. When you run hcm showhrefs with the -path option, the server is not contacted.

-target

-target <configuration_url>	Specifies the URL of a configuration for which the hierarchical references will be shown. o To specify the default configuration of the upper-level module, use the following syntax: 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. o To specify a configuration other than the default configuration, use the following

```
                        syntax:
                        sync[s]://<host>[:<port>]/<vaultPath>@<config>
                        where <host> is the SyncServer on which the
                        module resides, <port> is the SyncServer port
                        number, and <vaultPath> identifies the module
                        and <config> identifies the specific
                        configuration.

                      Note:
                      - If you do not have access to browse the server
                        on which the module is located (set via
                        DesignSync's BrowseServer access control), the
                        hcm showhrefs command fails.
```

RETURN VALUE

```
If you run the hcm showhrefs 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

```
hcm addhref, hcm get, hcm release, hcm showmcache, hcm showstatus
,
```

EXAMPLES

- Displaying the hrefs Associated with a Configuration
- Understanding When Hierarchical References Are Updated

Displaying the hrefs Associated with a Configuration

```
The following examples assume the following hierarchy in your work
area:
Chip              <= uses the default configuration
   Cpu            <= uses the default configuration
   Alu@R1         <= R1 is already released
     Dpath@R2     <= R2 is already released

The "Creating, Fetching, and Releasing Module Hierarchies" example in
the hcm Example topic includes the steps for creating the modules,
configurations, and hierarchical references that are presented in
this example.

Displaying the Hierarchical References on the Server
----------------------------------------------------
This example displays to the screen the hierarchical references for
the default configuration of the Chip module.
```

```
dss> hcm showhrefs -target sync://chip.ABCo.com:2647/Projects/Chip \
         -report command

This command displays the following output:
-fromtarget sync://chip.ABCo.com:2647/Projects/Chip
 -totarget sync://alu.ABCo.com:2647/Projects/Alu@R1 -relpath Alu
-fromtarget sync://chip.ABCo.com:2647/Projects/Chip
 -totarget sync://cpu.ABCo.com:2647/Projects/Cpu -relpath Cpu

The output includes only the Cpu and Alu hierarchical references;
in this example the hcm showhrefs command was run without specifying
the -recursive option.

To show the entire directory structure, run the hcm showhrefs
command with the -recursive option as follows:

dss> hcm showhrefs -recursive \
                   -target sync://chip.ABCo.com:2647/Projects/Chip

This command displays the following output:

Target:         sync://chip.ABCo.com:2647/Projects/Chip

REFERENCE  URL                                          RELATIVE PATH
-------------------------------------------------------------------
Alu@R1     sync://alu.ABCo.com:2647/Projects/Alu@R1 Alu
Cpu        sync://cpu.ABCo.com:2647/Projects/Cpu    Cpu


====================================================================

Target:         sync://alu.ABCo.com:2647/Projects/Alu@R1
Parent:         sync://chip.ABCo.com:2647/Projects/Chip

REFERENCE  URL                                              RELATIVE PATH
-------------------------------------------------------------------------
Dpath@R2   sync://dpath.ABCo.com:2647/Projects/Dpath@R2 Dpath

Note: Target has no hierarchical references.
      Target: sync://dpath.ABCo.com:2647/Projects/Dpath@R2
Note: Target has no hierarchical references.
      Target: sync://cpu.ABCo.com:2647/Projects/Cpu

Alternately, to show the entire directory structure visually, run the
hcm showhrefs command with both -recursive and the '-report brief'
options as follows:

dss> hcm showhrefs -recursive -report brief \
                   -target sync://chip.ABCo.com:2647/Projects/Chip

This command displays the following output:

Hierarchy for sync://chip.ABCo.com:2647/Projects/Chip

TARGET           URL
-----------------------------------------------------------
```

```
Chip              sync://chip.ABCo.com:2647/Projects/Chip
    Alu@R1        sync://alu.ABCo.com:2647/Projects/Alu@R1
        Dpath@R2  sync://Dpath.ABCo.com:2647/Projects/Dpath@R2
    Cpu           sync://cpu.ABCo.com:2647/Projects/Cpu
```

Displaying the Hierarchical References in the Work Area

The previous example displayed the hierarchical references for the default configuration of the Chip module that resides on the server. To display the hierarchical references for the default configuration of the Chip module that was fetched into the work area substitute the -path option for the -target option as follows:

```
dss> hcm showhrefs -recursive -report brief \
                   -path /home/jsmith/Designs/Chip
```

This command displays the following output:

```
Hierarchy for sync://chip.ABCo.com:2647/Projects/Chip

TARGET            PATH
-------------------------------------------------
Chip              /home/jsmith/Designs/Chip
    Alu@R1        /home/jsmith/Designs/Chip/Alu
        Dpath@R2  /home/jsmith/Designs/Chip/Alu/Dpath
    Cpu           /home/jsmith/Designs/Chip/Cpu
```

Displaying Hierarchical References in the Work Area When It Contains Releases Fetched from the Module Cache

The previous example showed using hcm showhrefs with the -path option to display hierarchical references for the default configuration of the Chip module that was fetched into the work area directory /home/jsmith/Designs/Chip.

This example shows the use of the hcm showhrefs -path command when:

- The Alu@R1 release resides in the module cache, /home/ChipDev/cache.
- The Alu@R1 release in the module cache contains a hierarchical reference to the Dpath@R2 configuration, which is also a release.
- The default configuration of the Chip module was fetched to the work area by using hcm get in module cache link mode. (For information, see the hcm get command.)
- As a result of the fetch of the Chip module, the work area contains a link to the Alu@R1 release in the module cache.

```
dss> hcm showhrefs -recursive -report brief \
                   -path /home/jsmith/Designs/Chip
```

This command displays the following output:

```
Hierarchy for sync://chip.ABCo.com:2647/Projects/Chip

TARGET            PATH
-------------------------------------------------------
Chip              /home/jsmith/Designs/Chip
    Alu@R1        /home/jsmith/Designs/Chip/Alu (mcached)
        Dpath@R2  /home/ChipDev/cache/Alu/Dpath
```

```
    Cpu           /home/jsmith/Designs/Chip/Cpu

Notes:
o In this example, '(mcached)' indicates that the
  /home/jsmith/Designs/Chip/Alu directory is a link to the Alu@R1
  release in the module cache.
o Because Dpath@R2 is a submodule of the Alu@R1 release in the
  module cache, the path for Dpath@R2 shows its location in the
  module cache, not the work area.
```

Understanding When Hierarchical References Are Updated

```
This example assumes the following hierarchy in your work area:
Top               <= uses the default configuration
   Mem@DEV        <= DEV is a branch configuration
   IO@TEST        <= TEST is a selector configuration

The "Creating and Removing Module Configurations" example in the hcm
Example topic includes the steps for creating the modules,
configurations, and hierarchical references that are presented in the
following examples.

This example displays the hierarchical references associated with
the default configuration of the Top module and their status. It is
assumed that the default configuration of Top has previously been
fetched recursively to the work area directory,
/home/jsmith/Designs/Top, using the -recursive option to the hcm get
command.

After the default configuration of Top was fetched, the hierarchical
reference between Top and IO@TEST was removed on the server.

dss> hcm rmhref \
            -fromtarget sync://srvr1.ABCo.com:2647/Projects/Top \
            -totarget sync://srvr1.ABCo.com:2647/Projects/IO@TEST

It is further assumed that after the hierarchical reference between
Top and IO@TEST was removed on the server the work area was not
fetched again.

To display the hierarchical references for the default Top
configuration, run the following command:

dss> hcm showhrefs -recursive -path /home/jsmith/Designs/Top

Note:
-This command displays the hierarchical references that are in your
 work area. Notice that even though the reference between the
 default configuration of Top and IO@TEST was removed on the server,
 it still exists in your work area.
-The specified path can be relative.

This command displays this following output:
```

```
Target:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top

REFERENCE  URL                                              RELATIVE PATH
-----------------------------------------------------------------------
IO@TEST    sync://srvr1.ABCo.com:2647/Projects/IO@TEST IO
Mem@DEV    sync://srvr1.ABCo.com:2647/Projects/Mem@DEV Mem

Note: Target has no hierarchical references.
      Target: sync://srvr1.ABCo.com:2647/Projects/IO@TEST
      Path:   /home/jsmith/Designs/Top/IO
Note: Target has no hierarchical references.
      Target: sync://srvr1.ABCo.com:2647/Projects/Mem@Dev
      Path:   /home/jsmith/Designs/Top/Mem

To determine if the hierarchical references for the default Top
configuration in the work area are current, run the hcm showstatus
command.
```

hcm showmcache

hcm showmcache Command

NAME

```
hcm showmcache      - Lists the releases in module caches
```

DESCRIPTION

- Understanding the Output

```
This command searches one or more specified module caches and
returns a list of releases that can be used by the hcm get command
in link or copy mode. Module caches to be searched can be specified
with the -mcachepaths option or the DesignSync registry setting for
default module cache paths. (The command searches the module caches
in the order specified with the -mcachepaths option or the registry
setting.)

This command should not be used for non-legacy modules.  For
non-legacy modules, use the showmcache command with no hcm prefix. The
command syntax for hcm showmcachs is different than the showmcache
command syntax.

When you use the hcm showmcache command but do not specify the
-mcachepaths option, the get operation uses the path list specified
in the default module cache paths registry setting to locate the
module caches. (Note: To override the default module cache paths,
you can use the -mcachepaths option to specify the module cache
paths you want.)
```

Understanding the Output

```
By default, or if you run the hcm showmcache command with the
'-report normal' option, the command displays a list of module
cache paths searched, followed by a list of releases in the module
caches. The table is presented in order of module cache paths
searched and entries for each module cache are sorted by PATH.

The table includes the following information:

o PATH              The module cache directory in which the
                    release resides.
o TARGET            The URL of the release configuration on the
                    server.
o AVAILABLE         Whether the release is available for linking or
                    copying. A release might be unavailable if, for
                    example, it is being fetched to the module
                    cache.
o HIERARCHY         Whether the entire hierarchy of the release is
                    present in the module cache directory. 'yes'
                    indicates that the entire hierarchy is
                    present. 'no' indicates that only the
                    upper-level module is present (no hierarchy).

If you run the hcm showmcache command with '-report script', it returns
a Tcl list of configurations in the following form:

{target <target1> path <path1> type <type1> selector <selector1>
hierarchical <hierarchical1> available <available1>}{target
<target2> path <path2> type <type2> selector <selector2>
hierarchical <hierarchical2> available <available2>}

The returned information lists the following properties:

o target        The URL of the release configuration on the server.

o path          The module cache directory in which the release
                resides.

o type          The type of configuration. (Note: The type is always
                Release; the module cache contains only releases.)

o selector      The selector name for the configuration. (Note: The
                selector for a release is always {}.)

o hierarchical  A value indicating whether the entire hierarchy is
                present in the module cache directory. 1 indicates
                that the entire hierarchy is present. 0 indicates
                that only the upper-level of the module is present
                (no hierarchy).

o available     A value indicating whether the release is available
                for use by the hcm get operation in link or copy
```

```
                mode. 1 indicates the release is available for use;
                0 indicates the release is not available. A release
                might be unavailable if, for example, it is being
                fetched to the module cache.
```

SYNOPSIS

```
hcm showmcache [-mcachepaths <path_list>]
               [-report {brief | normal |verbose | script}]
```

OPTIONS

- -mcachepaths
- -report

-mcachepaths

```
-mcachepaths          Identifies one or more module caches to
 <path_list>          search.

                      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"

                      If -mcachepaths is absent, the command uses
                      the list of paths defined in the registry
                      setting.

                      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 searches the module caches in
                        the order specified with the -mcachepaths
                        option or in the default module cache paths
                        registry setting if this option is absent.
                        (For information about the registry setting,
                        see "Setting the Default Module Cache Path
                        or Mode" in HCM Help.)
```

-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 table
                                   format. This is the default behavior.

                        o verbose - Displays the same information as
                          'normal'.

                        o script - Returns a Tcl list of name/value pairs.
```

RETURN VALUE

If you run the hcm showmcache command with the '-report script' option, it returns a Tcl list of release configurations. Each list element contains a list of name/value pairs representing data about the configuration. If you specify '-report normal' or if you do not specify the -report option, the command does not return any Tcl values. For a description of the output, see the "Understanding the Output" section.

SEE ALSO

hcm get

EXAMPLES

This example displays to the screen in table format the releases available in two module caches, /A5/cache and /B1/cache:

```
dss> hcm showmcache -mcachepaths "/A5/cache /B1/cache" -report normal
```

This example displays the following output:

```
Mcachepaths search order:

/A5/cache/CPU
/B1/cache/ALU

Configurations found:

PATH           TARGET                                    AVAILABLE  HIERARCHY
-------------------------------------------------------------------------
```

```
/A5/cache/CPU  sync://cpu.ABCo.com:2647/Projects/Cpu@Rel1   yes    yes
/A5/cache/SLIB sync://svr1.ABCo.com:2647/Projects/STDLIB@R1 yes    no
/B1/cache/ALU  sync://alu.ABCo.com:2647/Projects/Alu@Rel2   yes    yes
```

hcm showmods

hcm showmods Command

NAME

```
hcm showmods        - Displays the modules available on a server
```

DESCRIPTION

- Understanding the Output

```
This command displays the modules available on a given server.

This command should not be used for non-legacy modules.  For
non-legacy modules, use the showmods command with no hcm prefix. The
command syntax for hcm showmods is different than the showmods
command syntax.

Note:
- This command obeys DesignSync's BrowseServer access control; if you
  do not have permission to browse a server, this command displays an
  error.
```

Understanding the Output

```
By default, or if you run the hcm showmods command with the '-report
normal' option, the command displays a list of modules available on
the specified server in a user-friendly format.

The returned table includes the following information:
o NAME              The name of the module.
                    Note:
                    - Modules are displayed alphabetically according to
                      their names.
o OWNER             The user name of the person who created the module.
o VAULT PATH        The vault directory in which the files associated
                    with the module reside.

If you run the hcm showmods command with '-report script', it returns
a Tcl list in the following form:

   {description <description1> vault_path <vaultPath1>
```

```
     name <module_name1> owner <owner1> url <module_URL1>}
    {description <description2> vault_path <vaultPath2>
     name <module_name2> owner <owner2> url <module_URL2>}
    ...
```

The returned information lists the following properties:

o description	A brief text description of the module. This value can be set when you create your module (specified by the -description option of the hcm mkmod command).
o vault_path	The vault directory in which the files associated with the module reside.
o module_name	The name of the module.
o owner	The user name of the person who created the module.
o module_URL	The complete URL of a module including host, port, and vaultPath.

SYNOPSIS

```
hcm showmods  [-report {normal | brief | verbose}]
               -target <server_url>
```

OPTIONS

- -report
- -target

-report

-report	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'.

-target

-target <server_url>	Specifies the URL of the DesignSync server for which you want to see the available modules.

```
Specify the Synchronicity URL as follows:
 sync[s]://<host>[:<port>]
where <host> is the SyncServer on which the
module resides, <port> is the SyncServer port
number.

Note:
- If you do not have access to browse the server
  on which the module is located (set via
  DesignSync's BrowseServer access control), the
  hcm showmods command fails.
```

RETURN VALUE

```
If you run the hcm showmods 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

```
hcm mkmod
,
```

EXAMPLES

- Example Showing Modules on the Specified Server
- Example Showing Modules in TCL List Form on the Specified Server

```
These examples list the modules available on the DesignSync server
sync://srvr1.ABCo.com:2647. The hcm showmods command is run with
both '-report normal' and '-report script' to show the different
output.

The "Creating and Removing Module Configurations" example in the hcm
Example topic includes the steps for creating the modules that are
presented in this example.
```

Example Showing Modules on the Specified Server

```
This example displays to the screen in a user-friendly format the
modules available on the DesignSync server sync://srvr1.ABCo.com:2647.

dss> hcm showmods -target sync://srvr1.ABCo.com:2647 -report normal
```

```
This command displays the following output:
Modules hosted on server sync://srvr1.ABCo.com:2647

NAME    OWNER   VAULT PATH
--------------------------------
IO      jsmith  /Projects/IO
Mem     jsmith  /Projects/Mem
Top     jsmith  /Projects/Top
```

Example Showing Modules in TCL List Form on the Specified Server

```
This example returns to the screen a Tcl list of the modules available
on the DesignSync server sync://srvr1.ABCo.com:2647.

dss> hcm showmods -target sync://srvr1.ABCo.com:2647 -report script

This command displays the following output:
{description {HCM Module for IO} vault_path /Projects/IO name IO
owner jsmith url sync://srvr1.ABCo.com:2647/Projects/IO}
{description {HCM Module for Mem} vault_path /Projects/Mem name Mem
owner jsmith url sync://srvr1.ABCo.com:2647/Projects/Mem} {description
{Upper-level HCM (Top) Block} vault_path /Projects/Top name Top
owner jsmith url sync://srvr1.ABCo.com:2647/Projects/Top}

The correct usage of the hcm showmods command with the '-report script'
option is as follows:

stcl> catch {hcm showmods -target sync://srvr1.ABCo.com:2647 \
             -report script} module_list
0
stcl> foreach m $module_list {puts $m}
description {HCM Module for IO} vault_path /Projects/IO name IO owner
jsmith url sync://srvr1.ABCo.com:2647/Projects/IO
description {HCM Module for Mem} vault_path /Projects/Mem name Mem
owner jsmith url sync://srvr1.ABCo.com:2647/Projects/Mem
description {Upper-level HCM (Top) Block} vault_path /Projects/Top
name Top owner jsmith url sync://srvr1.ABCo.com:2647/Projects/Top
```

hcm showstatus

hcm showstatus Command

NAME

```
hcm showstatus      - Displays the status of a configuration
                      in your work area
```

DESCRIPTION

- Understanding the Output

```
This command lists the status of the hierarchical references of a
configuration in your local work area as compared to that
configuration on the server.

This command should not be used for non-legacy modules.  For
non-legacy modules, use the showstatus command with no hcm
prefix. The command syntax for hcm showstatus is different than the
showstatus command syntax.

Using this command, you can verify that your work area
configuration is up-to-date. If used with the -recursive option,
'hcm showstatus' lists status of hierarchical references at all
levels of the hierarchy. If used with the -files option, 'hcm
showstatus' lists status of all the files of a configuration in
combination with the status of its hierarchical references.

For this command to be successful, the following must be true:
- If the -path option is supplied, the specified directory must exist.
- The directory specified by -path, or the current directory if -path
  is not specified, must contain an HCM configuration. If either -path
  is not specified or it does not contain a configuration, a parent
  directory in the file system hierarchy must contain an HCM
  configuration. The hcm showstatus command looks first for a
  configuration in the directory identified by -path. If that directory
  does not contain a configuration, the hcm showstatus command searches
  up the file system hierarchy until it finds a parent directory that
  does contain a configuration or it reaches the root of the file
  system. If it reaches the root of the file system without finding a
  configuration, the command fails.
- The server on which the configuration resides must be available.

This command obeys DesignSync's BrowseServer access control; if you
do not have permission to browse a server, this command returns an
error.
```

Understanding the Output

```
By default, or if you run the hcm showstatus command with the '-report
normal' option, the command displays the status of the hierarchical
references for the configuration in a user-friendly format.

Note: Status of the configuration can differ depending on whether
you specify the -files option or not. If '-files' is not specified,
the status reported reflects the status of the hierarchical
references only. If '-files' is specified, the status reported
reflects the status of the objects contained in the configuration
as well as the status of its hierarchical references.

The hcm showstatus command, by default, displays the following
information:
o Configuration:   Identifies the configuration for which the status
```

```
                    is shown. This value is a Synchronicity URL.

o Base Directory:   Identifies the local file system directory in which
                    the configuration resides. Each module in a
                    hierarchy has its own base directory.

o Information about each hierarchical reference:
  - STATUS          The status of the hierarchical reference of the
                    configuration in the work area as compared to
                    the server. Possible values are:
     Local Only     Indicates the hierarchical reference exists
                    only in  the local work area.
     Out-of-date    Indicates that the hierarchical reference in
                    the local work area does not match the
                    hierarchical reference on the server; for
                    example, an alias on the server may have
                    changed to refer to a new release or the
                    relative path of the hierarchical reference on
                    the server may have changed.
     Server Only    Indicates the hierarchical reference was added
                    on the server.
     Unknown        Indicates that status of the hierarchical
                    reference cannot be determined. This status is
                    displayed only if you specify a recursive
                    showstatus operation. For example, if you
                    specify 'hcm showstatus -recursive' and the
                    server on which a referenced configuration
                    resides is unavailable, the showstatus operation
                    lists the status for that configuration as Unknown.
     Up-to-date     Indicates that the hierarchical reference in the
                    local work area matches the hierarchical
                    reference on the server.
  - HREF            Identifies the submodule to which the
                    configuration of the upper-level module is
                    connected. This value is a Synchronicity URL.
  - RELATIVE PATH   Identifies the path from the upper-level module
                    to the submodule.

o Configuration status
                    Indicates the status of the configuration.
                    (Displayed only when you specify the -recursive
                    option.)

o Summary           Indicates the overall status of the
                    configuration. By default (or if you specify the
                    '-report normal' option), this value is a
                    summary of the status of the configuration's
                    hierarchical references. Note: If you specify
                    the -files option, this value represents the
                    status of the objects contained in the
                    configuration in combination with the status of its
                    hierarchical references.
                    Possible values are:
   - Local Only     Indicates the configuration exists only in
                    the local work area.
   - Out-of-date    Indicates that the configuration in
                    the local work area does not match the
```

```
                  configuration on the server.
   - Server Only  Indicates the configuration was added
                  on the server.
   - Unknown      Indicates that status of the configuration
                  cannot be determined. This status is
                  displayed, for example, if the server on which
                  the configuration resides is unavailable.
   - Up-to-date    Indicates that the configuration in
                  the local work area matches the hierarchical
                  reference on the server.

To show the status of the objects contained in your work area
configuration (as compared to objects in the configuration on the
server), you can use the hcm showstatus command with the -files
option. Output from the command first shows the status of the
configuration's hierarchical references (as described above) and
then shows the status of its objects. When '-files' is specified,
the value for configuration status reflects the status of its
objects in combination with the status of its hierarchical
references.

Information for each object includes:

o Workspace Version       Identifies the version of the object in
                          the work area configuration. "Unmanaged"
                          indicates the object is not managed by
                          DesignSync; "Unknown" indicates that the
                          status cannot be determined. (For
                          example, the command might display
                          "Unknown" for an object if the server is
                          not available.) If no information is
                          displayed, it indicates that the version
                          is absent from the work area.

o Configuration Version   Identifies the version of the object in the
                          configuration on the SyncServer.

o Object Name             Identifies the name of the object for
                          which status information is given.

If you use the hcm showstatus command with '-report script', it returns
a Tcl list in the following form:

target <module_URL>
relpath <relative_path>
[notes {
         "Old aliased release: <release_name>" |
         "New aliased release: <release_name>" |
         "Relative path changed to '<path>'" |
         "Cannot determine current value of alias on server." |
         <miscellaneous other information>
       }
]
status <Local Only | Out-of-date | Server Only | Unknown | Up-to-date>
hierstatus <Local Only | Out-of-date | Server Only | Unknown | Up-to-date>
[hrefs {{<submodule_status>} {...}}] ...
```

```
The returned information includes the following:

o target            The URL of the module configuration.
o relpath           The relative path from the base directory of the
                    upper-level module configuration (Parent) to the
                    submodule configuration (Target). This path is used
                    by the hcm get command when you fetch the module
                    into your work area.

o status            The status of the configuration in your work
                    area (as compared to the configuration on the
                    server). Note: This status reflects the status
                    of the configuration's hierarchical
                    references. If the -files option is specified,
                    status reflects the status of configuration's
                    hierarchical references and objects.

                    Possible values are:
   - Local Only     Indicates the hierarchical reference exists
                    only in the local work area.
   - Out-of-date    Indicates that the hierarchical reference in
                    the local work area does not match the
                    hierarchical reference on the server; for
                    example, an alias on the server could have
                    changed to refer to a new release or the
                    relative path of the hierarchical reference
                    on the server could have changed.
   - Server Only    Indicates the hierarchical reference was added
                    on the server.
   - Unknown        Indicates that status of the hierarchical
                    reference cannot be determined. This status is
                    displayed, for example, if the server on which
                    the configuration resides is unavailable.
   - Up-to-date     Indicates that the hierarchical reference in
                    the local work area matches the hierarchical
                    reference on the server.
o hierstatus        Indicates the overall status of the
                    configuration. By default (or if you specify the
                    '-report normal' option), this value is a summary
                    of the status of the configuration's
                    hierarchical references. If you specify
                    the -files option, this value represents the
                    status of the objects contained in the
                    configuration as well as the status of its
                    hierarchical references. If you specify the
                    -recursive option, the value indicates the
                    status of the entire configuration hierarchy.
                    Possible values are:
   - Out-of-date    Indicates that the configuration in
                    the local work area does not match the
                    configuration on the server.
   - Unknown        Indicates that status of the configuration
                    cannot be determined. This status is
                    displayed, for example, if the server on which
                    the configuration resides is unavailable.
   - Up-to-date     Indicates that the configuration in
                    the local work area matches the hierarchical
```

```
                    reference on the server.
o hrefs             A list of property lists, one for each of the
                    configuration's hierarchical references.
                    (Displayed only if the configuration has
                    hierarchical references.)

Note:
o Your output will include 'notes' if an alias in your work area is
  out of date with respect to the server. For example, if DRAM@Silver
  initially references the DRAM@R1 configuration and the alias is
  changed such that DRAM@Silver now references the DRAM@R2
  configuration, your output would include the following notes:

  notes {{Old aliased release: R1} {New aliased release: R2}}

o When you run the hcm showstatus command recursively, your output
  will include 'hrefs' status for each submodule containing
  hierarchical references. The <submodule_status> is a Tcl list of
  the module status information.

To show the status of the objects contained in your work area
configuration, use the hcm showstatus command with the -files and
the '-report script' options. Output from the command lists the
status of the configuration and its hierarchical references (as
described above). In addition, the output includes a Tcl list
(content) that describes the status of each of the objects
contained in the configuration.

target <module_URL>
relpath <relative_path>
[notes {{Old aliased release: <release_name>}
        {New aliased release: <release_name>}}]
status <Local Only | Server Only | Up-to-date | Unknown | Out-of-date>
content
       {
         path1 <path>
         path2 <URL>
         type <folder | file>
         objects
             {
                {
                   name <object_name>
                   type <file | folder>
                   objects {object_list}
                   name <file_name> type file
                   props1
                       {
                         state <absent | modified | present | reference
                                | unknown | unmanaged>
                         version <version_number>
                       }
                   props2
                       {
                         state <absent | modified | present | reference
                                | unknown | unmanaged>
                         version <version_number>
                       }
```

```
                }
             }
       }

   hierstatus <Up-to-date | Out-of-date | Unknown>
   [hrefs {{<submodule_status>} {...}}] ...

o content           Lists the objects in the configuration
                    and reports the status of each. (Displayed
                    only if you specify the -files option with
                    the hcm showstatus command.)
o path1             The path to the work area directory containing
                    the configuration.
o path2             The URL of the configuration on the server.
o type              The type of object contained in the work area
                    path (folder or file).
o objects           A list of objects and their properties.
                    (Displayed only when type is
                    folder.) For each object, the following
                    information is provided:
   - name           The name of the object
   - type           The object's type (folder or file)
   - props1 {...} props2 {...}
                    Properties of the objects in the configuration.
                    (Displayed only when object type is file.)
                    Properties are:
                    o version - The version number of the object.
                      (In certain cases, this property may not be shown.)
                    o state - The status of the object in the path
                      (your work area or configuration on the server.)
                      Possible values are:
                       - absent - Indicates that the object is not
                         present on this path (work area or
                         configuration on the server).
                       - modified - Indicates that the object has been
                         locally modified.
                       - present - Indicates that the object is
                         present on this path. (The version that is
                         present is reported in the version property.)
                       - reference - Indicates that the object is a
                         referenced object.
                       - unknown - Indicates that the object exists
                         in the work area but the fetched version
                         is unknown. This state is most commonly
                         reported when an object has been removed from
                         the workspace (with the rmfile command)
                         and then recreated.
```

SYNOPSIS

```
hcm showstatus [-files] [-path <path>] [-recursive] [-releases]
               [-report {brief | normal |verbose | summary]
```

OPTIONS

- files
- -path
- -recursive
- -releases
- -report

files

-files	Lists the status of each object contained by the work area configuration as compared to the configuration on the server. If you specify the -files option, the value for configuration status reflects the status of the configuration's objects in combination with the status of its hierarchical references. If you do not specify '-files', the value for configuration status reflects the status of the hierarchical references only.

-path

-path <path>	Identifies the directory containing the configuration for which the status will be shown. If -path is absent, the status for the configuration associated with the current directory ('./') will be shown.

-recursive

-recursive	Displays the status for the specified configuration and all referenced configurations and identifies why particular hierarchical references are not recursed. Notes: - The hcm showstatus command does not recurse into hierarchical references that point to IP Gear deliverables, non-HCM servers, aliases, or references that do not exist locally. - If you run the hcm showstatus command with the '-report script' option, the hcm showstatus command captures all errors encountered in the hierarchy and displays the errors after it completes.

-releases

-releases	Lists the status of hierarchical references of the releases in your work area, in addition to the status of hierarchical references of other configurations. Note: You must use this option with the -files option in order to display the current status of the hierarchical references of releases in your work area. Otherwise, the status of hierarchical references of releases is always listed as up-to-date.

-report

-report <mode>	Specifies the type of status information to be displayed and the format in which the output appears. Valid values are: o brief - Displays a summary and lists hierarchical references that are out-of-date. (Optionally, this mode also lists file status for files that are out-of-date.) o normal - Displays the status of the hierarchical references (and optionally, file status) for the configuration in a user-friendly format. This is the default behavior. (For a description of the status information, see "Understanding the Output".) o verbose - Displays the same information as 'normal'. o summary - Displays (in a user-friendly format) the status of each configuration and reports the overall status of the configuration in the work area.

RETURN VALUE

If you run the hcm showstatus 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

```
hcm addhref, hcm get, hcm release, hcm rmhref, compare, ls
,
```

EXAMPLES

```
o Show the status of hierarchical references of a module
  configuration hierarchy in the work area as compared to the
  server:

  dssc> hcm showstatus -recursive -path /home/username/Designs/Top

o Show the status of hierarchical references and objects contained in
  a module configuration hierarchy in the work area as compared to
  the server:

  dssc> hcm showstatus -recursive -files -path /home/username/Designs/Top

o Show the status of each configuration in the module configuration
  hierarchy, followed by a summary of the overall status of the
  hierarchy. (Note: Because '-files' is specified, each configuration's
  status represents the status of the configuration's hierarchical
  references and its objects.)

  dssc> hcm showstatus -recursive -files -report summary \
        -path /home/username/Designs/Top

-------------
SAMPLE OUTPUT
-------------
Example 1:
============================================================
This example assumes the following hierarchy in your work area:
Top             kept in directory Designs/Top
   IO@TEST      kept in directory Designs/Top/IO
   Mem@DEV      kept in directory Designs/Top/Mem

"Example 2: Creating and Removing Module Configurations" in the
hcm Example topic includes the steps for creating the modules,
configurations, and hierarchy presented in these examples.

Example 1a
----------
This example lists the status of the hierarchical references in your
local work area as compared to the server.

dss> hcm showstatus -recursive -path /home/jsmith/Designs/Top
```

```
This command displays the following output:

Target:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top

STATUS      HREF                                              RELATIVE PATH
-------------------------------------------------------------------------
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/IO@TEST  IO
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Mem@DEV  Mem

Configuration status: Up-to-date

============================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/IO@TEST
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/IO

No local or remote hierarchical references found for configuration.

Configuration status: Up-to-date

============================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/Mem

No local or remote hierarchical references found for configuration.

Configuration status: Up-to-date

============================================================

Status of all visited configurations.
STATUS      TARGET                                          PATH
-------------------------------------------------------------------
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
/home/jsmith/Designs/Top/Mem
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/IO@TEST
/home/jsmith/Designs/Top/IO
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Top
/home/jsmith/Designs/Top

Summary: Up-to-date

Example 1b
----------
This example shows output of an hcm showstatus operation where the
hierarchical references of the configuration hierarchy in the work
area are up-to-date but a file in one of those configurations is not.

stcl> hcm showstatus -recursive -files -report normal

Target:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top
```

```
STATUS      HREF                                                  RELATIVE PATH
-------------------------------------------------------------------------------
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/IO@TEST   IO
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Mem@DEV   Mem

Workspace             Configuration          Object
Version               Version                Name
---------             -------------          ------
1.1                   1.1                    Top.v

Configuration status: Up-to-date

===============================================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/IO@TEST
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/IO

No local or remote hierarchical references found for configuration.

Workspace             Configuration          Object
Version               Version                Name
---------             -------------          ------
1.1                   1.1                    IO1.v

Configuration status: Up-to-date

===============================================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/Mem

No local or remote hierarchical references found for configuration.

Workspace             Configuration          Object
Version               Version                Name
---------             -------------          ------
1.1                   1.2                    Mem1.v

Configuration status: Out-of-date

===============================================================================

Status of all visited configurations.
STATUS      TARGET                                                PATH
-------------------------------------------------------------------------------
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/IO@TEST
/home/jsmith/Designs/Top/IO
Out-of-date sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
/home/jsmith/Designs/Top/Mem
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Top
/home/jsmith/Designs/Top

Summary: Out-of-date
```

```
Example 1c
----------
This example lists the status of the hierarchical references in your
work area as compared to the server. It assumes that after you
recursively fetched the default configuration of the Top module into
your work area, the hierarchical reference between the default
configuration of the Top module and the TEST configuration of the IO
module was removed.

dss> hcm rmhref \
         -fromtarget sync://srvr1.ABCo.com:2647/Projects/Top \
         -totarget sync://srvr1.ABCo.com:2647/Projects/IO@TEST

Until you recursively fetch the default configuration of the Top
module again, these changes do not appear in your work area. If you
run the hcm showstatus command, the output shows the Top module's
configuration status as "Out-of-date" because its hierarchical
reference to the IO@TEST configuration exists in the work area
only, not on the server. Because the Top module's configuration
status is "Out-of-date", the summary status for the entire Top
configuration hierarchy is also shown as "Out-of-date".

dss> hcm showstatus -recursive -path /home/jsmith/Designs/Top

This command displays the following output:

Target:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top

STATUS      HREF                                              RELATIVE PATH
------------------------------------------------------------------------
Local Only  sync://srvr1.ABCo.com:2647/Projects/IO@TEST  IO
Up-to-date  sync://srvr1.ABCo.com:2647/Projects/Mem@DEV  Mem

Configuration status: Out-of-date

========================================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/IO@TEST
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/IO

No local or remote hierarchical references found for configuration.

Configuration status: Up-to-date
========================================================================

Target:         sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
Parent:         sync://srvr1.ABCo.com:2647/Projects/Top
Base Directory: /home/jsmith/Designs/Top/Mem

No local or remote hierarchical references found for configuration.

Configuration status: Up-to-date

============================================================
```

```
Status of all visited configurations.
STATUS      TARGET                                              PATH
-------------------------------------------------------------------
Up-to-date sync://srvr1.ABCo.com:2647/Projects/IO@TEST
/home/jsmith/Designs/Top/IO
Up-to-date sync://srvr1.ABCo.com:2647/Projects/Mem@DEV
/home/jsmith/Designs/Top/Mem
Out-of-date  sync://srvr1.ABCo.com:2647/Projects/Top
/home/jsmith/Designs/Top

Summary: Out-of-date

Example 2:
============================================================
This example assumes the following hierarchy in your work area:

Block
    BIST@TEST
    Mem2
       DRAM@Silver
       Collar@DEV

This example assumes the following hierarchy on the server:

Block
    BIST@TEST
    IO2
    Mem2
       DRAM@Silver
       Collar@DEV

Notice that an IO2 module was added to the server since you fetched
the Block module. Also, the BIST@TEST module has been deleted, but
not the hierarchical references from the default configuration of
the Block module to BIST@TEST.

"Example 3: Creating, Modifying, and Putting Configurations" in the
hcm Example topic includes the steps for creating the modules,
configurations, and hierarchy presented in these examples.

This example lists the status of the hierarchical references in your
work area as compared to the server.

dss> hcm showstatus -recursive -path /home/jsmith/Designs/Block

This command displays the following output:

Target:         sync://srvr2.ABCo.com:2647/Projects/Block
Base Directory: /home/jsmith/Designs/Block

STATUS      HREF                                                RELATIVE PATH
-------------------------------------------------------------------------
Up-to-date  sync://srvr2.ABCo.com:2647/Projects/BIST@TEST  BIST
Server Only sync://srvr2.ABCo.com:2647/Projects/IO2        IO2
Up-to-date  sync://srvr2.ABCo.com:2647/Projects/Mem2       Mem2

The status of the following hrefs will not be individually reviewed:
```

```
     HREF                                              REASON
     ------------------------------------------------------
     sync://srvr2.ABCo.com:2647/Projects/IO2     not local

Configuration status: Out-of-date
Error: Failed to get hrefs for target.
       target: sync://srvr2.ABCo.com:2647/Projects/BIST@TEST
       Vault 'sync://srvr2.ABCo.com:2647/Projects/BIST' does not
       exist.

================================================================================

Target:         sync://srvr2.ABCo.com:2647/Projects/Mem2
Parent:         sync://srvr2.ABCo.com:2647/Projects/Block
Base Directory: /home/jsmith/Designs/Block/Mem2

STATUS      HREF                                                  RELATIVE PATH
------------------------------------------------------------------------------
Up-to-date  sync://srvr2.ABCo.com:2647/Projects/Collar@DEV   Collar
Up-to-date  sync://srvr2.ABCo.com:2647/Projects/DRAM@Silver  DRAM

The status of the following hrefs will not be individually reviewed:

     HREF                                                REASON
     ----------------------------------------------------------
     sync://srvr2.ABCo.com:2647/Projects/DRAM@Silver  alias

Configuration status: Up-to-date

================================================================================

Target:         sync://srvr2.ABCo.com:2647/Projects/Collar@DEV
Parent:         sync://srvr2.ABCo.com:2647/Projects/Mem2
Base Directory: /home/jsmith/Designs/Block/Mem2/Collar

No local or remote hierarchical references found for configuration.

Configuration status: Up-to-date

================================================================================

Status was not computed for the following configurations.
See above for details.

    sync://srvr2.ABCo.com:2647/Projects/DRAM@Silver
    sync://srvr2.ABCo.com:2647/Projects/IO2

================================================================================

Status of all visited configurations.
STATUS      TARGET                                                      PATH
------------------------------------------------------------------------------
Unknown     sync://sync://srvr2.ABCo.com:2647/Projects/BIST@TEST
/home/jsmith/Designs/Block/BIST
Up-to-date  sync://srvr2.ABCo.com:2647/Projects/Mem2
/home/jsmith/Designs/Block/Mem2
```

```
Up-to-date  sync://srvr2.ABCo.com:2647/Projects/Collar@DEV
/home/jsmith/Designs/Block/Mem2/Collar
Out-of-date sync://srvr2.ABCo.com:2647/Projects/Block
/home/jsmith/Designs/Block

Summary: Unknown

Error: Errors were encountered for 1 hierarchical element(s).
       Review output for details.
```

The hcm rmmod command does not remove hierarchical connections from configurations of other modules to a removed module. Prior to the execution of the hcm showstatus command in this example, the BIST module was removed. The hierarchical reference between the default configuration of the Block module and the BIST@TEST configuration was not removed. For this reason, when you run the hcm showstatus command, the output shows an error for BIST@TEST and the status of BIST@TEST is Unknown. In addition, because the status of BIST@TEST is Unknown, the Summary status is also Unknown. To clear the error and resolve the Unknown status, you need to remove this hierarchical reference from the configurations of other modules.

hcm upgrade

hcm upgrade Command

NAME

```
hcm upgrade          - Upgrades a DesignSync vault to a legacy module
                       (legacy)
```

DESCRIPTION

This command upgrades a DesignSync vault path to an HCM module. It converts the specified vault path to an HCM module, searches for sync_project.txt files occurring within the vault directory structure below the specified vault path, and if they are found, creates hierarchical references for the REFERENCE keyword and each CONFIG keyword in each sync_project.txt file found. Many configurations for the module may be implicitly created by the upgrade process. See the Examples section below for more details about how sync_project.txt files are processed.

IMPORTANT: This command does not upgrade your legacy module or your DesignSync vault to the modern modules format. To upgrade to modern modules, use the "upgrade" command without the "hcm" prefix.

When hcm upgrade is used on a vault folder, in addition to creating an HCM module from the specified vault folder, the hcm upgrade command:

- Updates existing email subscriptions associated with the vault path that is being converted into a module. As the email subscriptions only apply to the newly created module, not its submodules, HCM sets

```
  the Scope field of the subscriptions to "This object only". HCM also
  sets the Scope field to "This object only" for existing subscriptions
  to any configurations of the module.
- For each existing subscription associated with the vault path,
  creates a new subscription covering the entire vault structure below
  the vault folder that is being converted into a module.
For information about email subscriptions, see the HCM Help topic, "The
CPU Team Subscribes to Email on a Hierarchy."

Notes:
- The hcm upgrade command will fail if the target is specified as
  sync://<host>:<port>/Projects/. The Projects directory cannot be
  converted to an HCM module. Subdirectories within the Projects
  directory can be converted to HCM modules, for example,
  sync://chip.ABCo.com:2647/Projects/Chip.
- The hcm upgrade command can be run multiple times without harming
  your data. For example, if someone creates REFERENCE statements in a
  sync_project.txt file rather than with the hcm addhref command, you
  can run the hcm upgrade command to convert the REFERENCE statements
  into HCM hierarchical references.
- The hcm upgrade command does not require a sync_project.txt file in
  the vault directory specified by the vault path. If this file does
  not exist, the hcm upgrade command creates one.
- After searching for and processing sync_project.txt files within the
  vault directory structure below the specified vault path, the hcm
  upgrade command cleans up the vault structure by deleting each
  sync_project.txt file found and deleting the directory which
  contained it. Before deleting each sync_project.txt file, a copy of
  it is placed in the vault folder located one level above the folder
  that was deleted. The copied file is named
  .SYNC.<deletedDirectoryName>.sync_project.txt.
- If the hcm upgrade command detects an unexported module, it displays
  an error. Before rerunning the hcm upgrade command, you need to
  move the referenced directory to the /Projects level and modify
  the REFERENCE statement in the identified sync_project.txt file.
- After the hcm upgrade command has been used to convert REFERENCE
  statements to hrefs, it is highly recommended that users fetch
  each new configuration to an empty work area rather than try to
  update their existing work areas.

After converting a DesignSync vault path into an HCM module, you can
use the hcm showconfs and hcm showhrefs commands to view the
configurations and hierarchical references created by the hcm upgrade
command.

The ability to upgrade vault paths can be access-controlled. By
default, any user can upgrade vault paths. If this policy is not
acceptable for your project, you (as the administrator or project
leader) can restrict or deny this capability for your site or
individual SyncServers. For more information, see the Synchronicity
Access Control Guide. See also information on the access allow and
access deny commands.
```

SYNOPSIS

```
hcm upgrade -target <DesignSyncVault_url>
```

OPTIONS

- -target

-target

```
-target                 Specifies the URL of the DesignSync vault you
 <DesignSyncVault_url> want to upgrade to an HCM module.

                        Specify the URL as follows:
                         sync[s]://<host>[:<port>]/<vaultPath>
                        where <host> is the SyncServer on which the
                        module resides, <port> is the SyncServer port
                        number, and <vaultPath> identifies DesignSync
                        vault you want to convert to an HCM module.
```

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
access allow, access deny, mvfolder
,
```

EXAMPLES

```
This example guides you through the process of upgrading your
Projects/Chip vault path to an HCM module. This example assumes the
following hierarchy:

sync://chip.ABCo.com:2647/Projects/Chip
   IP subdirectory
     ALU subdirectory <= contains a sync_project.txt file with
                         REFERENCE
                                 sync://alu.ABCo.com:2647/Projects/ALU
                         CONFIG Trunk Golden
                         CONFIG Silver V1.1

dss> hcm upgrade -target sync://chip.ABCo.com:2647/Projects/Chip

When you run this command, HCM does the following:
1. Converts the Chip directory into a module by running the hcm mkmod
```

command:
```
dss> hcm mkmod -target sync://chip.ABCo.com:2647/Projects/Chip
```

2. Recurses through all the subdirectories contained in the /Projects/Chip directory looking for sync_project.txt files.

3. When it finds the /Projects/Chip/IP/ALU/sync_project.txt file, it first determines if there is a CONFIG statement that maps the Trunk configuration (or Trunk:Latest) to any selector. If so (as in the example), then the remainder of step 3 is skipped.

   If it doesn't find a configuration mapping for Trunk, then the command creates a hierarchical reference between Chip and ALU by invoking the equivalent of the hcm addhref command and using the vault path value of the REFERENCE statement:

```
dss> hcm addhref \
         -fromtarget sync://chip.ABCo.com:2647/Projects/Chip \
         -totarget sync://alu.ABCo.com:2647/Projects/ALU \
         -relpath IP/ALU
```

   Notes:
   - HCM calculates the -relpath value from the location of the sync_project.txt file, not from the REFERENCE target in that file. For this example, the sync_project.txt file resides in 'Projects/Chip/IP/ALU'; HCM calculates the -relpath value as 'IP/ALU'.
   - The hierarchical reference created is between the default configuration of Chip to the default configuration of ALU.
   - In the example, the DesignSync behavior of the original sync_project.txt file is to populate the Golden configuration of ALU when the Trunk configuration of Chip is populated recursively. For HCM to perform the equivalent of DesignSync's recursive populate operation, a hierarchical reference should exist from the default configuration of Chip and connect to the Golden configuration of ALU. (Remember that the default configuration is implicitly defined by HCM to be the selector Trunk:Latest). Creating this hierarchical reference occurs as shown in step 4. The creation of this hierarchical reference thus removes the need for hcm upgrade to create an href that connects the default configuration of Chip to the default configuration of ALU. (In other words, this connection is the typical step taken by hcm upgrade when processing the REFERENCE statement in the sync_project.txt file).

4. Converts the CONFIG keywords to hierarchical references by running the equivalence to the hcm addhref command for each keyword:

```
dss> hcm addhref \
           -fromtarget sync://chip.ABCo.com:2647/Projects/Chip \
           -totarget sync://alu.ABCo.com:2647/Projects/ALU@Golden \
           -relpath IP/ALU
dss> hcm addhref \
         -fromtarget sync://chip.ABCo.com:2647/Projects/Chip@Silver \
         -totarget sync://alu.ABCo.com:2647/Projects/ALU@V1.1 \
         -relpath IP/ALU
```

```
     Notes:
     - A CONFIG map involving Trunk is created as an HCM hierarchical
       reference from the default configuration of Chip instead of from
       the Chip@Trunk configuration.
     - The second hcm addhref command shown implicitly creates the
       Silver configuration of the Chip module.

  5. Saves a copy of the sync_project.txt file into
     /Projects/Chip/IP/.SYNC.ALU.sync_project.txt. The ALU subdirectory
     that contained the sync_project.txt file is removed.
```

Index

A

Alias

removing 1099

B

Branch

creating 467

C

Check In

objects 241

Checkout

canceling 230

Command

interrupting 12

logging 50

managing command alias 47

output 54, 62

Comparing Objects

comparing files 748, 792

compressed archive

upload 592

D

Date Format 16

Date Selectors 202

DesignSync Client

DesSync 31

dss 33

dssc 35

stcl 38

stclc 42

Development Area

changing 66

creating 75

GUI 70

joining 71

listing 73

removing 81

Directories

changing current directory 605

displaying path 605

E

Enterprise Objects

enterprise product revisions 912

F

Fetch State

default 8

File

naming 28

Filter

persistent filters 197, 548

Folders

creating 479

deleting 521

G

Get

fetching a legacy module 1041

showing contents of a module view 86

H

HCM

checking in configurations 1073

creating a hierarchical reference 1024

creating a legacy configuration 1065

creating a release alias 1062

creating a release configuration 1079, 1089

creating an HCM module 1070

displaying configuration status 1136

displaying legacy module configurations 1113

displaying modules on the server 393, 1133

displaying the module configuration hierarchy 710, 1120

fetching a legacy module configuration 1041

module cache releases 1129

removing a configuration 1101

removing a hierarchical reference 1104

removing a module 1109

removing a release alias 1099

upgrading vault to legacy module 1151

Help

getting help 804

viewing 953

Hierarchical References

adding whereused support 654

creating 659, 1024

displaying 696, 710, 861, 1120

removing 691, 1104

tracing 628, 718

updating 676

I

ip

upload 592

L

Legacy Module

creating a configuration 1065

creating an alias 1062

displaying 1113

removing a hierarchical reference 1104

upgrading 575

Locking

changing lock owner 555

releasing from a server 560

M

Merge Edge

creating 475

removing 516

Module

adding content to 222

branching 467

changing 453, 559, 744

comparing 748

creating 283

deleting 525

displaying status 383, 396, 881, 891

exporting 450

filtering 197, 548

hrefs 504, 688

importing 459

information 773, 791, 811

locking 461, 560, 914

members

moving 481

removing 506, 518

tagging 417

moving on the server 214, 489

purging 491

rolling back 541

selector 202

swapping 606, 609, 617, 624

tagging 417, 464

unremove 569

Module Cache

listing modules 875

listing releases 1129

Module Hierarchy

creating 659

displaying 696, 861

removing 691

updating 504, 676, 688

whereused 628, 637, 642, 649, 654, 718, 727, 732, 739

Module Views

setting 219

O

Objects

deleting 518

listing information 811

P

Populate

objects 106, 292

R

Revision Control

keywords

using 13

S

Selectors

how to use 16

persistent 202

Server

removing a legacy configuration 1101

SITaR overview

branching 979

comparing local workspace to a configuration 1007

creating a new module release and updates baseline alias 996

creating and releasing the initial version of modules 985

displaying a list of modules matching criteria 971

identifying module configurations to process 1000

integrating changes into the project integration workspace 966

SITaR environment variables 961

submitting a module version for integration 1011

updating a submodule workspace with the specified module version 1017

updating the workspace with the specified module version 990

viewing the context for a module 958

Swap

edit-in-place 606

replacing module version 609

restoring swapped module versions 617

showing swapped modules 624

SyncRef 1

T

Tag

branches 417

versions 417

tar

upload 592

Troubleshooting

returning environment information 919

U

upload

command 592

V

Vaults

deleting version 535

locked items 914

purging 491

upgrading 575

vault association 214

Versions

history

displaying 929

View

associating with a work area 219

listing views 89

removing from the server 95

uploading to the server 92

verifying view definitions 85

W

Where Used

supporting hierarchical references 654

tracing use of hierarchical references 628, 642, 649, 718, 732, 739

where mofule versions are used 637, 727

Work Area

associating with a view 219

Workspace

roots

setting 211