



ENOVIA DesignSync

HCM User's Guide

3DEXPERIENCE 2022

©2021 Dassault Systèmes. All rights reserved. 3DEXPERIENCE®, the Compass icon, the 3DS logo, CATIA, SOLIDWORKS, ENOVIA, DELMIA, SIMULIA, GEOVIA, EXALEAD, 3D VIA, BIOVIA, NETVIBES, IFWE and 3DEXCITE are commercial trademarks or registered trademarks of Dassault Systèmes, a French "société européenne" (Versailles Commercial Register # B 322 306 440), or its subsidiaries in the United States and/or other countries. All other trademarks are owned by their respective owners. Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.

**DASSAULT
SYSTEMES**

Table of Contents

Overview	1
ENOVIA Synchronicity DesignSync Data Manager with HCM Capability	1
Using ENOVIA Synchronicity DesignSync Data manager with HCM User's Guide Documentation	1
Before Reading this Guide.....	1
Related Topics.....	2
Introducing Modules	3
Overview of Modern and Legacy Module Command Sets	5
HCM Concepts	7
Module	7
Configuration	9
Hierarchical References	11
Base Directory	12
Alias.....	14
Guidelines for HCM Use	14
Using DesignSync with HCM.....	15
Using ProjectSync with HCM.....	17
Naming Guidelines	21
The Relationship Between HCM Modules and ProjectSync Projects	22
How the Release Operation Works.....	23
How HCM Operations Handle an Alias.....	25
Module Cache	26
A Comparison of Module Caches and DesignSync Caches.....	28

Using HCM from DesignSync..... 31

 Commands in the DesignSync GUI client..... 31

 Viewing Modules and Configurations..... 31

 Viewing a Module Hierarchy 34

 Getting a Module 35

 Showing the Status of a Configuration..... 40

 Moving an Alias to Another Release..... 42

 Showing the Contents of Module Caches..... 44

 Deleting HCM Objects 45

 Removing a Hierarchical Reference from a Module Configuration 48

Scenarios for Using HCM..... 51

 Overview..... 51

 The Chip Architect Sets Up the Project Structure..... 52

 The PLL Team Creates a Module and Its Contents..... 54

 The ALU Team Creates a Module from a Vault..... 56

 An ALU Designer Gets Files of a Module 57

 A Designer Puts Files of a Module Back on the Server 60

 A Designer Adds Files to a Configuration 62

 A Designer Removes Files from a Configuration 66

 The CPU Team References the ALU Module 69

 The ALU Team Makes a Release Available 70

 The CPU Team Subscribes to Email on a Hierarchy 73

 The CPU Team Leader Queries for Defects..... 77

The CPU Team References an IP Gear Deliverable	79
The CPU Team Changes a Reference to a New ALU Release	80
A CPU Designer Gets a Module Hierarchy	83
A Designer Puts a Module Hierarchy Back on the Server	85
The CPU Team Creates a Release	89
The ALU Team Creates an Alias for a Release	93
Designers Use the Module Cache	97
The MPU Team Upgrades to HCM	103
A Designer Creates a Configuration for Experimentation	106
The MPU Team References the ALU Work in Progress	108
The CPU Team Creates a Work-in-Progress Configuration from a Release	109
The CPU Team Develops a Checking Script Using HCM	110
The CPU Team References a Tools Module	112
The MPU Team Removes a Module	114
A Designer Removes a Configuration	117
The ALU Team Removes an Alias	119
Quick Steps	123
Quick Steps	123
HCM Administration	133
Overview of Administration Tasks	133
Enabling Legacy Module Support	135
Legacy Module Triggers and Email Notification	135
Mapping Note Types	137

Table of Contents

Storing Logins.....	141
Setting Up Email Notification of HCM RevisionControl Notes.....	145
Access Controls.....	146
Module Caches.....	146
Command Reference	155
Accessing the Command Reference	155
Getting Assistance	157
Using Help.....	157
Getting a Printable Version of Help.....	158
Contacting ENOVIA.....	159
Index	161

Overview

HCM is software that supports development in a hierarchical fashion. With HCM, design teams manipulate module configurations, which can correspond to blocks of a design. Each module can represent any level of the design hierarchy, from leaf cell through DSP core to the whole IC. HCM lets designers define hierarchical relationships between blocks (module configurations) and provides interfaces to work on the entire hierarchy of blocks as one entity (a module hierarchy).

ENOVIA Synchronicity DesignSync Data Manager with HCM Capability

HCM enhances the ENOVIA Synchronicity command set with additional capabilities to support legacy modules, including:

- Viewing Modules and Configurations
- Viewing a Module Hierarchy
- Getting a Module
- Showing the Status of a Configuration
- Moving an Alias to Another Release
- Showing the Contents of Module Caches
- Deleting HCM Objects
- Removing Hierarchical References from a Module Configuration

Using ENOVIA Synchronicity DesignSync Data manager with HCM User's Guide Documentation

This guide is single-sourced in HTML and available from the Documentation Index page.

- Windows - available from the **Dassault Systems** product group in the Windows **Start** menu.
- UNIX, - available by pointing your web browser to `$SYNC_DIR/share/content/doc/index.html`

Note: References from the *ENOVIA Synchronicity DesignSync Data Manager HCM User's Guide* to the *ENOVIA Synchronicity Command Reference* guide always link to the ALL version of the guide, which contain information about all working methodologies for DesignSync. For more information about the available working methodologies, see ENOVIA Synchronicity Command Reference.

Before Reading this Guide

You might need to refer to the following guides if you are learning to use the ENOVIA Synchronicity DesignSync Data Manager for Visual Studio product.

<i>ENOVIA Synchronicity Command Reference</i>	Full documentation for all HCM commands.
<i>ENOVIA Synchronicity DesignSync Administrator's Guide</i>	Describes the customizations available to optimize performance and usability and allows you to enable HCM.

Related Topics

Introducing Modules

Overview of Modern and Legacy Module Command Sets

Introducing Modules

The DesignSync Data Manager features two types of modules:

- Modern modules.
- Hierarchical Configuration Manager (HCM) modules, also called legacy modules.

This documentation focuses exclusively on legacy (HCM) modules.

HCM is software that supports development in a hierarchical fashion. With HCM, design teams manipulate module configurations, which can correspond to blocks of a design. Each module can represent any level of the design hierarchy, from leaf cell through DSP core to the whole IC. HCM lets designers define hierarchical relationships between blocks (module configurations) and provides interfaces to work on the entire hierarchy of blocks as one entity (a module hierarchy).

HCM supports such activities such as tracking the version history of blocks and the chips that use the blocks, releasing blocks that have met validation criteria, and packaging blocks for re-use.

To use HCM software, you must:

1. Have an HCM license.
2. Enable HCM in DesignSync.
3. Run HCM commands from a command line, for example in a DesignSync command shell (stcl, stclc, dss, dssc):

```
% stclc
stcl> hcm showmods -target sync://alu.ABCo.com:2647
```

Note: Modern modules documentation is fully integrated into the other documentation in DesignSync as appropriate, for example, Access Controls for modules are documented in the Access Control User's Guide; System Administrator tasks for modules are documented in the DesignSync Data Manager Administrator's Guide, etc.

Related Topics

Release Information

Overview of Modern and Legacy Module Command Sets

This table shows the command functionality available to users of V6 (modern) and legacy modules.

Important: The command syntax is different between V6 and legacy module commands, unless otherwise noted. For full command syntax, see the ENOVIA Synchronicity Command Reference, or type `<command> -usage` in your DesignSync client.

Command	V6 Modules command	Legacy Modules command	Description
Add	add	X	Adds new objects to a module.
Addbackref	addbackref	X	Adds whereused support to targets of hierarchical references.
Addhref	addhref	hcm addhref	Creates a hierarchical reference between modules.
Addlogin	addlogin	hcm addlogin	Stores a server login to enable hierarchical queries to legacy modules across servers. Note: Command syntax is identical.
Doc	X	X	This command is now obsolete.
Get	populate	hcm get	Fetches a legacy module configuration into your work area. Note: Command syntax is identical except legacy module mode enabled the <code>edit</code> (and <code>edit recursive</code>) and <code>merge</code> options.
Lock	lock	X	Creates a server-side lock on a module branch.
Migratetag	migratetag	X	Migrates DesignSync and legacy module tags to modules.
Mkalias	X	hcm mkalias	Creates an alias for a release of a legacy module.
Mkconf	X	hcm mkconf	Creates a legacy configuration for a module on a server.
Mkedge	mkedge	X	Creates a merge edge between specified versions.
Mkmod	mkmod	hcm mkmod	Creates a module on the server.
MvMember	mvmember	X	Changes the natural path of a module member.
Put	X	hcm_put	Checks in a legacy module configuration.
Release	X	hcm_release	Creates a configuration that cannot be modified.

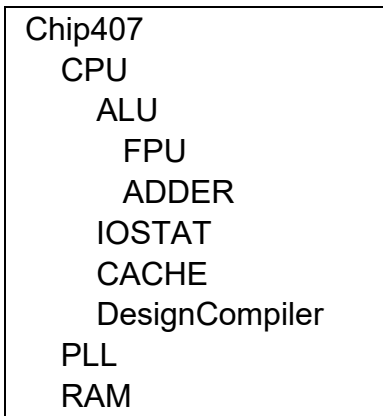
Overview of Modern and Legacy Module Command Sets

Remove	remove	X	Removes a module member from a module
Rmalias	X	hcm rmalias	Removes an alias for a release.
Rmedge	rmedge	X	Removes previously set merge edges.
Rmconf	X	hcm rmconf	Removes a legacy configuration from the server.
Rmhref	rmhref	hcm rmhref	Removes a hierarchical reference from a modern or legacy (HCM) module to a modern or legacy module, external module (modern modules only), DesignSync vault, or IP Gear deliverable.
Rmlogin	rmlogin	hcm rmlogin	Removes a stored server login used when following hierarchical queries from legacy modules across servers. Note: Command syntax is identical.
Rmmod	rmmod	hcm rmmod	Removes a module from a server or workspace.
Rollback	rollback	X	Reverts a module back to a previous version, rolling back any structural and content changes.
Showconfs	showconfs	hcm showconfs	Displays legacy module configurations. Note: Command syntax is identical.
Showlogins	showlogins	showlogins	Displays the logins stored on a server.
Showmcache	showmcache	hcm showmcache	Lists the modules in one or more caches.
Showmods	showmods	hcm showmods	Displays the modules available on a server or workspace.
Showstatus	showstatus	hcm showstatus	Displays the status of a module in your workspace.
Unremove	unremove	X	Restores a previously removed module member into the module workspace.
Upgrade	upgrade	hcm upgrade	Upgrades a legacy module or a DesignSync vault to a module or legacy module.
Version	version	hcm version	Displays the DesignSync installation version. Note: Command syntax is identical.

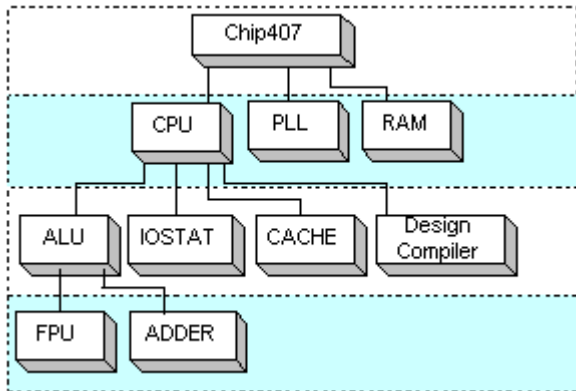
HCM Concepts

Module

A module is data that represents a level of the design hierarchy. For example, in the following design hierarchy, any one of the levels can be a module.

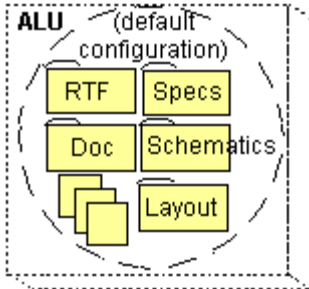


In the following example design hierarchy, Chip407 is the main module; CPU, PLL, and RAM are submodules of Chip. The CPU module has submodules: ALU, IOSTAT, CACHE, and DesignCompiler. The ALU submodule has submodules of its own: FPU and ADDER.

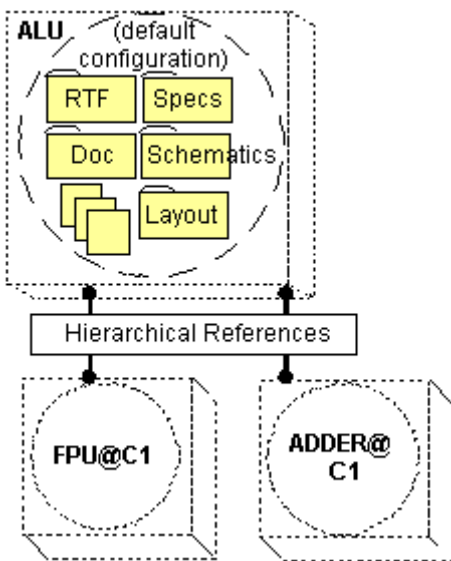


A module's data includes:

- Files managed in DesignSync or an entire vault directory of files.



- (Optionally) Hierarchical references to other modules (submodules). The contents of the submodules, however are not included. It is through hierarchical references that a module hierarchy is created



Module Hierarchy

HCM lets designers define hierarchical relationships between blocks (represented in modules) and provides simple yet powerful interfaces to work on the entire hierarchy of such modules as one entity.

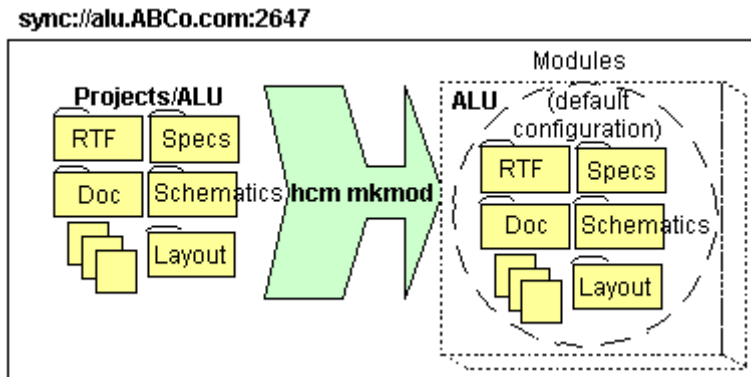
There are two general phases of the creation of a hierarchy of modules:

- Creation of the modules
- Creation of the connections between modules (hierarchical references)

Module Creation

HCM modules are created through the hcm mkmod operation or the **Create a module** dialog. See The ALU Team Creates a Module from a Vault for an example scenario of this operation.

Note: Modules can be removed from the server with the **hcm rmmod** command or the **Delete** dialog. For an example scenario, see *The MPU Team Removes a Module*.



Hierarchical Reference Creation

A hierarchical reference is a connection between two module configurations that specifies each module's place in a hierarchy. Because a hierarchical reference identifies an upper-level module and a submodule, adding a hierarchical reference to a module configuration in effect creates submodules.

Modules and Configurations

In HCM, a module is always associated with a configuration, either a default configuration created by the **hcm mkmod** command (or the **Create a module** dialog) or a configuration you create for a module with the **hcm mkconf** command (or the **Create a configuration for the module** dialog).

Related Topics

[hcm addhref Command](#)

[hcm mkmod Command](#)

[hcm rmmod Command](#)

[hcm mkconf Command](#)

[hcm rmconf Command](#)

Configuration

A configuration is a group of managed files, each file at a particular version.

There are four types of configurations, depending on how the configuration was created:

- **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** operation. (**Note:** Although the contents of a release cannot be changed, the release can be removed from the server with the **hcm rmod** command or the **Delete** dialog. This operation removes a module's configuration definitions as well as the module itself.)
- **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** operation with the **-branch** option. This type of configuration can be removed from the server by using the **hcm rmod** operation.

- **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** operation. This type of configuration can be removed from the server by using the **hcm rmod** operation.

Note: In HCM operations, when you specify a module without a configuration name, the HCM operation uses the module's default configuration.

- **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** operation with the **-selector** option. This type of configuration can be removed from the server by using the **hcm rmod** operation.

Note: Because an alias is a symbolic name for a release, it can be thought of as a type of configuration. In addition, HCM operations such as **hcm showconfs** list aliases as well as other configuration types.

To display a list of configurations of a module, use the **hcm showconfs** operation. For example:

```
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU
```

Configurations of module sync://alu.ABCo.com:2647/Projects/ALU:

NAME	TYPE	OWNER	SELECTOR/ALIASED	RELEASE
<Default>	Branch	Anne	Trunk:Latest	
R1	Release	Anne		
R2	Release	Anne		
GOLDEN	Alias	Anne	R2	

Related Topics

Module

hcm mkconf Command

Working with Configurations

hcm rmconf Command

hcm mkmod Command

hcm rmmod Command

hcm release Command

hcm showconfs Command

Hierarchical References

A **hierarchical reference** is a connection that defines a hierarchical relationship between two module configurations.

A hierarchical reference connects an upper-level module configuration to any of the following:

- Another module configuration (making that module a submodule of the upper-level module)
- An IP Gear deliverable
- A DesignSync vault location

Creating a hierarchical reference lets you define a hierarchy of modules and submodules. Such a reference is created with the **hcm addhref** command or the **Create a hierarchical reference for the configuration** dialog. In this operation, you specify:

- An upper level module URL (the **-fromtarget** option)
- A submodule URL (the **-totarget** option)

- The relative path from the upper-level module's base directory to the submodule's base directory (the **-relpath** option). The **hcm get** operation uses this value when users get a module hierarchy to their work areas. For more information on how the get operation uses this value, see Base Directory.

Related Topics

hcm addhref Command

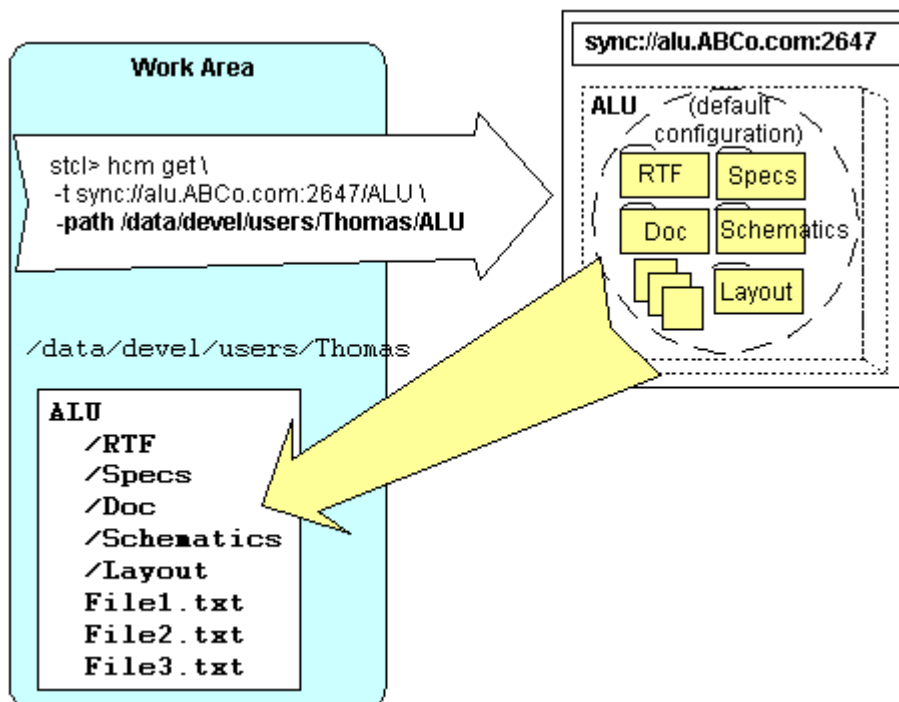
Working with a Module Hierarchy

hcm get Command

Base Directory

The **base directory** of a module is the file system directory into which a module is fetched when users get the module to their work areas.

You specify the location of the base directory with the **hcm get -path** operation when you get the module to your work area. (If you do not specify the **-path** option, the operation uses your current work area directory.) For example:



To determine where to place a module hierarchy in your work area, the **hcm get** operation:

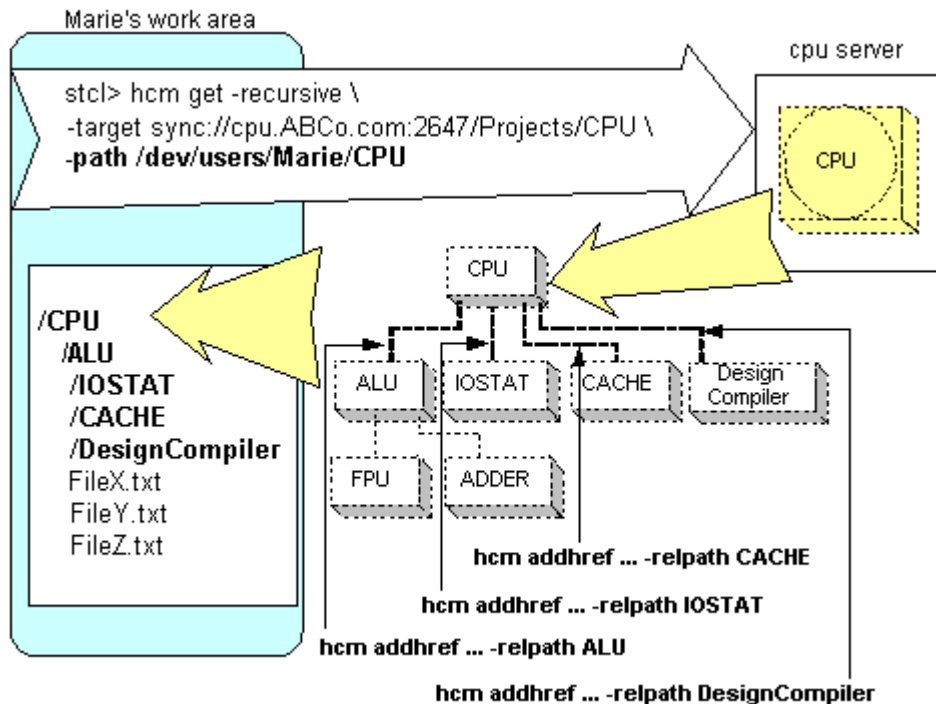
- Uses the base directory you specify with the **-path** option.

- Adds to that the relative path (specified with the **-relpath** option when a hierarchical reference was added with the **hcm addhref** operation). (HCM stores this information as part of the configuration's data.)

The relative path is always in relation to the upper-level module's base directory. When you fetch the module hierarchy with the **hcm get -recursive** operation, HCM software places the submodules' directories and files in your work area in relation to the upper-level module's base directory (specified with **hcm get -path**).

The **hcm get -recursive** operation follows hierarchical references to the configuration's submodules (if any) and fetches their files. Files are fetched into the directory locations indicated by the relative path of each hierarchical reference. Thus a referenced submodule is placed in the path `<Path>/< RelativePath>`.

For example, the CPU team leader adds a hierarchical reference from the CPU module to the ALU module, specifying a **-relpath** of `ALU`. Later, a designer on the CPU team then gets the CPU module hierarchy to her work area (with **hcm get -recursive**), using the **-path** option to specify CPU be put in her current work area directory, `/dev/users/Marie/CPU`.



Related Topics

hcm get Command

Alias

An **alias** is a symbolic name for a release. For example, the ALU design team might create ALU@GOLDEN as the symbolic name for its ALU@R1 release.

You can use an alias just as you would other configuration types in a Synchronicity URL passed to HCM commands or features. For example, when you create hierarchical references to submodules, you can specify an alias as the submodule (the **-totarget** value).

In addition, you can change an alias to point to a different release. For example, the ALU team might initially use the GOLDEN alias to point to the ALU@R1 release. Later, when it creates another release of the ALU design (ALU@R2), the team might change the ALU@GOLDEN alias to point to the new release. By referencing the ALU@GOLDEN alias, other design teams can get the appropriate release of the ALU module.

In operating on an alias, HCM commands and features like hierarchical subscription resolve the alias to the release to which it points and then operate on the release. For more information, see [How HCM Operations Handle an Alias](#).

To create an alias or to change it to point to a different release, use the **hcm mkalias** operation or the **Create an Alias for the release** dialog. For an example scenario, see [The ALU team Creates an Alias for a Release](#).

To remove an alias from the server, use the **hcm rmalias** operation or the **Delete** dialog. For an example scenario, see [The ALU Team Removes an Alias](#).

Related Topics

[Creating or Changing an Alias](#)

[Deleting HCM Objects](#)

[hcm addhref Command](#)

[hcm mkalias Command](#)

[hcm rmalias Command](#)

Guidelines for HCM Use

When using HCM, follow these rules:

- When you use **hcm mkconf -selector** or **hcm mkconf -branch** to create a hierarchical configuration, do not use `--R` in the **-selector** or **-branch** option value. (See Naming Guidelines for more information.)

If the selector or branch value contains `--R`, HCM considers the configuration a release or an alias. However, using the **hcm mkconf** operation to create a release or an alias is not a good practice. Instead, use the **hcm release** and **hcm mkalias** operations.

- When you specify a URL in HCM commands, 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 the **hcm addhref** operation, if the upper-level module and submodule reside 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.)

For example, if you specify a **-fromtarget** that uses a fully qualified host name, you should specify the same fully qualified host name for the **-totarget**. If you do not specify the same domain name for both options, the addhref operation succeeds, but HCM does not utilize a performance enhancement for references on the same server.

- In HCM operations that specify a target (with **-target**, **-fromtarget**, and **-totarget** options), specify a domain name that will work for all teams that want to access and reuse your modules.

Related Topics:

[Using DesignSync with HCM](#)

[Using ProjectSync with HCM](#)

[Naming Guidelines](#)

Using DesignSync with HCM

When HCM is installed on your SyncServer, certain DesignSync behaviors change, and in some cases you must use DesignSync differently if you want to use HCM. The following is a list of changes:

- You cannot modify both legacy and modern modules in the same workspace. If you have a hierarchical reference tree containing both, you should not modify the sub-modules in the workspace containing the full hierarchy. You should create a new workspace for the editing the sub-module.

- When you work with HCM modules, do not use the DesignSync **mkbranch -recursive** command to branch an entire directory tree. (You can, however, use **mkbranch** without the **-recursive** option. You can also use autobranching.)

Using the **mkbranch -recursive** operation results in the branching of files when submodules are located within the directory structure of the upper level module.

To branch an entire directory tree, follow these steps:

Note: This procedure branches only the upper-level module of a module hierarchy.

1. Use the **hcm release** command or the **Release a module** dialog to create a release of the configuration you want to branch.
2. Use the DesignSync **mkbranch** command to create the branch of the module configuration. For the argument, specify the vault URL of the upper-level module. For the **-version** option, specify the release's name.
 - **Note:** You can use the **mkbranch -recursive** operation in a work area to which no hierarchical references (submodules) have been fetched. For an example scenario, see *The CPU Team Creates a Work-in-Progress Configuration from a Release*.
 - Using previous versions of DesignSync, some design teams used vault REFERENCES to import modules from other design projects into their designs. Such vault REFERENCES performed a role similar to the one that hierarchical references now perform in HCM.

When you are using HCM, you no longer need to use REFERENCE statements to incorporate other design teams' work into your design. Instead, use hierarchical references to create a hierarchy of HCM modules. In addition, you can use the **hcm upgrade** operation to change vault REFERENCES to HCM hierarchical references. For an example scenario, see *The MPU Team Upgrades to HCM*.

Note: Although you do not need to use REFERENCES, you can still create projects and provide vault paths as REFERENCE statements. HCM software recognizes and follows REFERENCE statements.

- The **hcm get** operation supports the **-edit** option only on an upper-level module of a module hierarchy. For example, suppose a module hierarchy has CPU as the upper-level module. CPU contains files and a submodule called ALU, which has a submodule, FPU. If you perform an **hcm get -recursive -edit** operation on CPU, the operation locks only the files of the CPU module but not files of ALU or its FPU submodule.
- In using HCM, you should be aware that, for the most part, HCM operations follow DesignSync settings. For example, the **hcm put** operation follows the **Minimum checkin comment length** setting. If you must provide a checkin

comment of a certain length when you check files in with the DesignSync checkin (**ci**) operation, you will have to do the same when you use the **hcm put** operation. Other examples of settings that HCM operations follow are: the `RmVaultKeepVid` registry setting, (affects the behavior of the **hcm rmod -vaultdata** operation), the `PopulateNoEmptyDirs` registry setting (the **hcm get** operation follows this setting if it is specified; if the setting is not specified, the get operation removes empty directories), and the Cadence Recurse View Folders setting (**hcm get** and **hcm put** operations follow this setting). Consult your project leader or Synchronicity administrator about the effect of these settings on HCM operations.

However, HCM operations do not follow certain DesignSync settings:

- - The **hcm get** and **hcm put** operations do not support the default fetch state of **Always point to the latest version (Mirror)**. If this setting has been specified in DesignSync, your project leader or Synchronicity administrator will have to change it in order for you to use the get and put operations. For information on the default fetch state, see *Defining a Default Fetch State in DesignSync Help*.

If you want to maintain another workspace with links to the latest versions, you can use the DesignSync **populate -mirror** operation. You must explicitly specify the **-mirror** option. If you omit **-mirror** and your system administrator has set the default fetch state to **Always point to the latest version (Mirror)**, HCM operations will fail.

- The **hcm put** operation does not follow settings that specify dereferencing of symbolic links to files or directories. Consult your project leader or Synchronicity administrator for information about this setting's effect on HCM operations.
- For more information about the interaction of DesignSync settings and HCM operations, see *Overview of Administration Tasks*.
- As Synchronicity administrator or project leader, you do not have to enable RevisionControl notes for HCM operations. HCM installation enables RevisionControl notes for HCM commands, and notes are automatically generated in response to HCM commands.
- DesignSync commands treat module cache links as unmanaged directory links. When using the DesignSync **ci** command, you should not use **ci -new** to check in directory links.

Using ProjectSync with HCM

When HCM is installed on your SyncServer, certain ProjectSync forms have additional fields, and in some cases you use ProjectSync differently for operations on HCM objects (for example, a query on a module hierarchy). The following is a list of changes:

- With previous versions of DesignSync and ProjectSync, team members used vault REFERENCES to import modules from other design projects into their own designs.

With HCM software, REFERENCE statements are not needed; HCM uses hierarchical references (created with the **hcm addhref** command or the **Create a hierarchical reference for the configuration** dialog) to incorporate design work from other sources into a module hierarchy. For an example scenario, see The CPU Team References the ALU Module.

- Certain HCM operations create or delete ProjectSync projects or configurations.

In certain cases, the **hcm upgrade** and the **hcm mkmod** operations create ProjectSync projects in addition to HCM modules. (For information, see The Relationship Between HCM Modules and ProjectSync Projects.) Configurations created by the **hcm mkconf**, **hcm release**, and **hcm mkalias** operations (when the target resides below the `Projects` vault folder) appear in ProjectSync as Configurations.

The **hcm rmmod**, **hcm rmconf**, and **hcm rmalias** operations remove ProjectSync projects or configurations. These operations also detach and, optionally, delete ProjectSync notes.

- The **hcm rmmod** operation removes the ProjectSync project and/or configurations associated with a module only when the **-vaultdata** option is specified. (If the **-vaultdata** option is not specified, ProjectSync still displays the module and its configurations as a project.) The **rmmod** operation also detaches ProjectSync notes attached to the module and its configurations. If the **-notes** option is specified, the operation deletes those notes that were detached and not attached to other objects.
- The **hcm rmconf** and **hcm rmalias** operations remove the ProjectSync configuration created with the **hcm mkconf** or **mkalias** operation. The **rmconf** and **rmalias** operations also detach any ProjectSync notes attached to the configuration. If the **-notes** option is specified, these operations delete those notes that were detached and not attached to other objects.
- The standard RevisionControl note type has the field **Tag or HCM sub-cmd**. For HCM operations, this field displays the name of the HCM command, minus its **hcm** prefix. For information on how to configure email notifications to display this field, see Setting Up Email Notification of HCM RevisionControl Notes.

Email Subscriptions

When you use ProjectSync to subscribe for notes attached to any part of a module hierarchy, ProjectSync's behavior depends on the extent of the subscription, as specified in the **Scope** field.

- To subscribe to email for notes on an HCM module or module hierarchy (perform a hierarchical subscription), use the ProjectSync **Advanced Subscriptions** panel. (The standard **Add New Subscriptions** panel does not support hierarchical subscriptions.) For an example scenario, see *The CPU Team Subscribes to Email on a Hierarchy*.
- In the ProjectSync **Advanced Subscriptions** panel, to have an email subscription apply only to a single object, specify the object's URL in the **Object Filter** field and select **This object only** from **Scope** pulldown menu. This action is equivalent to using the trailing slash in the **Object Filter** field in ProjectSync when HCM is not installed.
- When you subscribe to email on a module configuration that is an alias, ProjectSync resolves the alias to the release to which it points and adds the subscription to that release. In the Subscription status panel, ProjectSync displays the alias associated with the release.
- HCM does not automatically update email subscriptions when an alias changes to point to a different release. You must manually update your email subscription to subscribe to notes for the new release. For an example scenario, see *Robert Updates Email Subscriptions*.

Query

To query for notes on an HCM module hierarchy (a hierarchical query), you first select **Standard Query** from the ProjectSync menu. (The **Advanced** and **Custom Query** forms do not support hierarchical references.) Then you use the ProjectSync **Query** form's **Project**, **Configuration**, and **Scope** fields to specify the module configuration to query and the extent of that query.

Note: All HCM modules located in the `/Projects` vault folder in DesignSync appear as Projects in ProjectSync, and all of their configurations appear as ProjectSync Configurations.

The selections of the **Scope** pulldown menu are:

- This object only
- This object and one level below
- This object and all levels below

The **Project**, **Configuration**, and **Scope** fields affect a query on a module configuration in the following ways:

- If you query for notes on a module and you leave the configuration field blank, you are performing a query on the module's default configuration.
- If you query for notes on a module but not its submodules (Scope=This object only), then a default configuration query displays both:
 - Notes that are attached to the module with no configuration (for example, Project=CPU Configuration=<blank>)

- Notes that are attached to the module at a specific configuration (for example, Project=CPU Configuration=C21)
- If you query for notes on a module and some or all of its submodules (Scope=This object and one level below or Scope=This object and all levels below), then a default configuration query behaves differently from one with Scope=This object only. The query mechanism descends into the module hierarchy according to the value in the Scope field, but at each level in the hierarchy it displays only notes that are attached to the module at a specific configuration (for example, Project=CPU Configuration=C21). For an example scenario of this type of query, see The CPU Team Leader Queries for Defects.
- A query for notes on a configuration that is an alias displays the same results as a query on the release to which the alias points. This behavior is due to the NoteAttach functionality, which synchronizes notes on aliases and releases.

The NoteAttach Functionality

When a note is attached to an object:

If the object is...	The note is attached to...
An alias	The alias and the release configuration to which the alias points
A release	The release and any aliases that point to the release.

Notes:

- This behavior also applies to any notes you create and attach to objects.
- Upon installation, the HCM software enables the generation of RevisionControl notes for HCM operations and attaches the notes to objects.

When the alias is changed to point to a different release, ProjectSync:

- Detaches from the alias any notes associated with the previous release
- Attaches any notes associated with the release to which the alias now points

Note: For the NoteAttach functionality to operate in this way, you must add the hcmNoteAttach trigger. For information, see Adding the hcmNoteAttach Trigger.

Related Topics

hcm addhref Command

hcm mkalias Command

hcm mkconf Command

hcm mkmod Command

hcm release Command

hcm rmalias Command

hcm rmconf Command

hcm rmmod Command

hcm upgrade Command

Naming Guidelines

When you create a configuration, alias, or release, follow these guidelines for naming it.

- Configuration, release, and alias names:
 - Can contain letters, numbers, underscores (`_`), periods (`.`), and hyphens (`-`). 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).
 - 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.
- Configuration and alias names cannot end with '--R'. This tag is reserved for use by HCM.
- Avoid using names starting with "Sync" (case-insensitive) because in the future, Synchronicity may define new keywords using that naming convention.
- A configuration's name must be unique within the scope of the module. When you create a configuration with **hcm mkconf**, if you specify a name that is the same as an alias or release of the module, the mkconf operation fails.
- A release's name must be unique within the scope of a module. When you create a release with **hcm release**, if you specify a name that is the same as any of the module's other configurations, the release operation fails. However, you can use the same release name for another module's configuration.
- The scope of an alias name is confined to the scope of a module. You can use the same alias name for any number of modules, but each alias must point to a different module's release.
- When you create a new alias with **hcm mkalias**, if you specify a name that is the same as an existing release, branch, or selector configuration, the mkalias operation fails.
- When you use **hcm mkalias** to change the release to which an alias points, specify a name that matches an existing alias name.

Related Topics

hcm mkalias Command

hcm mkconf Command

hcm release Command

The Relationship Between HCM Modules and ProjectSync Projects

An HCM module and a ProjectSync project are two different entities; however, with the SOC Developer Suite, HCM modules and ProjectSync projects are often associated. This association allows you to take fullest advantage of HCM capabilities.

An association between module and project can occur with the creation of HCM modules. For example, when the **hcm upgrade** operation upgrades vault directories to modules, ProjectSync projects are created accordingly. In addition, in certain cases, the **hcm mkmod** operation not only creates an HCM module but also creates a ProjectSync project. Two cases where the **hcm mkmod** operation creates a ProjectSync project are:

- When the files for the module reside in a DesignSync vault directory directly below the Projects directory. In this case, the **hcm mkmod** operation creates a ProjectSync project associated with the vault directory. For an example of this use of the **hcm mkmod** operation, see [The ALU Team Creates a Module from a Vault](#).

Note: While modules can reside anywhere in the vault, Synchronicity recommends that you place modules in the Projects vault folder. Locating modules in this directory allows greatest use of ProjectSync features.

- When a ProjectSync project already exists for a vault directory and has ProjectSync configurations that correspond to DesignSync configurations. In this case, the **hcm mkmod** operation makes the project's configurations available in HCM as configurations of the module.

Of course, not every ProjectSync project has an associated HCM module. However, when a ProjectSync project has an upper-level module associated with it, the data that makes up the project includes:

- Notes logged on the project
- (Optionally) The data of the project's uppermost-level module. The module's data is design data associated with the top level of the design, for example, the footprint of a chip or RTL describing the logical breakdown of the chip. Data might also be specifications and test files for that level of the chip.

The **hcm rmmod** operation removes the ProjectSync project and/or configurations associated with a module when the **-vaultdata** option is specified. (If the **-vaultdata** option is not specified, ProjectSync still displays the module and its configurations as a

project.) The `rmmod` operation also detaches ProjectSync notes attached to the module and its configurations. If the `-notes` option is specified, the operation deletes those notes that were detached and not attached to other objects.

Related Topics:

Module

`hcm mkmod` Command

`hcm rmmod` Command

Using ProjectSync with HCM

What Are Projects? (in ProjectSync help)

How the Release Operation Works

A release is a special type of configuration that is "frozen"; the files and hierarchical references it contains cannot be changed. Such a frozen configuration allows a design team to cycle back to a known state of the design at any point in the future. A release is created when you perform the `hcm release` operation (or use the **Release a module** dialog) on a module configuration. (**Note:** Although the contents of a release cannot be changed, the release can be removed from the server with the `hcm rmmod` command or the **Delete** dialog. This operation removes a module's configuration definitions as well as the module itself.)

The release operation is a client-side operation; you perform the operation on a module and its submodules in your work area on a DesignSync client. The release operation determines the contents of the release configuration from your work area and creates the release on the server. By default, the release operation is recursive.

When a recursive release operation is performed on a module, the operation creates:

- A release of the each submodule that is a branch, selector, or default configuration.

If a submodule resides on a remote SyncServer, the release operation creates its release on that server. (The operation checks access controls to ensure the user is allowed to run the release command on that server.)

- A release of the upper-level module.
- A hierarchical reference from the upper-level module release to each submodule's release.

For example, for the following hierarchy:

CPU (default configuration)
 ALU@C4
 FPU@Dev1024

This **hcm release** command (includes **-recursive** by default):

```
% stclc
stcl> hcm release -path /users/Robert/CPU -name Rel1
```

Performs these actions:

- Creates a release of the FPU@Dev1024 module configuration (FPU@Rel1) on the FPU module's SyncServer. Then creates a release of the ALU@C4 module configuration (ALU@Rel1) on the ALU module's SyncServer.
- Creates a release of the CPU module's default configuration (CPU@Rel1) on the CPU module's SyncServer.
- Creates hierarchical references from ALU@Rel1 to FPU@Rel1 and from CPU@Rel1 to ALU@Rel1.

For a release operation on a module hierarchy, how the release operation handles submodule configurations depends on the type of configuration:

If the module being released has a hierarchical reference to...	Then the release operation...
An IP Gear deliverable	Adds a reference from the release of the upper level module to the deliverable
A default configuration	<ul style="list-style-type: none"> • Creates a release of the default configuration • Adds a reference from the release of the upper level module to the release of the default configuration
A branch configuration	<ul style="list-style-type: none"> • Creates a release of the branch configuration • Adds a reference from the release of the upper level module to the release of the branch configuration
A release	Adds a reference from the release of the upper level module to the release
An alias	Adds a reference from the release of the upper level module to the release to which the alias points
A selector configuration	<ul style="list-style-type: none"> • Creates a release of the selector

	<p>configuration</p> <ul style="list-style-type: none"> • Adds a reference from the release of the upper level module to the release of the selector configuration
A vault folder on a SyncServer that does not have HCM installed	<p>Adds a reference from the release of the upper level module to the folder and issues a warning.</p> <p>Note: Because the contents of the referenced folder can change, a release that includes this type of reference is not immutable.</p>

Note: 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.

Related Topics

hcm release Command

Releasing a Single Module

hcm rmod Command

Deleting HCM Objects

How HCM Operations Handle an Alias

An alias is a symbolic name for a release. You can use an alias in HCM operations as you would other module configuration types. For example, you can get files of an alias or create a hierarchical reference to an alias. This section describes how HCM operations handle an alias.

The hcm get Operation

If you use a nonrecursive get operation on a module configuration that is an alias, the operation identifies the release to which the alias points and fetches the files of the release to your work area. (**Note:** The **hcm get** operation is nonrecursive by default.)

If you use **hcm get -recursive** on a module configuration that is an alias or on a module hierarchy that includes an alias, the operation identifies the release to which the alias points and fetches the files of the release. The get operation also follows hierarchical references to the configuration's submodules (if any) and fetches their files. Files are fetched into the directory locations indicated by the relative path of each hierarchical reference.

If you use the **hcm get** operation after an alias changes, the get operation fetches the files of the release to which the alias currently points. For example, suppose the alias ALU@GOLDEN was changed from ALU@R1 to point instead to ALU@R2. The next time users get the ALU@GOLDEN configuration, the get operation fetches ALU@R2.

Note: If you are fetching the module configuration into the same work area directory where you fetched the previous configuration, the get operation removes files in the work area that don't match the contents of the new configuration on the server, unless the files are locally modified. This behavior is standard for the **hcm get** operation, which applies the **-replace** option by default.

The hcm release Operation

The **hcm release** operation "freezes" the module hierarchy; making it immutable. Aliases by nature are dynamic and can be changed. When the release operation encounters an alias in the module hierarchy, the operation resolves the alias to the release to which it points. Then the operation adds a hierarchical reference from the upper-level module to the release.

For a description of how ProjectSync subscription and query mechanisms and the NoteAttach functionality handle an alias, see Using ProjectSync with HCM.

Related Topics

hcm get Command

hcm mkalias Command

hcm release Command

Module Cache

A **module cache** is a directory residing on the same local area network (LAN) as a group of users and containing copies of releases fetched from an HCM server.

Note: A module cache should contain only HCM releases. The get operation ignores default, branch, and selector configurations, as well as aliases in the module cache.

Using a module cache can reduce fetch time and save disk space for users. For example, a design team working on the files of a module configuration can set up a module cache to store copies of releases that the configuration references. When team members fetch the configuration, they can specify that the get operation fetch any releases from the module cache instead of the server. Because releases are fetched from the local module cache, the overall time required to fetch a module configuration is reduced. In addition, team members can save disk space by creating links from their

work areas to releases in the module cache, instead of fetching releases to their work areas.

Users can fetch from the module cache any releases that:

- Are available in the module cache. (To determine if a release is available, use the **hcm showmcache** command or the **Show contents of module caches** dialog.)
- Match the hierarchical nature of the hcm get operation being performed.

If you specify an **hcm get -recursive** operation, then for releases to be fetched from the module cache, the entire hierarchy of each release being fetched must exist in the module cache. If you specify an **hcm get** operation without the **-recursive** option, then just the upper-level module of each release being fetched must exist in the module cache. Otherwise, the release is fetched from the SyncServer. (To determine the nature of a release's hierarchy in the module cache, use the **hcm showmcache** command or the **Show contents of module caches** dialog.)

To allow a design team to take advantage of the benefits of a module cache, a design team leader would first set up one or more module caches. Typically, the team leader would then define the DesignSync registry settings that specify the default module cache paths and default mode for fetching from the module cache (link or copy). Once the registry settings are defined, team members can use **hcm get** as usual to fetch releases from the module cache. Team members can override the default settings by using **hcm get** with its **-mcachepaths** and **-mcachemode** options. These options can also be used when registry settings haven't been set.

For more information about using the **-mcachepaths** and **-mcachemode** options, see the hcm get command. For a scenario about fetching from or linking to a module cache, see Designers Use the Module Cache.

Note: A module cache is different from a DesignSync LAN cache. See A Comparison of Module Caches and DesignSync Caches for information.

Related Topics

hcm get Command

Getting a Module

hcm showmcache Command

Showing the Contents of Module Caches

Setting Up a Module Cache

Setting the Default Module Cache Path or Mode

A Comparison of Module Caches and DesignSync Caches

In general, a module cache is similar to a DesignSync LAN cache. Both cache types are directories residing on the same local area network (LAN) as a group of users and containing copies of objects in the DesignSync vault. Both serve the same purpose: to save time on fetches of updated design data and to save disk space.

However, the two cache types differ in their contents, method of access, and setup/maintenance.

- A module cache stores copies of HCM module releases. (**Note:** Only releases should be stored in the module cache; other HCM configuration types are ignored by HCM operations.)

A DesignSync cache stores copies of DesignSync objects (file versions and Cadence collections).

- To use the module cache, HCM users specify the **-mcachemode** and **-mcachepaths** options with the **hcm get** operation.

To use the LAN cache, DesignSync users specify the **-share** option with the DesignSync **populate**, **co**, **ci**, or **cancel** operation.

- Both DesignSync LAN caches and module caches let users link from their work areas to items in the cache instead of fetching from the server. HCM users can choose one of these modes:
 - A link to the base directory of the release in the module cache (**-mcachemode link**)
 - A copy of the release from the cache (**-mcachemode copy**)
 - A copy of the release from the server (**-mcachemode server**)

For DesignSync users, however, specifying the **-share** option always results in a symbolic link from your work area to the file in the cache.

- Any number of module caches can be set up on a LAN; the number is constrained only by local disk space. The **hcm get** operation lets users specify a list of caches from which to fetch.

Due to the way the DesignSync cache is set up, only one DesignSync LAN cache can be designated.

- A module cache is set up after HCM installation by a project leader or team member who will be responsible for owning and updating the cache. To ensure that releases cannot be changed, the module cache directory should have UNIX read-only permissions for all users except the owner. The cache owner uses the **hcm get** operation to fetch each release to the cache. In the same way, the cache owner updates the cache with new releases. (The cache owner can set up automated scripts to update the module cache when module releases are created.) For more information, see *Setting Up a Module Cache and Updating a Module Cache*.

A DesignSync cache is set up during Developer Suite installation, when the installer specifies the location of the default cache directory (`sync_cache`). A LAN administrator can later specify that a directory on the LAN be used as the default cache directory instead. The cache directory must have UNIX permissions that grant full access to users of the cache because certain user operations update the cache directory. (**Note:** An alternative to granting full access to cache users is to perform a SUID installation of DesignSync, which enforces the read-only intent of DesignSync mirror and cache directories.) The LAN administrator does not fetch files to the cache; file versions are fetched to the LAN cache when users specify the **-share** option with the DesignSync **populate**, **ci**, **co**, or **cancel** command. The cache is also updated in the same way. For more information, see *ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide: Setting Up a LAN Cache*.

Related Topics

ENOVIA Synchronicity DesignSync Data Manager User's Guide: What Is a File Cache?

Module Cache

Setting Up a Module Cache

Using HCM from DesignSync

Commands in the DesignSync GUI client

HCM commands have limited support in the DesignSync GUI. You can use the DesignSync to perform the following operations:

- Viewing Modules and Configurations
- Viewing a Module Hierarchy
- Getting a Module
- Showing the Status of a Configuration
- Deleting HCM Objects
- Showing the Contents of Module Caches

Additionally, while in the DesignSync GUI, you can type all available HCM commands directly into the command bar which will graphically display the results of the command in the Output Window and update the display in the List View pane.

Viewing Modules and Configurations

You can use DesignSync GUI (Tree View and List View) to display modules and configurations that reside on the server or that you have fetched to your work area.

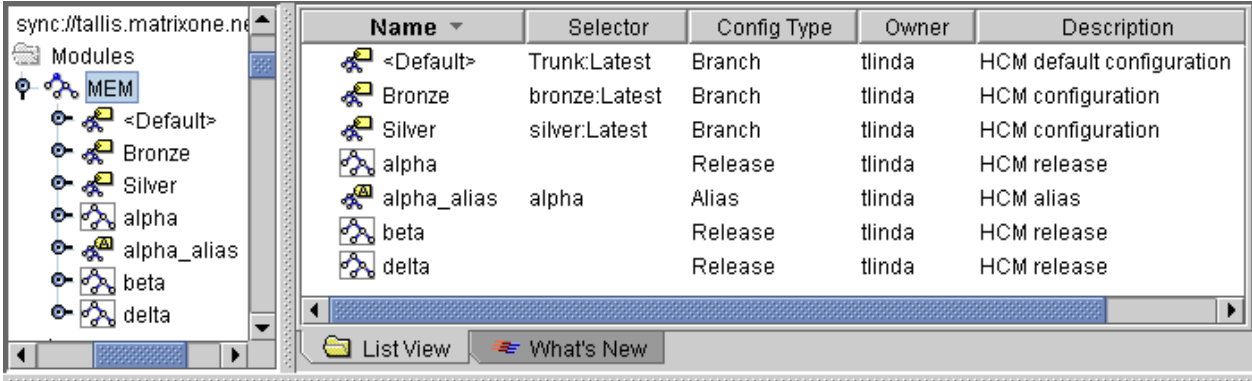
Viewing Modules and Configurations on the Server

Displaying modules and configurations that reside on the server is useful when you want to display a list of module configurations in order to fetch one to your work area (with the `hcm get` operation). In addition, it is from the server that you create and delete module configurations, hierarchical references, and aliases for releases.

To view modules and configurations on the server:

1. In DesignSync **Tree View**, select the server where the modules reside. (Type the path to the server in the Location field or click the server from the My Servers or Visited Servers list.)
2. In Tree View, click the **Modules** folder. DesignSync **List View** lists the modules (in alphabetical order by module folder name).

Once you have selected a module in DesignSync Tree View, you can expand it to display its configurations. You can then select a configuration and display the submodules that it contains. This example shows the submodules in the CPU module's default configuration:



To view a list of hierarchical references for a module configuration, click the configuration name in Tree View. DesignSync List View lists the hierarchical references for the configuration.

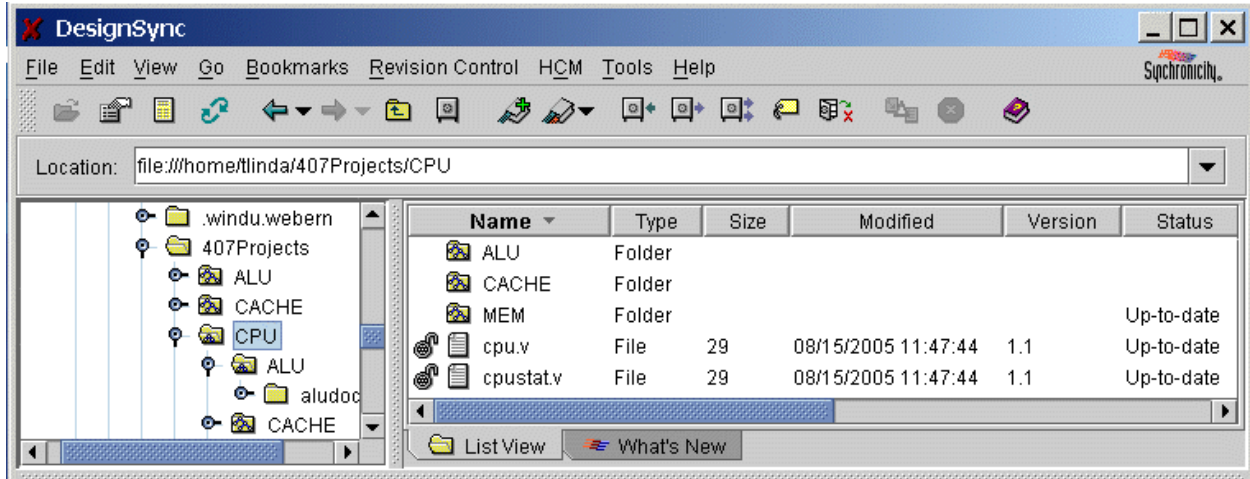
See Viewing a Module Hierarchy for information on displaying an entire module configuration hierarchy (a configuration's submodules, the submodules of those submodules, and so on).

Viewing Modules and Configurations in Your Work Area

From DesignSync Tree View and List View, you can display modules and configurations in your work area. You use this display when you want to view the files of a configuration you have in your work area, update (get) a module configuration you previously fetched, put a configuration back on the server (check in your changes to its files), or create a release of a configuration.

The display of modules and configurations in your work area differs from that on the server:

- The work area display does not show hierarchical references. To view a configuration's hierarchical references, you need to display the hierarchy of modules in the configuration. See Viewing a Module Hierarchy.
- When you click a module or configuration folder in your work area, DesignSync List View displays the folder's contents and not the module hierarchy. This example shows the contents of the CPU module's default configuration in a work area:



Moving from a Workspace Configuration to the Configuration on the Server

Much of the time when you work with module configurations, it will be in your workspace: fetching a configuration to your workspace, creating and modifying design files, then putting the configuration back on the server. But there are times when you need to work with modules and configuration on the server. For example, creating (and deleting) configurations, hierarchical references, and aliases.

To move from a workspace configuration to that configuration on the server:

1. In DesignSync **Tree View**, in your workspace, click the module configuration's base directory in your workspace.
2. Select **HCM=>Go to Configuration**. DesignSync Tree View goes to the module configuration on the server and highlights it.

Tip: To see if your workspace configuration is up-to-date, you can display the status of the configuration in your work area as compared that same configuration on the server. See Showing the Status of a Configuration.

Related Topics

Viewing a Module Hierarchy

DesignSync Help: Tree View Pane

DesignSync Help: List View Pane

Viewing a Module Hierarchy

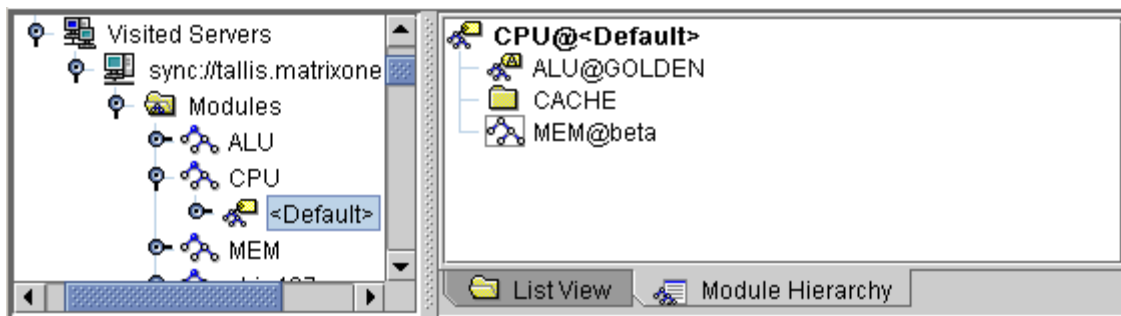
Viewing a Module Hierarchy

You can display an entire module configuration hierarchy that resides on the server or that you have fetched to your work area with the hcm get operation. **Note:** The hierarchy displayed on the server can differ from that in your work area.

To view the hierarchy of a module configuration on the server:


1. In DesignSync **Tree View**, select the server where the module configurations reside. Then click the **Modules** folder to display the modules.
2. In **Tree View**, click the module and then click the configuration you want to display.
3. Select **HCM=>Module Hierarchy**. DesignSync displays the hierarchy in the **Module Hierarchy View**.

This example shows the hierarchy for the default configuration of the CPU module on the server:



Note: Module Hierarchy view displays hierarchical references to valid HCM Modules

with the Configuration Icon  Release icon  or Alias icon  , as appropriate.

Module Hierarchy view displays the following hierarchical reference target objects with a standard Folder icon :

- Level hierarchical references to IPGear Deliverables
- DesignSync Vaults and ProjectSync configurations
- Any hierarchical reference that cannot be located

To view the hierarchy of a configuration you have fetched to your work area:

1. In DesignSync Tree View, select the folder for the configuration.
2. Select **HCM=>Module Hierarchy**. HCM displays the module hierarchy in Module Hierarchy View.

Tip: To go from a configuration in Module Hierarchy View to the List View of its contents, right-click the configuration and select **Visit** from the pop-up menu.

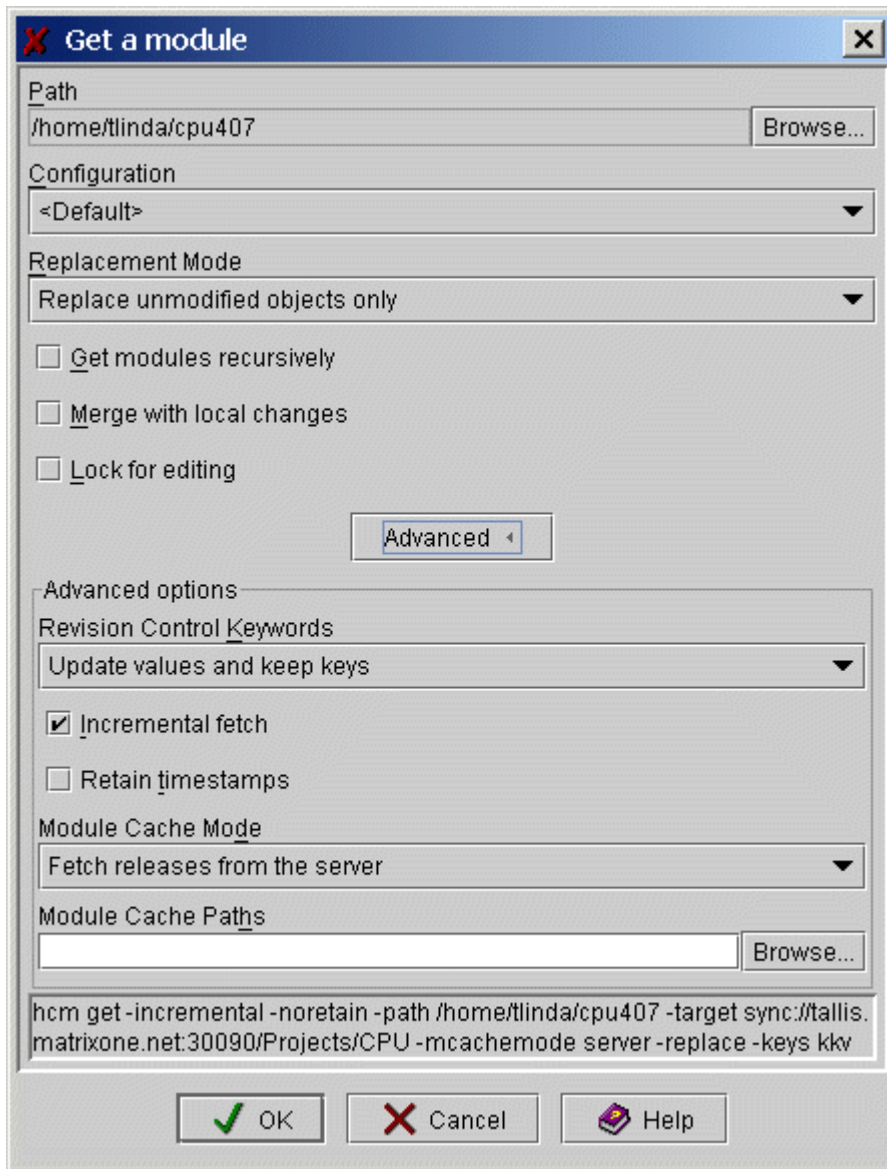
Related Topics

Viewing Modules and Configurations

Getting a Module

The **Get a module** dialog lets you fetch a module configuration to your work area or update a configuration you previously fetched.

Click the following illustration for information.



To get a module configuration to your work area or update a configuration:

1. In DesignSync **Tree View** or **List View**, on the server, click the module or configuration you want to fetch to your work area. To update a module configuration you already fetched to your work area, click that configuration in your work area.

Notes:

- You cannot use the **Get a module** dialog to fetch files from a DesignSync vault. To fetch files from a vault, use the **hcm get** command.
- To update a module you already fetched to your work area, you must select the base configuration of the module to update it. You cannot

update a module from its subdirectory or from another module in the module hierarchy.

2. Select **HCM=>Get**.

HCM displays the **Get a module** dialog.

3. In the **Path** field, click **Browse** to select the work area directory where you want to place the module configuration. (You can also type the absolute path to the directory.)
4. Select other options as needed.
5. Click **OK**.

Path

Specifies the path to the directory in your work area where you want the fetched module to reside. Click **Browse** to select your work area directory where you want to place the module configuration. You can also type the absolute path to the directory.

Configuration

Specifies the name of module configuration you are fetching. This field is required if you selected a module or configuration on the server.

If you selected a module on the server, this field displays Default and HCM fetches the default configuration of the module. To fetch another configuration of the module, select its name from the pull-down menu.

If you selected a configuration on the server, this field displays the name of the configuration you selected.

If you selected a configuration in your work area, this field displays the name of the configuration.

Get modules recursively

Specifies a fetch of only the top-level module (the default) or a fetch of the top-level module and each submodule in its hierarchy.

Incremental fetch

An incremental fetch updates the directories of the configuration in your work area only if the corresponding DesignSync vault folders have been modified since you last fetched the configuration. This option specifies whether the hcm get operation performs an incremental fetch.

The hcm get operation follows the registry setting for **Optimizations => Perform incremental populate**. By default, this setting is enabled; therefore, the option displays a check and the get operation performs an incremental fetch. If the checkbox is unchecked, the hcm get operation updates all of the directories of the configuration in your work area. To disable the default setting, your Synchronicity administrator can use the SyncAdmin tool. For information, see SyncAdmin Help: Command Defaults.

Merge with local changes

Fetches the Latest versions from the configuration on the server and merges them with the current, locally modified versions.

You cannot use this option with Lock for editing.

Lock for editing

Locks the files of the module in DesignSync so that only you can edit them. By default, this option is not checked.

You cannot use this option with **Merge with local changes** or with **Incremental fetch**.

Retain timestamps

Retains the last modified timestamp of the fetched objects as recorded when the object was checked into the vault.

The hcm get operation 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 get operation. To change the default setting, your Synchronicity administrator can use the SyncAdmin tool. For information, see SyncAdmin Help: Command Defaults.

Replacement mode

Determines how the get operation updates your work area with the files of the modules you are fetching. You can specify the update method the get operation uses by selecting one of the following replacement mode options:

- **Don't replace any objects.** This option preserves local modifications you made to objects/files and leaves intact any objects that are in your work area but are no longer part of the configuration.
- This option causes the least disruption to your work area; however, it may require you to clean up resulting work area data.
- **Replace unmodified objects only.** (The default.) This option updates managed objects that have not been locally modified and that are part of the configuration

being fetched. The operation also removes any unmodified managed objects that are not part of the configuration being fetched.

- This option leaves intact any managed objects you have locally modified.
- **Force overwrite of modified objects.** This option updates the managed objects in our work area. The option replaces or removes managed objects whether you have modified them locally or whether they are part of the configuration being fetched or not.
- This option forces your work area's managed objects to match the set of objects in the configuration being fetched.

Revision Control Keywords

Controls the processing of RCE revision-control keywords. You can select:

- **Update values and keep keys.** Expands keyword values and retains the keywords in the file (default option). For example: \$Revision 1.4 \$
- **Update values and remove keys.** Expands keyword values but removes keys from the file. This option is not recommended when you check out files for editing. If you edit and then check in the files, future keyword updates are impossible, because the value without the keyword is interpreted as regular text. For example, 1.4.
- **Remove values and keep keys.** Keeps the keywords but removes keyword values. This option is useful if you want to ignore differences in keyword expansion, such as when you are comparing two different versions of a file. For example, \$Revision: 1.8 \$
- **Do not update.** Keeps exactly the same keywords and values as were present at checkin.

Module Cache Mode

Fetches the target configuration from the module cache instead of the server. Using module cache 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. See Showing the Contents of a Module Cache for information.)

- **Link to releases in the module cache.** (Unix only) For each release it finds in the module cache, the hcm get operation 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.
- **Copy releases from the module cache.** For each release it finds in the module cache, the hcm get operation copies the release to your work area. **Note:** This mode is the default mode on Windows platforms.
- **Fetch releases from the server.** Causes the hcm get operation to fetch releases from the server, not from the module cache.

- This option overrides the default module cache mode registry setting. If the registry value does not exist, the **Module Cache Mode** selection defaults to **Link to releases in the module cache** (Unix platforms) or to **Copy releases from the module cache** (Windows platforms).

Module Cache Paths

Specifies paths to the module caches that the hcm get operation searches. Click **Browse** to select one or more paths or type paths in the field. Paths must exist. If you do not specify a module cache path, the get operation fetches the module from the server.

If you type paths in the field, you must specify the absolute path to each module cache. To specify multiple paths, separate them with a comma (,).

OK

Closes the form and performs the operation.

Cancel

Closes the form without performing the operation.

Help

Invokes HCM Help and displays the topic associated with the current dialog.

Related Topics

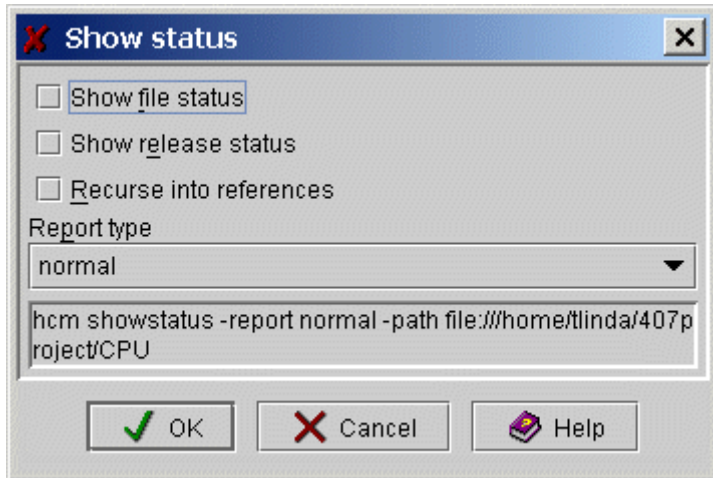
hcm get Command

An ALU Designer Gets Files of a Module

A CPU Designer Gets a Module Hierarchy

Showing the Status of a Configuration

The **Show status** dialog lists the status of a configuration in your work area as compared to the server. You can use this dialog to verify that your work area configuration is up-to-date.



To show the status of a configuration:

1. Click the work area folder that contains the top-level module (base directory) of the configuration.
2. Select **HCM=>Show=>Module Status**. HCM displays the **Show status** dialog.
3. Optionally, you can select to display the status of files (in addition to status of the hierarchical references), status of releases, and the status of all submodules in the configuration. You can also select a different report type.
4. Click **OK**. HCM displays the status report in **Module Status View**.

Show file status

Displays the status of each object contained by the work area configuration as compared to the configuration on the server. If you specify this 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 select **Show file status**, the value for configuration status reflects the status of the hierarchical references only.

Show release status

Displays 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 **Show file status** 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.

Recurse into references

Displays the status for the specified configuration and all referenced configurations and identifies why particular hierarchical references are not recursed.

Report type

Specifies the type of status information to be displayed and the format in which the output appears.

You can select:

- 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 output, see the hcm showstatus Command in Synchronicity Command Reference Help.)
- brief - Displays a summary and lists hierarchical references that are out-of-date. (Optionally, you can use this option to list file status for files that are out-of-date.)
- summary - Displays the status of each configuration and reports the overall status of the configuration in the work area.

OK

Closes the form and performs the operation.

Cancel

Closes the form without performing the operation.

Help

Invokes HCM Help and displays the topic associated with the current dialog.

Related Topics

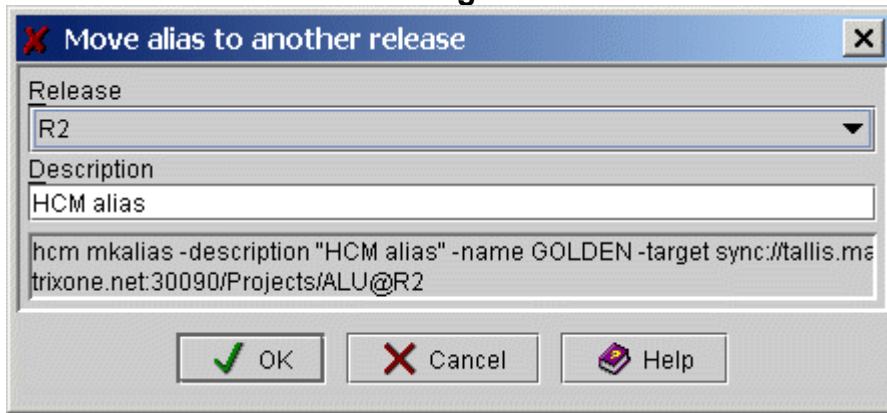
hcm showstatus Command

The CPU Team Creates a Release

Moving an Alias to Another Release

You can use the **Move an alias to another release** dialog to move an alias from one release to another.

Click the fields in the following illustration for information.



To move an alias:

1. In DesignSync **Tree View**, on the server, click the module to display its configurations.
2. In **Tree View** or **List View**, click the alias you want to move.
3. Select **HCM=>Move Alias**.
4. From the **Release** pulldown menu, select the release to which you want to move the alias.
5. Click **OK**. HCM moves the alias to the release you selected.

Release

Specifies the release to which you want to move the alias. From the pulldown menu, select a release. The menu displays only existing, valid releases for the configuration. The default is the current release to which the alias is assigned.

Description

Specifies a description for the alias. In this field, HCM displays the description of the alias you selected; you can change the description. This field is optional.

OK

Closes the form and performs the operation.

Cancel

Closes the form without performing the operation.

Help

Invokes HCM Help and displays the topic associated with the current dialog.

Related Topics

Alias

The ALU Team Creates an Alias for a Release

hcm mkalias Command

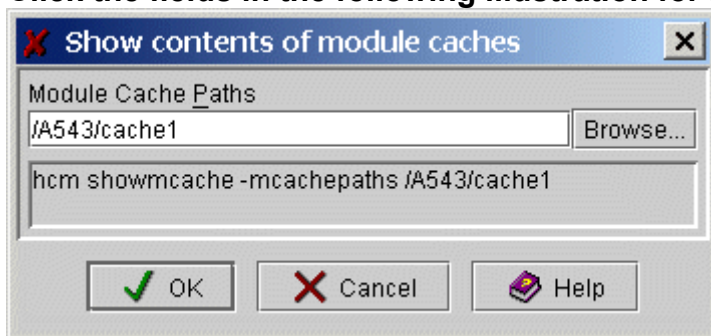
Showing the Contents of Module Caches

To reduce fetch time of large configurations and to save disk space, a design team can set up one or more module caches. The caches contain released configurations for the design the team is working on. Designers can then fetch those releases from the cache or link to them to save disk space.

The **Show contents of module caches** dialog lets you search module caches for releases you can fetch with the get operation.

In order for you to select a module cache to search, your Developer Suite administrator must first set up that module cache.

Click the fields in the following illustration for information.



To display the contents of module caches:

1. In DesignSync, from the HCM menu, select **Show=>Module Cache**.
HCM displays the Show contents of module caches dialog.
2. In the **Module Cache Paths** field, click **Browse** to select one or more module caches to search or type the absolute path to the module cache. **Note:** If your Developer Suite administrator has defined the DesignSync registry setting for default module cache paths, HCM displays that module cache in the **Module Cache Paths** field.
3. Click **OK**.

Module Cache Paths

Specifies the module caches to search. The paths must already exist. The field is required.

Click **Browse** to select one or more module caches to search or type the absolute path to the module cache. To specify more than one path, separate paths with a comma (,).

HCM searches the module caches in the order specified with the Module Cache Paths field or the registry setting.

OK

Closes the form and performs the operation.

Cancel

Closes the form without performing the operation.

Help

Invokes HCM Help and displays the topic associated with the current dialog.

Related Topics

Designers Use the Module Cache

Setting up a Module Cache

Setting the Default Module Cache path or Mode

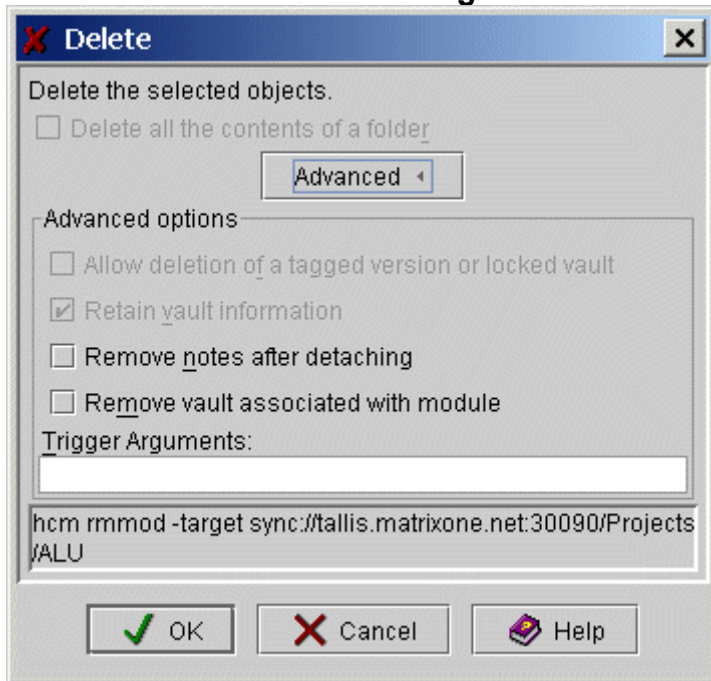
hcm showmcache Command

Deleting HCM Objects

The **Delete** dialog lets you delete HCM modules, configurations and aliases from the server. (To delete hierarchical references from a module configuration, see Removing Hierarchical References from a Module Configuration.)

Note: By default, HCM denies access to delete modules, configurations, and aliases for all users. Your Synchronicity administrator can customize this access. See Access Controls on HCM Operations.

Click the fields in the following illustration for information.



Remove notes after detaching

Removes those notes that were detached (by the operation that deletes the module, configuration, or alias) and not attached to other objects. This option is available only if you selected a module, configuration or alias for deletion.

By default, DesignSync does not remove notes (the checkbox is not checked).

Remove vault associated with module

Removes the vault folder in which the module contents reside and deletes all objects (even locked objects) in that vault. This option is available only if you selected a module for deletion.

By default, DesignSync does not remove the vault when deleting a module (the checkbox is not checked).

Trigger Arguments

Type an argument to be passed from the command line to the triggers set on the delete operation. Consult your project leader for information about any triggers that are in use and how they use arguments.

To delete an HCM object:

1. In DesignSync **Tree View**, select the server where the HCM object (module, configuration, or alias) resides. Then click the **Modules** folder to display the modules. **Note:** Objects must be deleted from the server. For example, you cannot delete a configuration by selecting it in your workspace.
2. Click a module to select it for deletion or continue expanding the hierarchy to select configurations (including aliases). **Note:** To delete the default configuration or a release configuration of a module, you must select and delete the module.
3. Select **File=>Delete** or **HCM=>Delete**. DesignSync displays the **Delete** dialog. (**Note:** To delete a hierarchical reference from a configuration, select **HCM=>Remove Href from Parent**. See Removing Hierarchical References from a Module Configuration.)
4. Optionally, select **Advanced Options** to:
 - Remove notes after detaching
 - Remove vault associated with a module
5. Click **OK**.

HCM deletes the objects you selected:

- If the object is a module, HCM removes the module and all its configurations from the server and detaches the ProjectSync notes that are attached to the module and its configurations. The operation also generates a RevisionControl note.
- If the object is a configuration, HCM removes the configuration from the server. It removes all hierarchical references from the deleted configuration and detaches any ProjectSync notes that may have been attached to the configuration. The operation also generates a RevisionControl note for the configuration and attaches the note to the owning module.
- If the object is an alias (symbolic name for a release), HCM removes the alias and detaches any ProjectSync notes that may have been attached to the alias. The delete (rmalias) operation also generates a RevisionControl note and attaches it to both the owning module and the release to which the alias points.

OK

Closes the form and performs the operation.

Cancel

Closes the form without performing the operation.

Help

Invokes HCM Help and displays the topic associated with the current dialog.

Related Topics

The MPU Team Removes a Module

hcm rmmod Command

A Designer Removes a Configuration

hcm rmconf Command

The ALU Team Removes an Alias

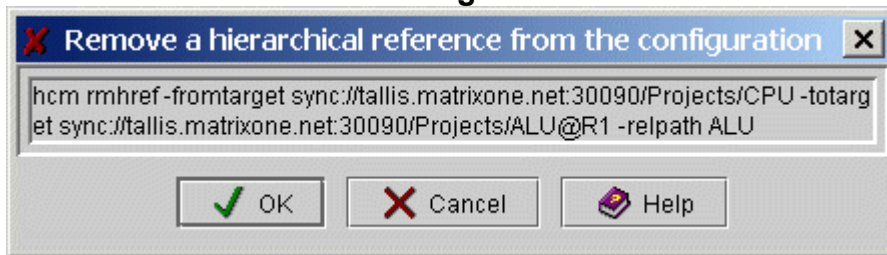
hcm rmalias Command

Removing Hierarchical References from a Module Configuration

Removing a Hierarchical Reference from a Module Configuration

The **Remove hierarchical reference from the configuration** dialog lets you delete the hierarchical reference from a module configuration to a submodule configuration.

Click the fields in the following illustration for information.



To remove a hierarchical reference from a configuration:

1. In DesignSync **Tree View**, select the server where the upper-level module configuration resides. (**Note:** Hierarchical references must be deleted from the server.) Then click the **Modules** folder to display the modules.
2. In **Tree View**, click the module you want. Then click the configuration from which you want to remove a hierarchical reference (the "from target").
3. Select **HCM=>Show Module Hierarchy**. DesignSync displays the configuration's hierarchical references in Module Hierarchy View.
4. In **Module Hierarchy View**, right-click the hierarchical reference you want to remove (the "to target"). The hierarchical reference can be a submodule configuration, a DesignSync vault, or an IP Gear deliverable. For example, to remove the hierarchical reference from the CPU default configuration to the submodule configuration ALU@R1, you would right-click ALU@R1.
5. From the pop-up menu, select **Remove Href from parent**.
6. On the **Confirm** dialog, click **OK**.

HCM removes the hierarchical reference from the upper-level module configuration to the submodule.

OK

Closes the form and performs the operation.

Cancel

Closes the form without performing the operation.

Help

Invokes HCM Help and displays the topic associated with the current dialog.

Scenarios for Using HCM

Overview

The scenarios in this section describe how design teams use the SOC Developer Suite to share and reuse design work at a fictitious company called ABCo. ABCo has design teams that are geographically distant. The designers' tasks are typical of those you might perform, such as fetching files, updating your work area, and making files for a block available for reuse in other teams' designs.

The chip architect has partitioned the overall design into major blocks and located the IP for certain blocks that the project will incorporate from another team rather than develop itself. The table shows the blocks and their sources, as well as whether the development team uses HCM for development of the block:

Block	Design Source	Server/Location	Synchronicity Tool
chip407	New design from chip architect	srvr1.ABCo.com	SOC Developer Suite (DesignSync + HCM)
CPU	New design from CPU team	cpu.ABCo.com	SOC Developer Suite
ALU	New design from ALU team	alu.ABCo.com	SOC Developer Suite
IOSTAT	Design from third party	ipgsrvr1.IOCo.com	IP Gear
CACHE	New design from CPU team	cpu.ABCo.com	SOC Developer Suite
DesignCompiler	CAD Tools Group of CPU team	cpu.ABCo.com	SOC Developer Suite
PLL	New Design from CPU team	cpu.ABCo.com	SOC Developer Suite
RAM	Reuse of design from RAM team	ram.ABCo.com	DesignSync (non-HCM enabled server)

Related Topics

Introducing HCM

Release Information

The Chip Architect Sets Up the Project Structure

After identifying the blocks for the SOC (Chip407) project (as shown in Overview), the chip architect determines the overall module hierarchy and creates its upper levels:

1. Using HCM, the chip architect creates the upper-level chip module:

```
% stcl
stcl> hcm mkmod -target
sync://srvr1.ABCo.com:2647/Projects/chip407 -description
"hcm module for 407 chip"
```

The **hcm mkmod** operation creates a default configuration of a module called Chip407. This module configuration resides on the srvr1 SyncServer.

The operation also creates a ProjectSync project because files for chip407 reside in a DesignSync vault folder directly below the Projects directory. While modules can reside anywhere in the vault, Synchronicity recommends that you place modules in the Projects vault folder. Locating modules in this directory allows greatest use of ProjectSync features. For an example scenario about creating a module from a DesignSync vault, see [The ALU Team Creates a Module from a Vault](#).

2. The chip architect sends email to the leaders of the teams designing the CPU and the Phase Locked Loop, telling them to create the CPU and PLL modules. Each project leader uses **hcm mkmod** to create a module. For an example scenario about creating a module and its files, see [The PLL Team Creates a Module and Its Contents](#).
3. The chip architect creates the next level of the chip407 hierarchy by adding hierarchical references from the chip407 module to the newly-created CPU and PLL modules:

```
stcl> hcm addhref -fromtarget
sync://srvr1.ABCo.com:2647/Projects/chip407 -totarget
sync://cpu.ABCo.com:2647/Projects/CPU -relpath CPU
stcl> hcm addhref -fromtarget
sync://srvr1.ABCo.com:2647/Projects/chip407 -totarget
sync://cpu.ABCo.com:2647/Projects/PLL -relpath PLL
```

Note: The **-relpath** option is not required. If a **-relpath** is not specified, the **addhref** operation uses the **-totarget** module name as the name of the submodule's base directory and places it directly below the upper-level module's base directory. (In these examples, the result would be the same as the specified value for **-relpath**.)

4. Chip407 will also incorporate an existing design for RAM. Files for that design reside on a non HCM-enabled server running DesignSync v3.1. To incorporate those files into the module hierarchy, the chip architect creates a hierarchical reference from the chip407 module to the RAM vault folder:

```
stcl> hcm addhref -fromtarget
sync://srvr1.ABCo.com/Projects/chip407 -totarget
sync://ram.ABCo.com/Projects/RAM -relpath RAM
```

5. To get an idea of what the hierarchy looks like so far, the chip architect uses the **hcm showhrefs** operation:

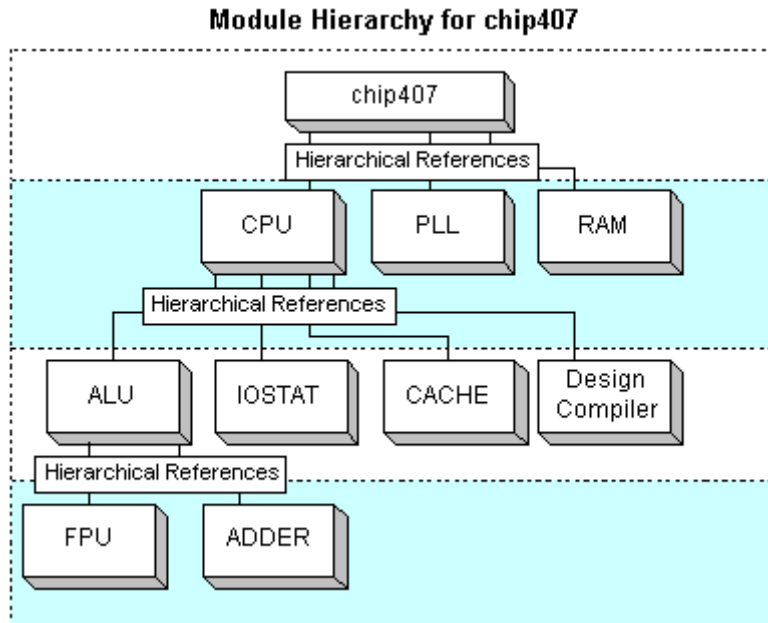
```
stcl> hcm showhrefs -target
sync://srvr1.ABCo.com/Projects/chip407 -report brief

Hierarchy for sync://srvr1.ABCo.com/Projects/chip407
TARGET      URL
-----
chip407     sync://srvr1.ABCo.com/Projects/chip407
  CPU       sync://cpu.ABCo.com:2647/Projects/CPU
  PLL       sync://cpu.ABCo.com:2647/Projects/PLL
  RAM       sync://ram.ABCo.com/Projects/RAM
```

The CPU team further develops its design, incorporating work from other design teams by adding hierarchical references to modules from those teams. (Other scenarios in this section show the creation of other teams' modules and incorporation of them into the CPU module.)

Note: To separate this project structure from the ongoing change of block development, each submodule team can set up an alias for releases of their designs and the architect can modify the hierarchy to reference each submodule's alias instead of a specific release. Then, when a team creates a new release of a module, the hierarchy will automatically include it. For an example scenario, see *The ALU Team Creates an Alias for a Release*.

Eventually, the chip407 module hierarchy looks like this figure:



Related Topics

[hcm addhref Command](#)

[hcm mkmod Command](#)

[hcm showhrefs Command](#)

The PLL Team Creates a Module and Its Contents

(Creating a module for files that do not reside in a DesignSync vault)

Mike, the leader of the PLL team, receives email from the chip407 architect directing the team to create a module for their design of the Phased Locked Loop (PLL). Because the design is a new one, it has no files already existing in a DesignSync vault. The team will be creating design files for the module.

1. Mike first uses **hcm mkmod** to create the PLL module on the cpu SyncServer:

```

% stclc
stcl> hcm mkmod -target
sync://cpu.ABCo.com:2647/Projects/PLL -description "PLL
design for chip407"
  
```

The **hcm mkmod** operation:

- Creates a default configuration of a module called PLL. This module configuration resides on the cpu SyncServer. At this point, it does not exist in Mike's work area.
 - Creates a DesignSync vault folder (`PLL`) because there are no files already existing on the server. (When vault data already exists, the `mkmod` operation associates the module with that vault. For an example scenario, see *The ALU Team Creates a Module from a Vault*.)
 - Creates a ProjectSync project for the PLL module. This action allows ProjectSync to recognize the PLL module. PLL team members can use Query and other ProjectSync features to track and report on the module.
 - Includes the description (specified with the **-description** option) in the ProjectSync project. This information is also available in HCM through the **hcm showmods -report script** operation.
2. Mike next prepares to get the module to his work area and create some initial files for it. In his work area, he creates a folder for the module and changes directory to it:

```
stcl> cd /dev/users/Mike/407project
stcl> mkfolder PLL
stcl> cd PLL
```

3. He uses the **hcm get** operation to fetch the default configuration of the PLL module to his work area:

```
stcl> hcm get -target sync://cpu.ABCo.com:2647/Projects/PLL
```

Because Mike did not specify the **-path** option, the `get` operation uses his current work area directory as the place to which it fetches the module.

The `get` operation fetches the module to Mike's work area. At this point the module contains no files. However, the `get` operation associates Mike's work area directory with the module's vault folder on the SyncServer and fetches some metadata about the module to the work area.

4. Mike creates a file (`procnotes.txt`) containing information that all team members need to know about the project.
5. He uses the DesignSync `checkin` operation to add the file to the module's default configuration:

```
stcl> ci -new -comment "adding procedure notes"
procnotes.txt
```

Mike can use this `checkin` method because he is working with the PLL module's default configuration, a branch configuration. If he is working with a selector configuration, the steps for adding a file are different. See *A Designer Adds Files to a Configuration* for information.

The checkin operation places the file under DesignSync management (in the PLL vault folder) and includes the file in the default configuration of the PLL module.

Other members of the PLL team use the **hcm get** operation to fetch the default configuration of the PLL module to their work areas. Then they can add their new files to it with the **ci** command as Mike did, or they can modify the files already existing in the module. To check in existing files they modified, they can use the **hcm put** operation. For an example scenario, see A Designer Puts Files of a Module Back on the Server.

Related Topics

hcm addhref Command

hcm mkmod Command

ci Command

The ALU Team Creates a Module from a Vault

(Creating a module from a DesignSync vault)

One of the teams contributing to the Chip project, the ALU team, uses DesignSync to manage its design data. The vault location for their project is `sync://alu.ABCo.com:2647/Projects/ALU`. This location contains the specifications, documentation, RTL, layout, and schematics for the ALU design.

During the design development, ALU team members have used the standard DesignSync commands to check files in and out, update files, and so on. The team also uses ProjectSync to track defects and specifications.

Recently, the team has installed the SOC Developer Suite (DesignSync V3.3 and HCM) on its development SyncServer and is ready to create a module containing the ALU design files.

Anne, the ALU team leader, invokes the Synchronicity Tcl command shell (`stcl`) and uses the **hcm mkmod** operation to create the ALU module:

```
% stclc
stcl> hcm mkmod -target sync://alu.ABCo.com:2647/Projects/ALU -
description "hcm module for alu"
```

The `mkmod` operation:

- Defines the ALU vault folder as a module and creates a default configuration for it.

- Creates a ProjectSync project for the ALU module. This action allows ProjectSync to recognize the ALU module. ALU team members can use Query and other ProjectSync features to track and report on the module.

The `mkmod` operation creates a ProjectSync project because files for ALU reside in a DesignSync vault folder directly below the Projects directory. While modules can reside anywhere in the vault, Synchronicity recommends that you place modules in the Projects vault folder. Locating modules in this directory allows greatest use of ProjectSync features.

- Includes the description (specified with the **-description** option) in the ProjectSync project. This information is also available in HCM through the **hcm showmods -report script** operation.

Anne and the ALU team members can now use the **hcm get** operation to fetch the module to their work areas. (See An ALU Designer Gets Files of a Module for an example scenario for the **hcm get** operation.)

IMPORTANT:

After the ALU module has been created, ALU team members always use the **hcm get** operation to fetch the module's design files to their work areas. They no longer use the DesignSync populate operation to fetch files of ALU or any other HCM module.

Related Topics

`hcm get` Command

`hcm mkmod` Command

`hcm showmods` Command

An ALU Designer Gets Files of a Module

Thomas, a designer on the ALU team, wants to fetch the design files of the ALU module to his work area. To fetch the files, he uses the **hcm get** operation. (**Note:** This scenario shows fetching from the HCM server. For a scenario that shows fetching from the module cache, see Designers Use the Module Cache.)

1. To locate the module he wants to fetch, Thomas invokes the Synchronicity Tcl (`stcl`) command shell and uses the **hcm showmods** operation. (This operation lists all of the modules on a particular SyncServer.) For example:

```
% stclc
stcl> hcm showmods -target sync://alu.ABCo.com:2647
```

Modules hosted on server sync://alu.ABCo.com:2647

```
NAME  OWNER  VAULT  PATH
-----
ALU   Anne   /Projects/ALU
```

2. Then he uses **hcm showconfs** to list the configurations of the ALU module on the SyncServer:

```
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU
```

```
Configurations of module
sync://alu.ABCo.com:2647/Projects/ALU
```

```
NAME          TYPE          OWNER  SELECTOR/ALIASED  RELEASE
-----
<Default>    Branch       Anne   Trunk:Latest
C1           Selector    Anne   valid12.06.01
```

From the operation's output, Thomas sees that the first item is the default configuration of the ALU module. The ALU team uses this configuration for its development work, so he knows this is the configuration to fetch to his work area.

3. Thomas uses the **hcm get** operation to fetch all of the files for the ALU module's default configuration to his work area:

```
stcl> hcm get -target sync://alu.ABCo.com:2647/Projects/ALU
-path /data/devel/users/Thomas/ALU
```

Because Thomas specified the ALU module without a configuration for the **-target** option, the **hcm get** operation fetches the ALU module's default configuration.

Thomas wants only the files of the ALU module, so he does not specify the **-recursive** option. (This option is used to fetch not only a module's files but also those of the module's submodules, their submodules, and so on down the module hierarchy. For an example scenario of fetching a module hierarchy, see *A CPU Designer Gets a Module Hierarchy*.)

The **hcm get** operation places the ALU files in the work area directory Thomas specified with the **-path** option. (If Thomas does not specify the **-path** option, the operation uses his current work area directory.) If the path Thomas specified does not exist, the get operation creates it.

The value for the **-path** option becomes the base directory for the ALU module in Thomas's work area.

4. Thomas works on the ALU design, checking out files for edit the files that he wants to modify. To check out files for edit, he can use either:
 - The **hcm get -edit** operation to check out all of the files in the ALU module default configuration and lock them for edit
 - The DesignSync **co -lock** operation to check out individual files and lock them for edit

When he has finished the changes he wanted to make, Thomas puts the files back on the server with the **hcm put** operation. (For an example scenario, see [A Designer Puts Files of a Module Back on the Server.](#))

5. To view and work with other team members' work on the ALU files, Thomas regularly updates the ALU module in this work area. He changes directory to the base directory for the ALU module. Then he uses the **hcm get** operation again, but this time he does not need to specify the **-target** option. The HCM software stores the value for **-target** and recalls the information he specified when he first got the ALU module. And since Thomas' current directory is the location where he wants to fetch the ALU module, he does not need to specify the **-path** option.

```
stcl> cd /data/devel/users/Thomas/ALU
stcl> hcm get
```

The **hcm get** operation uses the **-replace** option by default. The operation updates Thomas' work area, making its contents match that of the ALU module on the SyncServer. The operation:

- Fetches new files from the server to Thomas' work area.
- Replaces existing files with newer versions, unless Thomas has modified the files in his work area.
- Updates hierarchical references to submodules, if any.

Since Thomas did not specify the **-recursive** option, the **hcm get** operation considers only the hierarchical references of the upper-level module. (The **get** operation is nonrecursive by default.) However, in this scenario, the ALU module has no submodules yet, so hierarchical references are not updated.

Related Topics

[hcm get Command](#)

[hcm showconfs Command](#)

[hcm showmods Command](#)

[co Command](#)

A Designer Puts Files of a Module Back on the Server

In the scenario An ALU Designer Gets Files of a Module, Thomas, a designer on the ALU team, fetched files of the ALU module's default configuration to his work area with the **hcm get** operation. In the course of his design work, he checked out several files for edit (with the DesignSync **co -lock** command) and modified them.

Thomas has now finished with his work on the ALU design for the day and is ready to check in the modified files to the SyncServer. He uses the **hcm put** operation. (**Note:** Only branch or selector configurations can be checked in with the put operation.)

1. Because the put operation does not check in new (unmanaged) files, Thomas uses the **hcm showstatus** command with the **-files** option. This option lists the status of the files of the ALU configuration in the work area as compared to the server. Using **hcm showstatus -files** lets Thomas identify new files and decide whether he needs to check them in.

```
%stclc
stcl> cd /data/devel/users/Thomas/ALU
stcl> hcm showstatus -files
```

```
Target:          sync://cpu.ABCo.com:2647/Projects/ALU
Base Directory: /data/devel/users/Thomas/ALU
```

No local or remote hierarchical references found for configuration.

Workspace Version -----	Configuration Version -----	Object Name -----
1.1 (Locally Modified) Unmanaged	1.1	alu.v
1.2 (Locally Modified)	1.2	reg8.v
1.5	1.5	aludoc/aluprojinfo.txt
		aludoc/aluprojlist.txt

Configuration status: Out-of-date

Summary: Out-of-date

The output from the **hcm showstatus** operation reminds Thomas that he has a new file (`reg8.v`) that he wants to include in the ALU module. He uses the DesignSync checkin (**ci**) operation to check in the file. **Note:** The checkin operation Thomas uses depends on the type of configuration he wants to add the files to. For an example scenario, see A Designer Adds Files to a Configuration.

2. Thomas uses the **hcm put** operation with the **-lock** option to check in all files he modified but lock them for edit so that he can continue to modify them:

```
% stcl  
stcl> cd /data/devel/users/Thomas/ALU  
stcl> hcm put -lock
```

Thomas needs to put back on the SyncServer only the files of the ALU module configuration, so he does not specify the **-recursive** option. (By default, the put operation is not recursive.)

Because Thomas does not specify the **-path** option, the operation uses his current work area directory.

The put operation:

- Checks in files that Thomas modified. These files include both files that he locked for edit and files that he did not lock for edit but that are the latest version on the branch. **Note:** The put operation does not check in files that are up-to-date.
 - Locks for edit the files checked in. This action is a result of specifying the **-lock** option.
 - Generates a RevisionControl note. Users receive email about the put operation if they have subscribed to email notification of RevisionControl notes on either the ALU module or the ALU configuration being checked in.
3. Thomas examines the output from the put operation to see if any of the checkin operations failed. (Messages appear at the end of the output for the operation.) If there are any failures, Thomas fixes the problems that caused them.
 4. When Thomas is satisfied that he has fixed all checkin problems, he uses the **hcm put** operation again. For this put operation, however, he does not specify the **-lock** option.

```
stcl> hcm put
```

The put operation:

- Checks in all modified files belonging to the ALU module's default configuration
- Removes the locks from all files belonging to the configuration, even those that were not locally modified (and therefore not checked in)

Note: Thomas can also use the DesignSync **cancel** command to remove locks that he has placed on unmodified files (with the **co -lock** or **ci -lock** command). This operation, in effect, performs an "un"checkout operation on the specified locked object.

If another user has locked the files (with the **hcm get -edit** operation), Thomas can use the DesignSync **unlock** command to unlock the files. However, the

unlock command typically is access controlled to allow only Synchronicity administrators to break other users' locks.

Related Topics

A Designer Adds Files to a Configuration

hcm put Command

hcm showstatus Command

cancel Command

unlock Command

A Designer Adds Files to a Configuration

Thomas, an ALU designer, has been working on the ALU design files contained in the ALU@Dev02 configuration. In the course of his design work, he has fetched the configuration to his work area and has created some new files. Now he wants to make the new (unmanaged) files part of the ALU@Dev02 configuration, so that the put operation checks the files in along with other files of the configuration.

Note: If Thomas does not know which configuration of the ALU module he has in his work area, he can use the **hcm showhrefs** operation with the **-path** option. (The information for Target: provides the configuration name.) For example:

```
% stcl
stcl> hcm showhrefs -path /data/devel/users/Thomas/ALU

Target: sync://alu.ABCo.com:2647/Projects/ALU@Dev02
Path:   /data/devel/users/Thomas/ALU
.
.
```

To add files to a configuration, Thomas first has to identify whether the configuration is a branch or selector configuration. **Note:** Synchronicity does not recommend adding files to an alias or a release.

Thomas uses the **hcm showconfs** operation to show information about the configurations of the ALU module:

```
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU

Configurations of module sync://alu.ABCo.com:2647/Projects/ALU
```

NAME	TYPE	OWNER	SELECTOR/ALIASED RELEASE
<Default>	Branch	Anne	Trunk:Latest
C1	Selector	Anne	Trunk:valid12.06.01
Dev02	Branch	Anne	Dev02:Latest

A Designer Adds Files to a Branch Configuration

Using the **hcm showconfs** operation, Thomas has identified the ALU@Dev02 as a branch configuration with a selector of `Dev02:Latest`.

To add the new files to that configuration, Thomas first identifies the files and then checks them in:

1. Thomas uses the DesignSync **ls** command to identify the new (unmanaged) files:

```
stcl> cd /data/devel/users/Thomas/ALU
stcl> ls -recursive -unmanaged
```

Note: A recursive **ls** operation lists files in all subdirectories of a given directory, regardless of which HCM configuration the files belong to. Users must use selectors to identify files belonging to each configuration.

2. He then uses the DesignSync checkin (**ci**) operation to check in the unmanaged files he wants to add to the configuration:

```
stcl> ci -new -comment "adding mult8 logic" mult8.gv
mult8.v
```

Note: It is not necessary for Thomas to use the **-branch** option. By default, the **ci** command uses the branch that the **hcm get** operation used to fetch the configuration.

The checkin operation checks the two files into the Dev02 branch, making them part of the ALU@Dev02 configuration. When Thomas fetches the configuration or puts it back, the get and put operations will operate on these two files along with the other files of the configuration.

Note: HCM does not follow symbolic links to files or folders. For more information, see Overview of Administration Tasks.

A Designer Adds Files to a Selector Configuration

Suppose Thomas had been working on the design files of the ALU@C1 module configuration. He has fetched the configuration to his work area and has modified some

of its files. In addition, he has created several new files he wants to add to the configuration.

Using the **hcm showconfs** operation, Thomas has identified ALU@C1 as a selector configuration created with the version tag `valid12.06.01`. To be part of a selector configuration, new files must have one of the selectors associated with the configuration. So Thomas checks in the new files to the appropriate branch and then tags them with the `valid12.06.01` tag:

1. Thomas uses the DesignSync **ls** command to identify the new (unmanaged) files:

```
stcl> cd /data/devel/users/Thomas/ALU
stcl> ls -recursive -unmanaged
```

Note: A recursive **ls** operation lists files in all subdirectories of a given directory, regardless of which HCM configuration the files belong to. Users must use selectors to identify files belonging to each configuration.

2. Thomas uses the DesignSync checkin (**ci**) operation to check in the unmanaged files. He uses the **-branch** option to ensure that the files are checked in on the branch appropriate for the ALU@C1 configuration (`Trunk`, in this example):

```
stcl> ci -new -comment "new files for mult8 logic" -branch
Trunk mult8.gv mult8.v
```

The checkin operation checks in the files to the `Trunk` branch on the SyncServer.

3. Next, he uses the DesignSync **tag** operation to tag the new files with the `valid12.06.01` version tag:

```
stcl> tag valid12.06.01 mult8.gv mult8.v
```

The tag operation adds the `valid12.06.01` version tag to the two files, making them part of the ALU@C1 configuration. Subsequent put operations of the ALU@C1 configuration will check in these new files (if they have been modified).

Note: HCM does not follow symbolic links to files or folders. For more information, see Overview of Administration Tasks.

Thomas Moves the Version Tag of Files Added to a Selector Configuration

When it is used on a selector configuration, the put operation checks in all modified files. However, for files added to the configuration with a version tag (as shown in A Designer Adds Files to a Selector Configuration), the put operation does not move the version tag to the files' new versions. Users must manually move the tag. For example, each time Thomas puts the ALU@C1 configuration back on the SyncServer, he must

move the `valid12.06.01` tag to the new versions of the `mult8.gv` and `mult8.v` files.

To update the version tag of files added to a selector configuration:

1. Thomas identifies selector configurations that the put operation checked in. (These configurations are listed in the summary at the end of the output for the put operation.)
2. He next uses the **hcm showconfs** operation to show the selectors used to create the selector configurations. For example:

```
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU
```

```
Configurations of module
sync://alu.ABCo.com:2647/Projects/ALU
```

NAME	TYPE	OWNER	SELECTOR/ALIASED	RELEASE
<Default>	Branch	Anne	Trunk:Latest	
C1	Selector	Anne	valid12.06.01	
Dev02	Branch	Anne	Dev02:Latest	

3. Thomas uses the DesignSync **ls** command to identify the files that need the `valid12.06.01` version tag moved to their new versions.

```
stcl> ls -recursive -report SNGHR
```

The operation shows each file's name, version and branch tags, and current version.

4. For each file that needs the version tag moved to its new version, Thomas uses the DesignSync **tag** operation to move the tag. For example, suppose the put operation checked in `mult8.gv`, creating version 1.3. Thomas would use the **tag -replace** command to move the `valid12.06.01` tag to the new version of the `mult8.gv` file:

```
stcl> tag -replace valid12.06.01 mult8.gv -version 1.3
```

Notes:

- o Thomas does not have to include the **-version** option. The tag operation uses the file versions in the work area to determine which versions to tag on the server. Because Thomas used the put operation earlier to check in modified files in his work area, version 1.3 of `mult8.gv` now exists on

- the server as well as in his work area. From Thomas's work area, the tag operation determines that it should tag `mult8.gv;1.3` on the server.
- o Thomas can also use the DesignSync Tag dialog box to move the version tag. For information, see Tagging Versions and Branches in DesignSync Help.

The new 1.3 version of `mult8.gv` is included in the ALU@C1 configuration.

Note: Each time that Thomas modifies these files and then uses the put operation to check in the configuration, he must move the version tag to the new version of each file.

Related Topics

`hcm put` Command

`hcm showconfs` Command

`ci` Command

`ls` Command

selectors

`tag` Command

`url selector` Command

A Designer Removes Files from a Configuration

Thomas, an ALU designer, has been working on the ALU design files, fetching the configuration to his work area, creating and revising files, and putting the configuration back on the server. In the course of his development work, Thomas decides to combine two source files (`AluAddEx.v` and `AluDelEx.v`) into one file (`AluEx.v`). He creates the new file and adds it to the ALU configuration. (For information on adding files to a configuration, see A Designer Adds Files to a Configuration.) Then, since the two original files are no longer needed for the design, Thomas decides to remove them from the configuration.

Note: If Thomas does not know which configuration of the ALU module he has in his work area, he can use the `hcm showhrefs` operation with the `-path` option. (The information for Target: provides the configuration name.) For example:

```
% stclc
stcl> hcm showhrefs -path /data/devel/users/Thomas/ALU
```

```
Target: sync://alu.ABCo.com:2647/Projects/ALU@Dev02
```



```
Path: /data/devel/users/Thomas/ALU
.
```

To remove files from a configuration, Thomas first has to identify whether the configuration is a branch or selector configuration. (HCM software does not allow removal of files from a release or an alias.) **Note:** To remove files from a default configuration, Thomas would follow the same steps as for removing files from a branch configuration.

Thomas uses the **hcm showconfs** operation to show information about the configurations of the ALU module:

```
% stcl
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU
```

Configurations of module sync://alu.ABCo.com:2647/Projects/ALU

NAME	TYPE	OWNER	SELECTOR/ALIASED	RELEASE
<Default>	Branch	Anne	Trunk:Latest	
C1	Selector	Anne	Trunk:valid12.06.01	
Dev02	Branch	Anne	Dev02:Latest	

A Designer Removes Files from a Branch Configuration

Using the **hcm showconfs** operation, Thomas has identified ALU@Dev02 as a branch configuration with a selector of Dev02:Latest. To remove files from that configuration, Thomas retires each file's Dev02 branch.

Thomas uses the DesignSync **retire** command to retire the Dev02 branch of AluAddEx.v and AluDelEx.v:

```
stcl> retire AluAddEx.v AluDelEx.v
```

Note: Thomas can also use **Revision Control=>Retire** in DesignSync List View to retire each file's branch. For information, see Retiring Branches in DesignSync Help.

The retire operation:

- Retires the Dev02 branch of AluAddEx.v and AluDelEx.v
- Deletes the local copies of AluAddEx.v and AluDelEx.v unless Thomas has modified them or specified the **-keep** option of the retire operation

When Thomas uses the **hcm get** operation to fetch the files of the ALU@Dev02 configuration to his work area, the get operation does not fetch `AluAddEx.v` and `AluDelEx.v`. (Because each file's `Dev02` branch is retired, the files are no longer part of the configuration.)

A Designer Removes Files from a Selector Configuration

Suppose Thomas had been working on the design files of the ALU@C1 module configuration. He has fetched the configuration to his work area and has decided he wants to remove two of the files from it.

Using the **hcm showconfs** operation, Thomas has identified ALU@C1 as a selector configuration created with the version tag `valid12.06.01`. To be part of a selector configuration, files must have one of the selectors (usually a version tag) associated with the configuration. To remove files from a selector configuration, the configuration's version tag must be deleted from each of the files.

To remove the files from the ALU@C1 configuration, Thomas must delete each file's `valid12.06.01` version tag:

1. Thomas first uses the DesignSync **ls** command to make sure that all of the files in the configuration are up-to-date and to view the version tag and version number of the files he wants to remove from the configuration (`file4.v` and `file2.v`):

```
stcl> cd /data/devel/users/Thomas/ALU
stcl> ls -recursive -report SNGHR
```

The operation shows each file's name, version and branch tags, and current version.

Note: A recursive **ls** operation lists files in all subdirectories of a given directory, regardless of which HCM configuration the files belong to.

2. He then uses the DesignSync **tag -delete** command to remove the `valid12.06.01` version tag from the two files:

```
stcl> tag -delete valid12.06.01 file4.v file2.v
```

Note: Thomas can also use the DesignSync Tag dialog box to delete the version tag. For information, see Tagging Versions and Branches in DesignSync Help.

The tag operation deletes the `valid12.06.01` tag from `file4.v` and `file2.v` on the server. **Note:** The tag operation does not delete the files from the server or from Thomas's work area.

When Thomas uses the **hcm get** operation to fetch the files of the ALU@C1 configuration to his work area, the get operation does not fetch `file4.v` or `file2.v`. (Because they do not have the `valid12.06.01` version tag, the files are no longer part of the configuration.) In addition, if Thomas has not modified `file4.v` or `file2.v`, the get operation deletes them from his work area.

Related Topics

`hcm showconfs` Command

`retire` Command

`tag` Command

The CPU Team References the ALU Module

The CPU team is developing RTL code on its development SyncServer. In the course of development, team members decide they need to incorporate into their design a new release of the ALU module, ALU@R1, making it a submodule of their CPU module.

To make ALU@R1 a submodule of CPU, the CPU team leader (Robert) creates a hierarchical reference:

1. Robert uses the **hcm addhref** operation to create the hierarchical reference between the CPU module and the ALU@R1 module. (Adding this reference makes a connection between the CPU module and ALU@R1, in effect adding ALU@R1 as a submodule to CPU in the CPU module hierarchy.)

```
% stclc
stcl> hcm addhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://alu.ABCo.com:2647/Projects/ALU@R1 -relpath ALU
```

Robert identifies ALU@R1 as the submodule to add by identifying both ends of the connection with the **-fromtarget** and **-totarget** options (required by **hcm addhref**).

Because Robert specified only the CPU module (without a configuration) as the **-fromtarget**, the **hcm addhref** operation creates a reference from the CPU module's default configuration to the ALU module's R1 configuration.

The **hcm addhref** operation uses the **-relpath** value Robert specified as the path from the base directory of the CPU module to that of the ALU module. (This path does not need to exist; when users get the module to their work areas, the **hcm**

get operation creates directories and subdirectories as necessary.) If Robert does not specify the **-relpath** option, the operation uses the **-totarget** module name as the name of the submodule's base directory and places it directly below the upper-level module's base directory. (In this example, the result would be the same as the value Robert specified for **-relpath**.)

The **hcm addhref** operation creates the reference and stores the information with the CPU module default configuration on the SyncServer.

To change or remove the reference, Robert would use the **hcm rmhref** operation. For an example scenario, see The CPU Team Changes a Reference to a New ALU Release.

2. To view the CPU module's references he just created on the SyncServer and any others from the CPU module, Robert uses the **hcm showhrefs** operation:

```
stcl> hcm showhrefs -target sync://cpu.ABco.com:2647/Projects/CPU
```

```
Target:      sync://cpu.ABco.com:2647/Projects/CPU
```

REFERENCE PATH	URL	RELATIVE
-------------------	-----	----------

ALU@R1	sync://alu.ABco.com:2647/Projects/ALU@R1	ALU
CACHE@R4	sync://cpu.ABco.com:2647/Projects/CACHE@R4	CACHE

Note: To show the whole hierarchy of the CPU module, its submodules, their submodules, and so on, Robert could have used the **hcm showhrefs -recursive** operation. See the **hcm showhrefs** command in the HCM Command Reference for more information.

3. After adding the reference, Robert sends email to the CPU team members, reminding them to consult Guidelines for HCM Use when working with the HCM module configurations.

Related Topics

[hcm addhref Command](#)

[hcm rmhref Command](#)

[hcm showhrefs Command](#)

[The CPU Team Changes a Reference to a New ALU Release](#)

The ALU Team Makes a Release Available

(Release of a Single Module)

The ALU team's RTL design has reached a level of maturity such that the team decides it is time to make the design available to other teams. To make the design available, Anne, the ALU team leader, will use HCM to create a release of the ALU module's default configuration.

Note: The team is releasing the default configuration of the ALU module, which at this point contains no references to submodules (hierarchical references), only files in ALU vault directory and subdirectories. Anne has already fetched the files of the ALU module's default configuration to her work area with the **hcm get** operation.

The **hcm release** operation is a client-side operation, so Anne performs the operation from her work area:

1. The **hcm release** operation determines the contents of the release from file versions fetched to the work area. Locally modified or unmanaged files are not included in the release.

To show the status of the ALU module's files in her work area, Anne uses the DesignSync **hcm showstatus -files** option. This command compares the configuration's files in her work area to those on the server and tells her such information as whether each file is up-to-date. Using **hcm showstatus -files** also allows Anne to spot locally modified or unmanaged files in her work area that she may want in the release. (She can use the DesignSync **ci -new** operation to check in unmanaged files and the **hcm put** operation to check in modified files. For information, see A Designer Adds Files to a Configuration and A Designer Puts Files of a Module Back on the Server.)

```
% stclc
stcl> cd /data/devel/users/Anne/ALU
stcl> hcm showstatus -files

Target:          sync://cpu.ABCo.com:2647/Projects/ALU
Base Directory:  /dev/users/Marie/ALU

No local or remote hierarchical references found for configuration.

Workspace          Configuration      Object
Version            Version           Name
-----
1.1 (Locally Modified)  1.1              alu.v
1.3                  1.3              reg8.v
Unmanaged          aludoc/aluprojinfo.txt
1.8                  1.8              aludoc/aluprojlist.txt

Configuration status: Out-of-date

Summary: Out-of-date
```

From the output of the `showstatus` operation, Anne sees that her work area has an unmanaged file and a modified file she wants to include in the ALU release. Because the files in the work area do not match those of that same configurations on the server, the output from `hcm showstatus -files` shows `Configuration status: Out-of-date` for the configuration. (When you use the `-files` option, the `showstatus` operation determines each configuration's status from both the status of its files and its hierarchical references. If either is out-of-date with the configuration on the server, the status of the configuration is out-of-date.)

2. She uses the `hcm release` operation to create the ALU release and give it the name R1:

```
stcl> pwd
/data/devel/users/Anne/ALU
stcl> hcm release -name R1 -norecursive -description
"Initial release of ALU"
```

Because Anne did not specify the `-path` option, the `hcm release` operation uses her current work area directory. The operation determines the contents of `ALU@R1` from Anne's ALU work area and then creates the release on the SyncServer.

Because Anne specified the `-norecursive` option, the release operation creates a release of the ALU module only; the operation does not release submodules. (At this point, the ALU module's default configuration contains no submodules.)

The release operation includes the description (specified with the `-description` option) in the ProjectSync project. (If the `-description` option is not specified, the release operation provides a default description of "HCM release".) Team members can view the description in ProjectSync and in the output of the `hcm showconfs -report script` operation.

Other Teams Learn of New Releases

Other design teams at ABCo find that there is a new release of the ALU module. Teams can then include the `ALU@R1` module configuration in their design hierarchies by creating a hierarchical reference to it. For an example scenario of creating a hierarchical reference, see [The CPU Team References the ALU Module](#).

There are several ways that a team can find out about module releases of other teams:

- By subscribing for email notification of RevisionControl notes for HCM operations on the ALU module. For an example scenario, see [Robert Subscribes to RevisionControl Notes on HCM Operations](#).

- By using the **hcm showmods** operation to show modules on a server and **hcm showconfs** to show configurations of a module. For an example scenario, see *An ALU Designer Gets Files of a Module*.

Related Topics

hcm release Command

hcm showconfs Command

hcm showmods Command

hcm showstatus Command

The CPU Team Subscribes to Email on a Hierarchy

The CPU design is nearing its tape out date; any problem with the design at this point must be scrutinized. Design work is focused on the CPU@C21 module configuration, which contains the CPU module hierarchy. As CPU team leader, Robert wants to know when a defect is filed against any part of the CPU@C21 configuration, including its submodules. To be notified of these defects, Robert uses ProjectSync's **Advanced Subscription** panel to subscribe for email about defects logged against the CPU module hierarchy. This type of subscription is called a **hierarchical subscription**.

Note: This scenario assumes that:

- Robert has valid login accounts on all SyncServers with which the hierarchical subscription will communicate. For example, if Robert subscribes for notes on a module and all its submodules, he must have a login account on each submodule server.
- Robert or the Synchronicity administrator has mapped note types to ensure that hierarchical subscriptions to notes used on the cpu server include notes on submodule servers using different note types. This note type mapping must be done before performing hierarchical subscriptions or hierarchical queries. See *Mapping Note Types* for more information.
- The hcmNoteAttach trigger is installed on the SyncServer for the CPU module as well as all SyncServers hosting configurations referenced in the CPU module hierarchy. See *Adding the hcmNoteAttach Trigger* for information.

To subscribe for email notification:

1. In his browser, Robert enters the URL of the ProjectSync server for the CPU module (the same as the CPU development server).
2. From the User Profiles menu, he selects **Email Subscriptions** and then clicks **Advanced Subscriptions**. ProjectSync displays the **Add New Subscriptions for ...** panel.

Note: Hierarchical subscriptions (subscriptions to email about a module hierarchy) can be done only through the **Advanced Subscriptions** panel.

3. In the Advanced Subscriptions part of the panel, Robert selects **SyncDefect** from the Note Type pull-down menu.
4. Robert wants to receive email about all defects logged against the CPU@C21 module configuration, so he does not enter any value in the **Property Filter** field.

ProjectSync filters out, or excludes, items from email notification based on values specified in this field. For example, if Robert wanted to receive email notification only for CPU module defects with stopper priority, he could specify that property value in this field. (For information on specifying values in this field, you can view ProjectSync help information by clicking the **Help** button next to the **Property Filter** field.)

5. From the **Object Filter** pull-down menu, Robert selects CPU. ProjectSync displays the URL `sync:///Projects/CPU` in the **Object Filter** field.

Note: The CPU module is displayed below `/Projects` because it is a ProjectSync object. (ProjectSync displays all ProjectSync objects below `/Projects`.)

6. To the `sync:///Projects/CPU` URL, Robert appends the configuration name as `@C21`.

Robert can also browse to find the configuration he wants by selecting **Browse...** from the **Object Filter** pull-down menu. This action displays the Browse Server window, from which he can navigate ProjectSync projects, selecting projects or configurations. If Robert has already selected a project (as he did in step 5), the browser is positioned at that project, displaying its configurations for selection.

7. Since Robert wants to be notified of defects not only on the upper-level CPU module but also all of its submodules, he selects **This object and all levels below** from the **Scope** field.
8. When he has completed his subscriptions, Robert clicks **Submit**.

ProjectSync sets up the email subscription and displays the Success panel. The panel lists the email subscription for the SyncDefect note type on CPU@C21 and subscriptions added for referenced submodules.

Note: Email subscriptions do not apply to referenced IP Gear deliverables. However, IP Gear users can subscribe to information related to deliverables on the IP Gear server.

Robert Deletes an Email Subscription

Robert is taking an extended vacation and does not want to receive email about the CPU@C21 configuration while he is away.

To delete the email subscriptions to the CPU module and all its submodules, Robert again selects **Email Subscriptions**. The panel lists all of his subscriptions, including the CPU@C21 configuration. Robert clicks **Delete** for CPU@C21. Then he clicks **Submit** to have the change take effect. ProjectSync deletes his subscription to not only CPU@C21 but also its submodules, their submodules, and so on down the module hierarchy.

Robert Subscribes to RevisionControl Notes on HCM Operations

During design development, module hierarchies can change as submodules are added and removed or aliases are changed. HCM users, particularly module owners such as project leaders, need to know of such changes so they can update their module configurations or determine the effects of the changes on their work areas. To receive notification of such changes, they can use ProjectSync to subscribe to email notification of RevisionControl notes for HCM commands.

For example, suppose the CPU@C21 module configuration has a hierarchical reference to the submodule ALU@CTO19. Robert, the CPU team leader, wants to know of configuration changes at all levels in the CPU@C21 hierarchy. Using ProjectSync, he subscribes to email notification of RevisionControl notes for **hcm rmconf** and **hcm mkconf** operations on the CPU@C21 configuration and all submodules below it in the hierarchy.

Note: These steps assume that Robert or the Synchronicity administrator has set up email notification of HCM Revision Control notes on the alu server. For information, see Setting Up Notification of HCM RevisionControl Notes.

1. Robert follows the steps for subscribing for email notification, clicking **Advanced Subscriptions**.
2. At the **Add New Subscriptions for...** panel, for **Note Type**, he chooses **RevisionControl**.
3. For **Property Filter**, he types the string `Tag=` and the HCM command for which he wants notification. For example, `Tag=rmconf`.

Using this string causes ProjectSync to send email only for RevisionControl notes for the **hcm rmconf** operation and no others.

4. From the **Object Filter** pull-down menu, Robert selects CPU. ProjectSync displays the URL `sync:///Projects/CPU` in the **Object Filter** field.
5. To the `sync:///Projects/CPU` URL, Robert appends the configuration name as `@C21`.

Robert can also browse to find the configuration he wants by selecting **Browse...** from the **Object Filter** pull-down menu. This action displays the Browse Server

window, from which he can navigate ProjectSync projects, selecting projects or configurations. If Robert has already selected a project (as he did in step 4), the browser is positioned at that project, displaying its configurations for selection.

6. In the **Scope** field, he selects **This object and all levels below**.
7. He clicks **Submit** to have the subscription take effect.

ProjectSync sets up the email subscription and displays the Success panel. The panel lists the email subscription for the RevisionControl note type on CPU@C21 and subscriptions added for referenced submodules.

8. Robert repeats the steps, this time subscribing to email notification of RevisionControl notes for **hcm mkconf** operation on the CPU@C21 configuration.

A few weeks later, when the ALU team removes the ALU@CTO19 configuration from the alu server, Robert receives email notification of the operation. He can then remove the CPU module's reference to the ALU@CTO19 module and, if appropriate, replace it with another reference.

Robert Updates Email Subscriptions

Module hierarchies can change after a user has subscribed for email notifications. For example, suppose that the CPU module has a hierarchical reference to the submodule ALU@GOLDEN and that ALU@GOLDEN is an alias pointing to the ALU@R1 release. As development progresses, the ALU team creates a new release and then changes the ALU@GOLDEN alias to point to a different release, ALU@R2.

Or suppose that Robert (the CPU team leader) subscribes for email notification of defects on the CPU@C21 configuration and all of its submodules (a hierarchical subscription). Then some time later, one of the ALU submodule's hierarchical references changes from FPU@R1 to FPU@R2.

In each case, Robert must manually update his email subscriptions to reflect the change.

Manual Update of Email Subscriptions When an Alias Changes

To manage subscriptions when aliases change, Robert takes the following steps.

1. If he has not done so, Robert uses ProjectSync to subscribe to RevisionControl notes for the **hcm mkalias** operation. Taking this action ensures that he will always be notified when an alias changes. For information, see Robert Subscribes to RevisionControl Notes on HCM Operations.
2. When he receives email that an **hcm mkalias** operation has taken place on the ALU@GOLDEN object, Robert uses ProjectSync to delete his subscription to the

release to which ALU@GOLDEN previously pointed (in this case ALU@R1). (The email generated from an **hcm mkalias** command includes both the old and the new alias.)

3. Robert then subscribes for email on the ALU@GOLDEN alias again. The subscription now includes notes for the ALU@R2 submodule, to which the ALU@GOLDEN alias currently points.

Manual Update of Email Subscriptions When a Hierarchical Reference Changes

To manage subscriptions when hierarchical references change, Robert takes the following steps.

1. If he has not done so, Robert uses ProjectSync to subscribe to RevisionControl notes for the **hcm addhref** and **hcm rmhref** operations. Taking this action ensures that he will always be notified when a hierarchical reference changes. For information, see Robert Subscribes to RevisionControl Notes on HCM Operations.
2. Robert receives email notifications that an **hcm rmhref** removed ALU@C1's reference to FPU@R1 and a subsequent **hcm addhref** added a reference to FPU@R2. Robert resubscribes for notes on the CPU@C21 module hierarchy (a hierarchical subscription). After Robert resubscribes for notes on the CPU@C21, the subscription includes notes for the FPU@R2 submodule.

Notes:

- The email notification generated from the **hcm rmhref** and **hcm addhref** operations includes the previous and current hierarchical references.
- Although resubscribing to the CPU@C21 module hierarchy includes a subscription to notes on the new FPU@R2 submodule, it does not delete subscriptions to notes on submodules that the hierarchy no longer references (for example, FPU@R1). Robert must manually delete this subscription.

The CPU Team Leader Queries for Defects

Robert, the CPU team leader, uses a ProjectSync note type on the cpu server to track defects on the CPU module. He regularly uses ProjectSync to query for open defects assigned to CPU.

Robert has decided that he wants to view not only defects logged against the CPU module but also against all submodules referenced by CPU. Fortunately, with HCM software installed, Robert can perform a **hierarchical query**, a ProjectSync query that operates not only on the CPU module but all of its submodules, all of their submodules, and so on down the hierarchy.

Note: This scenario assumes that Robert or the Synchronicity administrator has mapped note types such that a query initiated on a module on one server can successfully query submodule servers that use different note types. See Mapping Note Types for more information.

Robert constructs the hierarchical query:

1. He invokes ProjectSync and from the Queries menu, he selects **Standard**.
2. From the Query Notes panel, he selects **SyncDefect**, which is the note type used on the cpu server and the note type he wants to query. **Note:** Robert can select any note type for his query, including the RevisionControl and custom note types.
3. On the Query: SyncDefect form, Robert selects the CPU module in the **Project** field.
4. Robert wants to query the CPU module's default configuration, so he does not select a configuration from the **Configuration** field.
5. Robert now uses the **Scope** field to specify how far down the query should extend into the CPU module hierarchy. The selections of the **Scope** pull-down menu are:
 - This object only
 - This object and one level below
 - This object and all levels below

For example, the CPU module's default configuration has submodules ALU, IOSTAT, and CACHE, and some of those submodules have submodules, and so on. For a query on the default configuration of CPU:

This selection...	Queries for notes logged against...
This object only	The CPU module only
This object and one level below	The CPU module and its submodules (ALU, IOSTAT, and CACHE) only. The query does not extend to submodules of ALU, IOSTAT or CACHE.
This object and all levels below	The CPU module, its submodules, their submodules, and so on down through the hierarchy

Because he wants to view defects on all modules on all levels of the CPU module hierarchy, Robert selects **This object and all levels below**.

Note: The behavior of a query on a module hierarchy depends on the values specified in the **Project**, **Configuration**, and **Scope** fields. For information on how these values affect a query, see Query.

6. Robert chooses **Open** as the State and clicks **Submit**.

The query mechanism queries the CPU module's default configuration (as specified by the **Configuration** field value) and all levels of the hierarchy below

(as specified by the **Scope** field). At each level in the hierarchy, the query displays only notes that are attached to the module's default configuration.

ProjectSync displays the results panel, which contains:

- Query criteria. This information includes:
 - The note type queried
 - The name of the project queried (in the case of a hierarchical query, the module queried)
 - The date the query was submitted
- Matches to the hierarchical query criteria. The panel displays the following information about each match:

Module	The module name, which is also a hyperlink to the module's Project, Configuration, or directory Data Sheet, or the Deliverable Home Page for an IP Gear deliverable. Submodules are indented under the upper-level module.
Server	The server's hostname, which is also a hyperlink to the server's ProjectSync Welcome page
ID	The note identifier, which is also a hyperlink to the note itself
Title	The title of the note

- **Notes:**
 - The fields displayed or downloaded will vary according to the report format options you selected. However, the Module and Server columns are always displayed or downloaded for a hierarchical query.
 - For a hierarchical query to display information about submodules that are IP Gear deliverables (such as IOSTAT in the example), the note type of the upper-level module's SyncServer must be mapped to a note type on the referenced IP Gear server, such as the CustTicket note type (on which the IP Gear Ticket is based). For information, see Mapping Note Types.

The CPU Team References an IP Gear Deliverable

Members of the CPU team decide that for a part of their design work, they want to reuse a block developed by the IOSTAT team at the IOCo company. No one on the CPU team knows on which server the IOSTAT block was originally developed. However, they do know that the IOSTAT team used IP Gear to publish the design for reuse. They also know that the IP Gear server is `sync://ipgsrvr1.IOCo.com:2647`.

Robert, the CPU team leader, locates the IOSTAT block and includes it as a submodule of the CPU module by creating a hierarchical reference to it:

1. Using the search features of IP Gear, Robert locates the IOSTAT block and determines that it is deliverable number 1028. The deliverable number is

important because it must be specified when the hierarchical reference is created.

2. Robert then uses the **hcm addhref** operation to create the hierarchical reference from the CPU module to IP Gear deliverable 1028. For the **-totarget** value, Robert specifies a URL of the form

```
sync://<IP_Gear_server_host>:<port>/Deliverable/<Deliverable number>.
```

```
% stclc
stcl> hcm addhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028 -relpath
IOSTAT
```

The **hcm addhref** operation creates the reference from CPU module's default configuration to the IOSTAT block on the IP Gear server. The reference information is stored with the CPU module default configuration on the SyncServer.

Note: The **-relpath** option is not required. If Robert did not specify the **-relpath** option, the addhref operation would use the **-totarget** module name as the name of the submodule's base directory and place it directly below the upper-level module's base directory. In this example, the addhref operation would use 1028 as the **-relpath** value.

When Robert or any other CPU team member gets the CPU module hierarchy (with **hcm get -recursive**) to his work area, the operation:

- Creates the IOSTAT subdirectory, if necessary.
- Fetches deliverable 1028's design files to the subdirectory.
- 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.)
- Creates or updates the Usage note on the IP Gear server and registers Robert 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 Robert desires to be registered for updates, Robert can still register a Usage note in IP Gear manually.)

Related Topics

hcm addhref Command

hcm get Command

The CPU Team Changes a Reference to a New ALU Release

The ALU team has created another release of the ALU block. Because the CPU team leader (Robert) has subscribed to notification of events for ALU, he is notified of the new release.

Currently, the CPU module has a submodule, ALU, which is the ALU team's R1 release (ALU@R1). The CPU team decides that it is time to remove the ALU@R1 submodule from the CPU module's hierarchy and replace it with the new ALU release (ALU@R2) .

To change one CPU submodule for another, Robert, the CPU team leader, removes the CPU module's hierarchical reference to ALU@R1 and then adds a new reference from CPU toALU@R2:

1. Robert uses the **hcm showconfs** operation to display a list of available configurations for the ALU module on the SyncServer:

```
% stcl
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU

Configurations of module
sync://alu.ABCo.com:2647/Projects/ALU
NAME          TYPE          OWNER  SELECTOR/ALIASED RELEASE
-----
<Default>    Branch       Anne   Trunk:Latest
C1            Selector    Anne   valid12.06.01
R1            Release     Anne
R2            Release     Anne
```

2. He uses the **hcm showhrefs** on the CPU module to display its hierarchical references and choose the one to remove:

```
stcl> hcm showhrefs -target sync://cpu.ABCo.com:2647/Projects/CPU
Target:          sync://cpu.ABCo.com:2647/Projects/CPU
REFERENCE        URL                                     RELATIVE
PATH
-----
----
ALU@R1           sync://alu.ABCo.com:2647/Projects/ALU@R1    ALU
CACHE@R4         sync://cpu.ABCo.com:2647/Projects/CACHE@R4  CACHE
Deliverable/1028 sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028 IOSTAT
```

3. Robert uses the **hcm rmhref** operation to remove the hierarchical reference between the CPU module's default configuration (the **-fromtarget**) and the ALU@R1 configuration (the **-totarget**). (Removing this reference removes the connection between the CPU module and ALU@R1, in effect removing ALU@R1 from the CPU module hierarchy.)

```
stcl> hcm rmhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://alu.ABCo.com:2647/Projects/ALU@R1 -relpath ALU
```

Robert further identifies the ALU@R1 submodule by specifying `ALU` as the **-relpath** option, which identifies the relative path from the base directory of the CPU module to that of the ALU@R1 submodule. If the ALU@R1 submodule is referenced multiple times, specifying the relative path helps identify it as the particular submodule to remove. If no **-relpath** option is specified, all of the references to the ALU@R1 submodule are deleted.

The **hcm rmhref** operation removes the reference from the CPU module's default configuration on the SyncServer.

Note: The **-fromtarget**, **-totarget**, and **-relpath** options are not all required by the **hcm rmhref** operation. See the **hcm rmhref** command for information.

4. Robert uses the **hcm addhref** operation to create a new reference from the CPU module's default configuration to the ALU@R2 configuration:

```
stcl> hcm addhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://alu.ABCo.com:2647/Projects/ALU@R2 -relpath ALU
```

Note: The **-relpath** option is not required. If Robert did not specify the **-relpath** option, the **addhref** operation would use the **-totarget** module name as the name of the submodule's base directory and place it directly below the upper-level module's base directory. (In this example, the result would be the same as the specified value for **-relpath**.)

The **hcm addhref** operation creates the reference and stores the information with the CPU module default configuration on the SyncServer.

When CPU team members find out about the change in hierarchical references, they update their work areas with the **hcm get** operation. For an example scenario, see [A CPU Designer Gets a Module Hierarchy](#).

Related Topics

[hcm addhref Command](#)

[hcm rmhref Command](#)

[hcm showconfs Command](#)

[hcm showhrefs Command](#)

A CPU Designer Gets a Module Hierarchy

Marie is a designer who has just joined the CPU team. The CPU team leader has given her the following information:

- The CPU team is using the SOC Developer Suite (DesignSync V3.3 and HCM). The vault location of the CPU design is:
`sync://cpu.ABCo.com:2647/Projects/CPU`
- Her account is `/dev/users/Marie`

Marie decides to get an overall idea of the scope of the CPU design, look at the data organization, and see what files and directories she'll be using. She uses the **hcm get -recursive** operation to get the CPU design hierarchy to her work area for the first time. (**Note:** This scenario shows fetching from the HCM server. For a scenario that shows fetching from the module cache, see *Designers Use the Module Cache.*)

1. Marie invokes the `stcl` command shell and uses the **hcm get -recursive** operation to get all of the files for the CPU module and its submodules to her work area:

```
% stcl
stcl> hcm get -recursive -target
sync://cpu.ABCo.com:2647/Projects/CPU -path
/dev/users/Marie/CPU
```

Because Marie specified the CPU module without a configuration for the **-target** option, the **hcm get** operation uses the CPU module's default configuration.

The **get -recursive** operation:

- Fetches all of the files in the CPU module to the work area directory specified with the **-path** option. (If Marie does not specify the **-path** option, the operation uses her current work area directory.) If the path Marie specified does not exist, the get operation creates it.
 - Follows CPU's hierarchical reference to each of its submodules (ALU, CACHE, and IOSTAT). For each submodule, the operation creates a subdirectory and fetches the submodule's files to it. (The location of this subdirectory in Marie's work area is determined by the relative path specified when the hierarchical reference was created.)
 - Continues following hierarchical references down through each level of the module hierarchy and fetching submodules' files until all files of the hierarchy are fetched.
2. Marie works on her part of the CPU design, revising existing files and creating new ones. When she has finished her work, she puts the hierarchy back on the SyncServer. See *A Designer Puts a Module Hierarchy Back on the Server* for information.

Marie Updates Her Work Area

Before she starts her work each morning, Marie uses the **hcm get** operation to update her work area with changes to the entire CPU module hierarchy (the CPU module and all its submodules).

1. Marie changes directory to the base directory for the CPU module configuration and invokes the stcl command shell:

```
stcl> cd /dev/users/Marie/CPU
```

2. She uses the **hcm get** operation to fetch files that have been modified since the last **hcm get** operation:

```
stcl> hcm get -recursive -incremental
```

Marie does not need to specify the **-target** option because the HCM software stored the information she specified when she first got the CPU module. She does not need to specify the **-path** option; by default the get operation uses her current work area directory, which is the one she wants.

The **-recursive** option causes the get operation to update not only the CPU module but all levels of the module hierarchy.

The **-incremental** option updates her work area directories only if their vault folders on the server have changed since she last got the module to her work area. Using this option is the fastest way to update.

By default, the **hcm get** operation applies the **-replace** option. This option removes files in the work area that do not match the contents of the updated modules on the server, unless the files are locally modified.

For example, suppose that on the CPU module on the server, two files were deleted and one hierarchical reference was changed from the ALU@R2 configuration to ALU@R3. When Marie uses **hcm get** (which applies **-replace** by default), the operation:

- Deletes the two files from Marie's work area, if she has not modified them. If she has modified the files but does not want to keep the changes, she can use the **-force** option to delete the modified files.
- Removes all unmodified files of ALU@R2 from the ALU directory of Marie's work area and then fetches files of the ALU@R3 configuration to that same directory. (This action takes place because the relative path for the new reference is the same as the relative path for the old reference.)

Related Topics

hcm get Command

hcm showstatus Command

ls Command

A Designer Puts a Module Hierarchy Back on the Server

In the scenario A CPU Designer Gets a Module Hierarchy, Marie, a designer on the CPU team, fetched files of the CPU module hierarchy to her work area with the **hcm get -recursive** operation. In the course of her design work, Marie modified files in the CPU module and created some new files. She also modified some files in the MEM submodule.

Marie has now finished with her work on the CPU design and is ready to check in the files she worked on to the SyncServer. She uses the **hcm put -recursive** operation:

1. To determine what configurations to check in, the put operation uses local data about the module hierarchy as it exists in a user's work area. For this reason, Marie uses the **hcm showstatus -recursive** operation to compare the hierarchical references of the CPU module hierarchy in her work area with those of the CPU hierarchy on the server. The put operation also does not check in new (unmanaged) files, so Marie uses the **-files** option with the **hcm showstatus** command. This option lists the status of the files of the CPU configuration in the work area as compared to the server. Using **hcm showstatus -files** lets Marie identify those files and decide whether she needs to check them in.

```
% stcl
stcl> cd /dev/users/Marie/CPU
stcl> hcm showstatus -files -recursive
Target:          sync://cpu.ABCo.com:2647/Projects/CPU
Base Directory: /dev/users/Marie/CPU
```

STATUS	HREF	RELATIVE PATH
Up-to-date	sync://alu.ABCo.com:2647/Projects/ALU@R2	ALU
Up-to-date	sync://cpu.ABCo.com:2647/Projects/CACHE@R4	CACHE
Up-to-date	sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028	IOSTAT
Up-to-date	sync://cpu.ABCo.com:2647/Projects/MEM	MEM

The status of the following hrefs will not be individually reviewed:

HREF	REASON
sync://alu.ABCo.com:2647/Projects/ALU@R2	release
sync://alu.ABCo.com:2647/Projects/CACHE@R4	release
sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028	IP Gear deliverable

Workspace Version	Configuration Version	Object Name
-------------------	-----------------------	-------------

```

-----
1.1 (Locally Modified) 1.1          cpu.v
1.3                    1.3          cpustat.v

Configuration status: Out-of-date

=====
==

Target:          sync://cpu.ABCo.com:2647/Projects/MEM
Parent:          sync://cpu.ABCo.com:2647/Projects/CPU
Base Directory: /dev/users/Marie/CPU/MEM

No local or remote hierarchical references found for configuration.

Workspace          Configuration          Object
Version            Version            Name
-----
1.6 (Locally Modified) 1.5          mem.v
Unmanaged          memstat.v

Configuration status: Out-of-date

=====
=

Status was not computed for the following configurations.
See above for details.

    sync://alu.ABCo.com:2647/Projects/ALU@R2
    sync://alu.ABCo.com:2647/Projects/CACHE@R4
    sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028

=====
=

Status of all visited configurations.
STATUS          TARGET          PATH
-----
---
Out-of-date sync://cpu.ABCo.com:2647/Projects/MEM
/dev/users/Marie/CPU/MEM
Out-of-date sync://cpu.ABCo.com:2647/Projects/CPU /dev/users/Marie/CPU

Summary: Out-of-date

```

This output gives Marie the following information:

- o The hierarchical references of the CPU module's default configuration in her work area are up-to-date with those of the CPU module on the SyncServer. (If she has references that are out-of-date, she can bring them up-to-date by using the **hcm get** operation again. See Marie Updates Her Work Area for information.)

- Even though a recursive **hcm showstatus** operation was specified, the operation does not descend into the ALU@R2 submodule, the CACHE@R4 submodule, or IP Gear Deliverable 1028 to review the status of its hierarchical references or files. The showstatus operation behaves in this way for a reference that is a release, an alias, an IP Gear deliverable, a vault folder residing on a non HCM-enabled server, or a reference that does not exist locally. The showstatus operation does descend into the MEM submodule (a default configuration) to review its hierarchical references.
 - Because the files of the CPU configuration and the MEM submodule configuration in the work area do not match those of that same configurations on the server, the output from **hcm showstatus -files** shows `Configuration status: Out-of-date` for both configurations. (When you use the **-files** option, the showstatus operation determines each configuration's status from both the status of its files and its hierarchical references. If either is out-of-date with the configuration on the server, the status of the configuration is out-of-date.)
2. The MEM submodule configuration has an unmanaged file (`memstat.v`), which Marie wants to include in the configuration. Because the **hcm put** operation does not check in unmanaged files, she uses the DesignSync checkin (**ci -new**) operation to check in the file. **Note:** The checkin steps Marie uses for checking in the unmanaged file depend on the type of configuration she wants to add the files to. For an example scenario, see A Designer Adds Files to a Configuration.
 3. The CPU and MEM configurations both have locally modified files. To put back on the SyncServer not only the files of the CPU module configuration, but also files of any submodule configuration that she modified, Marie uses the **hcm put** operation with the **-recursive** option:

```
stcl> cd /dev/users/Marie/CPU
stcl> hcm put -recursive
```

Because Marie does not specify the **-path** option, the operation uses her current work area directory.

The **hcm put -recursive** operation:

- Checks in modified files that Marie locked for edit and modified files that Marie did not lock for edit but that are the latest version on the branch. **Note:** The put operation does not check in files that are up-to-date.
- Cancels the locks on files locked for edit. This action occurs because Marie did not specify the **-lock** option.

Note: Marie can also use the DesignSync **cancel** command to remove locks that she has placed on unmodified files (with the **co -lock** or **ci -lock** command). This operation, in effect, performs an "un"checkout operation on the specified locked object.

If another user has locked the files (with the **hcm get -edit** operation), Marie can use the DesignSync **unlock** command to unlock the files. However, the **unlock** command typically is access controlled to allow only Synchronicity administrators to break other users' locks.

- Follows the CPU module's hierarchical reference to the MEM submodule and checks in files and cancels locks.

Note: The put operation does not operate on hierarchical references to the ALU, CACHE, and IOSTAT submodules. The operation does not operate on a hierarchical reference to a submodule that is a release, an alias, an IP Gear deliverable, or that does not exist on the server (for example, a configuration that has been removed from the server since Marie fetched it to her work area with the get operation).

- Continues following hierarchical references down through each level of the CPU module hierarchy (as defined by data in Marie's work area) and checking in modified files until all modified files of the hierarchy are checked in.
- Generates a RevisionControl note for each configuration in the CPU module hierarchy that has been put back on the SyncServer. (**Note:** If no files in the configuration were modified, no RevisionControl note for the configuration will be generated.)

Users receive email about the put operation if they have subscribed to email notification of RevisionControl notes on any of the modules or configurations that were checked in to the server.

4. Marie examines the output from the put operation to see if any of the submodule configurations in the hierarchy are selector configurations. (Output from the put operation lists selector configurations.) If she modified any files added to a configuration with a version tag and then checked the files in with **hcm put**, she must move the version tag to the new file versions. For an example scenario, see *Thomas Moves the Version Tag of Files Added to a Selector Configuration*.
5. Marie also examines the output from the put operation to see if any of the checkin operations failed. (Messages appear at the end of the output for the operation.) For example, the put operation may report errors for several files that failed to be checked in because they require merging. If there are any failures, Marie fixes the problems that caused them.
6. When Marie is satisfied that she has fixed all checkin problems, she uses the **hcm put** operation again.

```
stcl> hcm put -recursive
```

Related Topics

A Designer Adds Files to a Configuration

hcm put Command

hcm showstatus Command

The CPU Team Creates a Release

(Release of a Module Hierarchy)

For some time now, the CPU team has been building the CPU design, working on CPU design files and incorporating submodule releases such as release R2 of the ALU block. Now the team members believe they are ready to create a release of their entire CPU module hierarchy (the CPU module and all of its submodules).

The **hcm release** operation is a client-side operation, so Robert, the CPU team leader, will create the release based on the CPU module configuration and its hierarchy of submodules in his work area.

But before he performs the release operation, Robert wants to know if the files in his work area are the ones the team wants to release and if those files are ready to release. To check that he has the files the team wants to release, he uses the **hcm showstatus** operation. Then to determine if files are ready to release, he runs the regression tests his team has developed.

1. To see if any of the hierarchical references or files in his work area are out-of-date, Robert uses the **hcm showstatus -recursive** operation with the **-files** and **-releases** options. This operation compares the hierarchical references and files of the CPU module in his work area with those of the CPU module on the server. Because he specified the **-recursive** option, the showstatus operation shows the status of hierarchical references and files not only for the CPU module but also for each of its submodules, and so on down through the module hierarchy.

```
% stclc
stcl> cd /dev/users/Robert/CPU
stcl> hcm showstatus -files -recursive -releases
Target:          sync://cpu.ABCo.com:2647/Projects/CPU
Base Directory: /dev/users/Robert/CPU
```

STATUS	HREF	RELATIVE PATH
Up-to-date	sync://alu.ABCo.com:2647/Projects/ALU@R2	ALU
Local Only	sync://cpu.ABCo.com:2647/Projects/CACHE	CACHE
Server Only	sync://cpu.ABCo.com:2647/Projects/CACHE@R4	CACHE
Up-to-date	sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028	IOSTAT
Up-to-date	sync://cpu.ABCo.com:2647/Projects/MEM	MEM

The status of the following hrefs will not be individually reviewed:

HREF	REASON
------	--------

Scenarios for Using HCM

```
-----  
sync://alu.ABCo.com:2647/Projects/CACHE@R4      not local  
sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028  IP Gear deliverable
```

Workspace Version -----	Configuration Version -----	Object Name -----
1.1	1.2	cpu.v
1.3 (Locally Modified)	1.3	cpustat.v

Configuration status: Out-of-date

```
=====  
Target:      sync://alu.ABCo.com:2647/Projects/ALU@R2  
Parent:      sync://cpu.ABCo.com:2647/Projects/CPU  
Base Directory: /dev/users/Robert/CPU/ALU
```

No local or remote hierarchical references found for configuration.

Workspace Version -----	Configuration Version -----	Object Name -----
1.14	1.14	alu.v
1.11	1.11	reg8.v
1.5	1.5	aludoc/aluprojinfo.txt
1.9	1.9	aludoc/aluprojlist.txt

Configuration status: Up-to-date

```
=====  
Target:      sync://cpu.ABCo.com:2647/Projects/CACHE  
Parent:      sync://cpu.ABCo.com:2647/Projects/CPU  
Base Directory: /dev/users/Robert/CPU/CACHE
```

No local or remote hierarchical references found for configuration.

```
=====  
Target:      sync://cpu.ABCo.com:2647/Projects/MEM  
Parent:      sync://cpu.ABCo.com:2647/Projects/CPU  
Base Directory: /dev/users/Marie/CPU/MEM
```

No local or remote hierarchical references found for configuration.

Workspace Version -----	Configuration Version -----	Object Name -----
1.6	1.7	mem.v
1.2 (Locally Modified)	1.2	memstat.v
Unmanaged		memdoc.txt

Configuration status: Out-of-date


```

==

Status was not computed for the following configurations.
See above for details.

sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028

=====
=

Status of all visited configurations.
STATUS      Target                                     PATH
-----
-
Up-to-date  sync://alu.ABCo.com:2647/Projects/ALU@R2
            /dev/users/Marie/CPU/ALU
Up-to-date  sync://cpu.ABCo.com:2647/Projects/CACHE@R4
            /dev/users/Marie/CPU/CACHE
Unknown     sync://alu.ABCo.com:2647/Projects/CACHE@R4
            /dev/users/Marie/CPU/CACHE
Up-to-date  sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028
            /dev/users/Marie/CPU/IOSTAT
Out-of-date sync://cpu.ABCo.com:2647/Projects/MEM
            /dev/users/Marie/CPU/MEM

Summary: Out-of-date

```

This output gives Robert the following information:

- o The hierarchical references and files of the CPU module configuration in his work area differ from those of the CPU module on the SyncServer. The CPU module configuration on the server has a hierarchical reference to a new release of the CACHE submodule, CACHE@R4. In addition, the `cpu.v` file has a later version on the server. Robert needs to update his work area if he wants to include CACHE@R4 in the release he is creating.

Because the hierarchical references and the files in Robert's work area do not match those of the CPU configuration on the server, the output from **hcm showstatus -files** shows Configuration status: Out-of-date. (When you use the **-files** option, the showstatus operation determines each configuration's status from both the status of its files and its hierarchical references. If either or both are out-of-date with the configuration on the server, the status of the configuration is out-of-date.)

- o The files of the ALU@R2 release in his work area are up-to-date.

Because Robert specified the **-releases** option in addition to the **-files** option, the showstatus operation lists the status of the hierarchical references and files for the ALU@R2 release. Robert used this option because he will be doing some regression testing before he creates a release of the CPU module and he wants to know if he has added or

changed files of the ALU@R2 release in his work area, thereby affecting regression test results.

- o Even though a recursive **hcm showstatus** operation was specified, the operation does not descend into the CACHE@R4 submodule or IP Gear Deliverable 1028 to review the status of its hierarchical references or files. The showstatus operation behaves in this way for a reference that is an alias, an IP Gear deliverable, a vault folder residing on a non HCM-enabled server, a reference that does not exist locally, or a release (unless you specify the **-releases** option with the **-files** option). The showstatus operation descends into the MEM submodule (a default configuration) and the ALU@R2 submodule (because Robert specified the **-releases** option with the **-files** option).
 - o The output from **hcm showstatus -files** shows `Configuration status: Out-of-date` for the MEM configuration in Robert's work area. The showstatus operation reported this status because he files of the MEM submodule configuration in the work area do not match those of that same configurations on the server. (When you use the **-files** option, the showstatus operation determines each configuration's status from both the status of its files and its hierarchical references. If either or both are out-of-date with the configuration on the server, the status of the configuration is out-of-date.)
2. From the output of the **hcm showstatus -files** operation, Robert can see that he has one unmanaged file and several modified files in his work area. He wants to include these files in the release, so he uses the DesignSync **ci -new** operation to check in the unmanaged file and the **hcm put** operation to check in the modified files. For information, see *A Designer Adds Files to a Configuration and A Designer Puts a Module Hierarchy Back on the Server.*)
 3. From the output of the **hcm showstatus -files** operation, Robert notices that one of CPU module's hierarchical references (to CACHE) has changed to refer to a new configuration (CACHE@R4). In addition, the CPU and MEM modules on the server each contain a file version newer than the one in Robert's work area. To include the new CACHE@R4 configuration and the new file versions in the release, he updates his work area with a **hcm get -recursive** operation of the CPU module. For an example of this operation, see *A CPU Designer Gets a Module Hierarchy.*
 4. After ensuring that he has all of the files and references the team wants in the release, Robert runs the regression tests.
 5. Upon successful outcome of regression testing, Robert uses the **hcm release** operation to create a release. (The **hcm release** operation is recursive by default, so Robert does not need to specify the **-recursive** option.)

```
stcl> pwd
/dev/users/Robert/CPU
stcl> hcm release -name R1 -description "Baseline release
of CPU hierarchy"
```

Since Robert did not specify the **-path** option, the release operation operates on his current directory.

The **hcm release** operation:

- Creates the CPU@R1 release configuration on the server
- Creates a release for any submodule configuration that has not been released. (**Note:** The **hcm release** operation does not create a release of an IP Gear deliverable. Instead, the operation adds a reference from the release of the upper level module to the deliverable.)
- Includes the description (specified with the **-description** option) in the ProjectSync project. (If the **-description** option is not specified, the release operation provides a default description of "HCM release".) Team members can view the description in ProjectSync and in the output of the **hcm showconfs - report script** operation.

By creating the release CPU@R1, the CPU team has captured a snapshot of CPU design hierarchy at a particular stage of development. This "snapshot" cannot be changed and therefore can be reconstructed, if need be, at any time.

Other users are now able to create a new work area for the CPU design by using the **hcm get** operation. For an example of this operation, see A CPU Designer Gets a Module Hierarchy.

Other teams find that there is a new release of the CPU module. Teams who want to include the CPU design in their designs can now create a hierarchical reference from their design hierarchy to CPU@R1. For an example scenario of creating a hierarchical reference, see The CPU Team References the ALU Module.

Related Topics

hcm release Command

hcm showstatus Command

The ALU Team Creates an Alias for a Release

Over time, the ALU team creates ten or more releases of the ALU module. In addition, the CPU module refers to five other submodules that change just as frequently as ALU. The CPU team leader would prefer to monitor changes efficiently rather than to keep changing the CPU module's reference to the latest release of ALU.

In addition, the ALU team has just released an R12 release, but the team feels it is not ready for the CPU, MPU, and other teams to use the new release. So the team decides

to create an alias called GOLDEN and have it point to a release they want other teams to use. For now, that release is the R11 release.

Anne, the ALU team leader, creates the alias:

1. Anne makes sure that the R11 configuration has been released by entering the **hcm showconfs** command. For example:

```
% stcl
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU

Configurations of sync://alu.ABCo.com:2647/Projects/ALU
```

NAME	TYPE	OWNER	SELECTOR/ALIASED RELEASE
<Default>	Branch	Anne	Trunk:Latest
C1	Selector	Anne	valid12.06.01
R1	Release	Anne	
R2	Release	Anne	
.			
.			
R11	Release	Anne	
R12	Release	Anne	

2. Anne creates the GOLDEN alias, specifying ALU@R11 as the configuration to which GOLDEN refers. For example:

```
stcl> hcm mkalias -target
sync://alu.ABCo.com:2647/Projects/ALU@R11 -name GOLDEN -
description "Ready for distribution to other teams"
```

Notes:

- For information on alias names, see Naming Guidelines.
- The mkalias operation includes the description (specified with the **-description** option) in the ProjectSync project. (If the **-description** option is not specified, the mkalias operation provides a default description of "HCM alias".) Team members can view the description in ProjectSync and in the output of the **hcm showconfs -report script** operation.

Teams like the CPU team, who want to incorporate stable ALU releases into their designs, can now create hierarchical references from their module configurations to the alias ALU@GOLDEN.

The CPU Team Changes Its Reference to ALU's GOLDEN Release

Robert, the CPU team leader, is happy that the ALU team has created the GOLDEN alias. If he creates a hierarchical reference from the CPU module to the ALU@GOLDEN alias, he will not have to change CPU's reference to ALU every time the ALU team issues a new release. Robert changes the CPU module's reference to ALU@GOLDEN:

1. To identify the ALU configuration used in the hierarchy, Robert uses the **hcm showhrefs** operation:

```
% stcl
stcl> hcm showhrefs -target sync://cpu.ABCo.com:2647/Projects/CPU

Target:          sync://cpu.ABCo.com:2647/Projects/CPU

REFERENCE        URL                                     RELATIVE PATH
-----
ALU@R11          sync://cpu.ABCo.com:2647/Projects/ALU    ALU
CACHE@R4         sync://alu.ABCo.com:2647/Projects/CACHE  CACHE
Deliverable/1028 sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028 IOSTAT
```

2. Next, Robert removes the CPU module's reference to ALU@R11 release:

```
stcl> cd /dev/users/Robert/CPU
stcl> hcm rmhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://alu.ABCo.com:2647/Projects/ALU@R11 -relpath ALU
```

Note: The **-fromtarget**, **-totarget**, and **-relpath** options are not all required by the **hcm rmhref** operation. See the **hcm rmhref** command for information.

The **rmhref** operation removes the hierarchical reference from the server, in effect removing the ALU@R11 submodule configuration from the CPU module hierarchy.

3. Robert then adds a reference to the GOLDEN alias for ALU:

```
stcl> hcm addhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://alu.ABCo.com:2647/Projects/ALU@GOLDEN -relpath ALU
```

Note: The **-relpath** option is not required. If Robert did not specify the **-relpath** option, the **addhref** operation would use the **-totarget** module name as the name of the submodule's base directory and place it directly below the upper-level module's base directory. (In this example, the result would be the same as the specified value for **-relpath**.)

When CPU team members perform an **hcm get -recursive** operation of the CPU module, they will continue to get ALU@R11, since that release is the one the GOLDEN

alias currently represents. However, if the GOLDEN alias is changed to point to a new release of ALU, the recursive get operation fetches the new release.

Robert and other CPU team members subscribe to RevisionControl notes for the **hcm mkalias** and **hcm ralias** operations on the ALU@GOLDEN configuration. This action ensures they will be notified whenever the GOLDEN alias is changed to point to a new release or removed from the server. For information, see Robert Subscribes to RevisionControl Notes on HCM Operations.

The ALU Team Designates a Different Release as "GOLDEN"

At some point, the ALU team determines that a more recent release (R12, for example) is ready for other teams to use. Instead of having the other teams change their modules' references to R12, the ALU team leader changes the GOLDEN alias to point to ALU@R12. To change the alias, he uses the **hcm mkalias** operation again:

```
stcl> hcm mkalias -target
sync://alu.ABCo.com:2647/Projects/ALU@R12 -name GOLDEN -
description "Ready for distribution to other teams"
```

The mkalias operation:

- Changes the GOLDEN alias to point to ALU@R12
- Includes the description (specified with the **-description** option) in the ProjectSync project. (If the **-description** option is not specified, the mkalias operation provides a default description of "HCM alias".) Team members can view the description in ProjectSync and in the output of the **hcm showconfs -report script** operation.

The mkalias operation includes the description (specified with the **-description** option) in the ProjectSync project. (If the **-description** option is not specified, the mkalias operation provides a default description of "HCM alias".) Team members can view the description in ProjectSync and in the output of the **hcm showconfs -report script** operation.

Now when CPU team members perform an **hcm get -recursive** operation on the CPU module, they get files from ALU@R12 in their work areas.

If the CPU team creates a release of the CPU module configuration (with **hcm release**, which is recursive by default), the release operation resolves the GOLDEN alias to ALU@R12. For more information, see How the Release Operation Works.

Related Topics

hcm addhref Command

`hcm mkalias` Command

`hcm rmalias` Command

`hcm rmhref` Command

`hcm showconfs` Command

How HCM Operations Handle an Alias

The ALU Team Removes an Alias

Designers Use the Module Cache

ChipA543 is a large design; the design team would like to fetch the ChipA543 hierarchy from the HCM server in a shorter amount of time than the fetch has been taking. To decrease fetch time, the team leader has set up a local module cache, which contains releases of two of the submodules included in the ChipA543 hierarchy: CPU and STDLIB. When team members fetch the ChipA543 hierarchy, they can fetch these releases from the module cache instead of the HCM server.

The team leader has also set the default module cache paths and default module cache mode settings in the DesignSync registry. The setting for the default module cache paths is `/A543/cache1`. The setting for default module cache mode is link mode.

Because the default module cache paths and mode are set, team members do not need to specify the module cache path or mode when they use the get operation to fetch the releases from the module cache.

A Designer Creates Links to the Module Cache

Joe, a designer on the ChipA543 design team, has decided to take advantage of faster fetch time by using the team's module cache. In addition, he wants to save disk space required in his work area to hold the ChipA543 hierarchy. So he decides to have the get operation create links from his work area to the releases in the module cache.

To link from his work area to releases in the module cache, Joe takes these steps:

1. Joe isn't sure which submodules of the ChipA543 default configuration are in the module cache, so he uses the **`hcm showmcache`** operation to display the contents of the module cache:

```
% stclc
stcl> hcm showmcache

Mcachepaths search order:
```

```
/A543/cache1
```

Configurations found:

PATH	TARGET	AVAILABLE	HIERARCHY
/A543/cache1/CPU	sync://cpu.ABCo.com:2647/Projects/CPU@Rel1	yes	yes
/A543/cache1/STDLIB	sync://srvr1.ABCo.com:2647/Projects/STDLIB@RelA	yes	yes

Note: Because Joe's team leader has set the default module cache paths, Joe does not need to specify the **-mcachepaths** option. The showmcache operation uses the DesignSync registry setting to locate the module caches. Joe needs to specify the **-mcachepaths** option only if he wants to view module caches different from the default.

From the display, Joe sees that the module cache contains two of the releases used in the ChipA543 configuration. In addition, the `AVAILABLE` column of the display tells him that the releases are available for fetching or linking. The `HIERARCHY` column indicates that the entire hierarchy of each release is in the module cache. This hierarchy information pleases Joe; he wants to fetch the entire hierarchy of each release. (If only the upper-level module of each release is in the module cache, the get operation does not find the releases there and fetches them from the server instead.)

2. Joe creates a directory in his work area for the ChipA543 configuration hierarchy. He then changes directory to that directory:

```
stcl> cd /dev/users/Joe/designs
```

Note: For Joe to use module cache link mode, his work area directory must not contain a previously fetched HCM configuration. The get operation does not replace the contents of a work area directory with a link to the module cache.

3. He then uses the **hcm get** operation to fetch the ChipA543@Alpha configuration. Because Joe is using the default module cache path (`/A543/cache1`) and mode (link), he does not need to specify the **-mcachepaths** or **-mcachemode** options. (**Note:** Using **-mcachemode link** is not allowed on Windows platforms and the default module cache mode is copy.)

```
stcl> hcm get -recursive -target
sync://srvr1.ABCo.com:2647/Projects/ChipA543@Alpha -path
ChipA543
```


The get operation creates the ChipA543 directory specified with **-path** and then fetches the ChipA543 module hierarchy as directed by the **-recursive** option.

The get operation searches the module cache (specified in the default module cache paths registry setting) for releases of submodules in the ChipA543@Alpha configuration hierarchy. In this case, the get operation finds releases for the CPU and STDLIB submodules because the entire hierarchy of the each release exists in the module cache and that hierarchy level matches the hierarchy level Joe specified with **hcm get -recursive**.

Because the default module cache mode registry setting specifies link mode, the get operation creates a link from Joe's work area to each release's base directory in the module cache.

Note: Submodules in the ChipA543@Alpha hierarchy that do not have a release in the module cache are fetched from the server.

4. Joe decides that he wants to see which configurations he now has in the A543 directory in his work area. To list all of the configurations in that directory's hierarchy, he uses the **hcm showconfs** operation with the **-path** and **-directoryrecursive** options. For example:

```
stcl> hcm showconfs -path ChipA543 -directoryrecursive
```

PATH	TARGET HIERARCHY	TYPE	
Cpu (mcached) yes	sync://cpu.ABCo.com:2647/Projects/CPU@Rel1	Release	
Cpu/Alu	sync://alu.ABCo.com:2647/Projects/ALU@Rel4	Release	yes
Mem Rel16 yes	sync://cpu.ABCo.com:2647/Projects/MEM@Golden	Alias to	
Pll	sync://cpu.ABCo.com:2647/Projects/PLL@BetaBr no	Branch	
SLib (mcached) yes	sync://srvr1.ABCo.com:2647/Projects/STDLIB@RelA	Release	

The showconfs operation lists:

- Each directory that contains a configuration. (**Note:** Because Joe specified a relative path for the **-path** option, the showconfs operation shows the relative path for each directory.)
- The URL of the configuration on the SyncServer.
- The configuration type (alias, branch, release, or selector). For aliases, the TYPE column lists the alias and the release to which it points. For releases, (mcached) shows that the directory contains a link to a release in the module cache.

- Whether the entire hierarchy of the configuration is present in the directory or not.

A Designer Copies a Release from the Module Cache

Sara is also a member of the ChipA543 design team. Like Joe and the other designers, she wants to use the module cache instead of fetching from the server. Unlike Joe, Sara plans to use a design tool that writes temporary files to the directories that hold the CPU@Rel1 and STDLIB@RelA module configurations.

Because the design tool needs write permission to the CPU and STDLIB directories and the module cache has read-only permission, Sara cannot use the get operation in link mode (the default module cache mode) to create links from her work area to the releases in the cache. Still, she wants to avoid fetching the CPU@Rel1 and STDLIB@RelA module configurations from the server. Instead of using link mode, she uses copy mode (**-mcachemode copy**) with the get operation:

1. Sara creates a directory in her work area where she wants to place the top-level ChipA543 module. She then changes directory to that directory:

```
stcl> cd /dev/users/Sara/projects
```

Note: For Sara to use module cache copy mode, her work area directory must not contain a previously fetched HCM configuration. The get operation does not replace the contents of a work area directory with a copy from the module cache.

2. To fetch the ChipA543@Alpha hierarchy she uses the get operation with the **-recursive** option. In addition, she specifies the **-mcachemode copy** option because she wants to override the default module cache mode (link).

```
stcl> hcm get -recursive -target
sync://srvr1.ABCo.com:2647/Projects/ChipA543@Alpha -path
A543 -mcachemode copy
```

Note: Because Sara is using the default module cache path (/A543/cache1), she does not need to specify the **-mcachepaths** option. She needs to specify that option only if she wants to use module caches different from the default.

The get operation creates the A543 directory specified with **-path** and then fetches the ChipA543 module hierarchy as directed by the **-recursive** option.

The get operation searches the module cache (specified in the default module cache paths registry setting) for releases of submodules in the ChipA543@Alpha configuration hierarchy. In this case, the get operation finds releases for the CPU and STDLIB submodules because the entire hierarchy of the each release exists in the module cache and that hierarchy level matches the hierarchy level Sara specified with **hcm get -recursive**.

Because Sara specified **-mcachemode copy**, the get operation copies the CPU and STDLIB releases the cache to Sara's work area.

Note: Submodules in the ChipA543@Alpha hierarchy that do not have a release in the module cache are fetched from the server.

A Designer Uses Both Links to and Copies from the Cache

Marc is a member of the ChipA543 design team who, like Joe, wants to save disk space by linking to releases in the module cache. So Marc plans to use the module cache when he fetches the ChipA543 configuration to his work area. However, he also plans to run a parasitic extraction tool on the CPU configuration to create a log file and parasitics file in the CPU directory. The need to write files to the directory poses a problem. If Marc creates links from his work area to the CPU module in the cache, the parasitics tool will not be allowed to create files in the cache directory because it is write-protected. So, he needs to have a copy of the CPU release in his work area, not a link to the module cache.

To solve the problem, Marc uses link mode to fetch the ChipA543 configuration hierarchy and then uses copy mode to fetch just the CPU submodule configuration:

1. Marc uses the **hcm showmcache** operation to display the contents of the module cache:

```
% stclc
stcl> hcm showmcache

Mcachepaths search order:

/A543/cache1

Configurations found:

PATH                                TARGET                                AVAILABLE  HIERARCHY
-----
/A543/cache1/CPU                    sync://cpu.ABCo.com:2647/Projects/CPU@Rel1
    yes                               yes
/A543/cache1/CPUTop                 sync://cpu.ABCo.com:2647/Projects/CPU@Rel1
    yes                               no
/A543/cache1/STDLIB                 sync://srvr1.ABCo.com:2647/Projects/STDLIB@RelA
yes                                 yes
```

Note: Because the team leader has set the default module cache paths, Marc does not need to specify the **-mcachepaths** option. The showmcache operation uses the DesignSync registry setting to locate the module caches. Marc needs to specify the **-mcachepaths** option only if he wants to view module caches different from the default.

From the display, Marc sees that the module cache contains the releases used in the ChipA543 configuration. He also sees that there are two cache entries for the CPU@Rel1, one in a directory called CPU, the other in CPUPop. The `HIERARCHY` column tells him that the first CPU entry is the entire hierarchy of the CPU release and the second entry is just the upper-level module of the CPU configuration. In addition, the `AVAILABLE` column of the display indicates that the releases are available for fetching or linking.

- Marc changes directory to the work area directory where he wants to place the ChipA543@Alpha configuration:

```
stcl> cd /dev/users/Marc/designs
```

Note: For Marc to use module cache link or copy mode, his work area directory must not contain a previously fetched HCM configuration. The get operation does not replace the contents of a work area directory with a copy from the module cache.

- He uses **hcm get** option to fetch the ChipA543 configuration. (**Note:** Because Marc is using the default module cache path (`/A543/cache1`) and mode (link), he does not need to specify the **-mcachepaths** or **-mcachemode** options.)

```
stcl> hcm get -target
sync://srvr1.ABCo.com:2647/Projects/ChipA543@Test -
recursive -path A543Parasitics
```

The get operation searches the module cache (specified in the default module cache paths registry setting) for releases of submodules in the ChipA543@Alpha configuration hierarchy. In this case, the get operation finds releases for the CPU and STDLIB submodules because the entire hierarchy of the each release exists in the module cache and that hierarchy level matches the hierarchy level Marc specified with **hcm get -recursive**.

Because the default module cache mode registry setting specifies link mode, the get operation creates a link from Marc's work area to each release's base directory in the module cache.

- To fetch a copy of just the upper-level module of the CPU@Rel1 release configuration from the module cache to the CPU directory in his work area, Marc uses the get operation again. However, this time there are two differences:
 - He uses **hcm get** without the **-recursive** option to fetch just the upper-level module of the CPU@Rel1 release configuration. (Having the files of the CPU module in a writable work area directory is all Marc needs to run his parasitic extraction tool.)
 - He specifies the **-mcachemode copy** option to override the default module cache mode (link).

```
stcl> hcm get -target  
sync://srvr1.ABCo.com:2647/Projects/CPU@Rel1 -path  
A543Parasitics/CPU -mcachemode copy
```

The operation searches the module cache (specified in the default module cache paths registry setting) for the CPU@REL1 release. In this case, the get operation finds the CPU release submodule because the upper-level module of that release's hierarchy exists as a separate entry in the module cache and that hierarchy level matches the hierarchy level Marc specified by using **hcm get** without **-recursive**.

Because he specified **-mcachemode copy**, the get operation removes the link from Marc's work area to the CPU release in the module cache. Then get operation then copies the CPU release to his work area.

Note: For the steps to work, Marc must perform them in the order shown (fetching in link mode before fetching in copy mode). The hcm get operation does not allow overwriting the contents of a work area with links to the module cache. If Marc fetched ChipA543@Alpha in copy mode and then tried to fetch CPU@REL1 in link mode, the fetch of CPU would fail.

Related Topics

hcm get Command

hcm showconfs Command

hcm showmcache Command

Module Cache

Setting the Default Module Cache Path or Mode

The MPU Team Upgrades to HCM

Drawn by the success of the CPU team, the MPU design team wants to be a consumer of the ALU module.

In the past, the MPU team used REFERENCE statements in a ProjectSync project to import modules from other design projects into the MPU design. Recently, the Synchronicity administrator installed HCM software on the MPU development server. The MPU team decides to upgrade their existing DesignSync vault directories to HCM modules and change vault REFERENCES to HCM hierarchical references.

Jeanne, the MPU team leader, performs the upgrade:

1. To determine which DesignSync vault directories are good candidates to be HCM modules, Jeanne identifies the ProjectSync projects. To view a list of all ProjectSync projects defined on the mpu SyncServer, she clicks **Data Sheet** on the ProjectSync menu. She can also use the DesignSync **url projects** command for this task. For example:

```
% stcl
stcl> url projects sync://mpu.ABCo.com:2647
```

2. Jeanne runs the **hcm upgrade** operation on each vault directory that is a potential module. For example, suppose that Jeanne identifies the MPU vault directory as a potential module. She uses the **hcm upgrade** operation, specifying the MPU vault directory as the target of the operation:

```
stcl> hcm upgrade -target
sync://mpu.ABCo.com:2647/Projects/MPU
```

The **hcm upgrade** operation:

- Creates an HCM module from the vault directory specified with the **-target** option.
- Scans the vault directory specified by the **-target** option for `sync_project.txt` files located anywhere in the directory structure below that vault directory.
- For `sync_project.txt` files containing REFERENCE or CONFIG statements, the operation creates a hierarchical reference from each REFERENCE and CONFIG statement. (The operation determines the relative path from the REFERENCE statement.)
- Deletes the `sync_project.txt` file that contained the REFERENCES and the folder in which the file was located. (However, a copy of the file is saved in the vault folder located one level above the folder that was deleted.)
- Updates existing email subscriptions associated with a vault path to subscriptions on the newly-created module. Because the email subscriptions apply only to the newly created module, not its submodules, the upgrade operation sets the **Scope** field of each subscription to **This object only** (the default for HCM modules). HCM also sets the **Scope** field to **This object only** for existing subscriptions to any configurations of the module.

In addition, for each existing subscription associated with the vault path, the upgrade operation adds a new subscription covering the entire vault structure below the vault folder that is being converted into a module.

For example, suppose the vault directory

`sync://mpu.ABCo.com:2647/Projects/MPU/blocks/ALU` has a `sync_project.txt` file containing the following entries:

```
NAME ALU
REFERENCE sync://alu.ABCo.com:2647/Projects/ALU
CONFIG R1 A3
CONFIG R2 A6
.
.
```

In this example, when Jeanne runs the **hcm upgrade** operation on the MPU vault directory (`sync://mpu.ABCo.com:2647/Projects/MPU`), the operation:

- From the vault path, creates the MPU module and its default configuration.
- Scans the vault directory specified by the **-target** option for `sync_project.txt` files and finds one in `sync://mpu.ABCo.com:2647/Projects/MPU/blocks/ALU`.
- From the REFERENCE statement, creates a hierarchical reference from the MPU module's default configuration to the ALU module's default configuration, specifying `blocks/ALU` as the relative path.
- For each CONFIG statement, creates a hierarchical reference: from MPU@R1 to ALU@A3 (with a relative path of `blocks/ALU`) and from MPU@R2 to ALU@A6 (with a relative path of `blocks/ALU`).
- Deletes the `sync_project.txt` file containing the REFERENCE statement. (However, a copy of the file is saved to `<SYNC_DIR>/../syncdata/mpu/2647/server_vault/Projects/MPU/blocks/.SYNC.ALU.sync_project.txt`.)
- Deletes the ALU folder.
- Updates Jeanne's existing email subscriptions associated with the MPU vault path to subscriptions on the MPU module default configuration, and for each subscription, sets the **Scope** field to **This object only**. The upgrade operation also sets the same scope for Jeanne's existing subscriptions to configurations of MPU. Finally, for each existing subscription associated with the vault path, the upgrade operation adds a new subscription covering the entire vault structure below the vault folder that is being converted into a module.

Note: After performing the upgrade, Jeanne tells the MPU team to fetch each new configuration to an empty work area rather than trying to update their existing work areas. (Synchronicity recommends that users take this action after an **hcm upgrade** operation.)

After the upgrade has been performed, the MPU team can use **hcm showconfs** to view the HCM configurations and **hcm showhrefs** to view hierarchical references created by the operation.

Related Topics

[hcm showconfs Command](#)

hcm showhrefs Command

hcm upgrade Command

url projects

A Designer Creates a Configuration for Experimentation

The CPU team does its development work on the CPU module's default configuration, which has a hierarchical reference to the alias ALU@GOLDEN. Currently, the GOLDEN alias points to the ALU@R12 release configuration. While it has created an R14 release, the ALU team has not yet changed the ALU@GOLDEN alias to point to ALU@R14.

In order to make progress on her part of the CPU design, Shirley, a CPU designer, needs to experiment with ALU@R14 and possibly later releases of other modules too. Shirley wants to run an application against the CPU module's files as they currently exist; she also wants to test against the latest ALU release, ALU@R14. But she knows that releases cannot be changed, so she creates her own configuration for experimentation:

1. Shirley uses the **hcm mkconf** operation with its **-branch** option to create the configuration:

```
% stcl
stcl> hcm mkconf -branch Trunk -name Shirleyconf -target
sync://cpu.ABCo.com:2647/Projects/CPU -description
"Configuration of CPU for experiment"
```

Note: For information on configuration names, see Naming Guidelines.

Because Shirley specified no particular configuration of CPU as the target, the **hcm mkconf** operation creates the new configuration from the CPU module's default configuration.

The **mkconf** operation creates the new configuration on the SyncServer and gives it the name CPU@Shirleyconf. In addition, the operation includes the description (specified with the **-description** option) in the ProjectSync project. (If the **-description** option is not specified, the **mkconf** operation provides a default description of "HCM configuration".) Team members can view the description in ProjectSync and in the output of the **hcm showconfs -report script** operation.

2. Shirley creates the clone script, a Tcl script that uses the **hcm showhrefs** and **hcm addhref** operations to copy all of the hierarchical references from one specified module configuration to another. The script might look something like this example:


```
proc clone {from_target to_target} {  
    foreach href [hcm showhrefs -target $from_target -report script] {  
        array set hrefArray $href  
        hcm addhref -from $to_target -to $hrefArray(target) -relpath  
$hrefArray(relpath)  
    }  
}
```

3. She then sources the clone script and runs it:

```
stcl> source clone.tcl  
stcl> clone sync://cpu.ABCo.com:2647/Projects/CPU  
sync://cpu.ABCo.com:2647/Projects/CPU@Shirleyconf
```

The script recreates the hierarchical references of the CPU module's default configuration. The result is a configuration (CPU@Shirleyconf) identical to the one that the rest of the CPU team is using, except that Shirley can modify it.

4. As a final step, Shirley replaces the reference from the CPU module to ALU@GOLDEN with a reference to ALU@R14:

```
stcl> hcm rmhref -fromtarget  
sync://cpu.ABCo.com:2647/Projects/CPU@Shirleyconf -totarget  
sync://alu.ABCo.com:2647/Projects/ALU@GOLDEN -relpath ALU  
  
stcl> hcm addhref -fromtarget  
sync://cpu.ABCo.com:2647/Projects/CPU@Shirleyconf -totarget  
sync://alu.ABCo.com:2647/Projects/ALU@R14 -relpath ALU
```

Now Shirley can perform an **hcm get -recursive** operation on CPU@Shirleyconf and experiment with the design without affecting the work of other members on the CPU team.

Related Topics

[hcm addhref Command](#)

[hcm get Command](#)

[hcm mkconf Command](#)

[hcm rmhref Command](#)

[hcm showhrefs Command](#)

[Using DesignSync with HCM](#)

[A Designer Removes a Configuration](#)

The MPU Team References the ALU Work in Progress

The CPU team thinks of the ALU module as a reusable block with an interface they can use to incorporate the block into their design. So the team relies on releases of the ALU module.

The MPU team, on the other hand, considers the ALU module as part of their in-progress design. For this reason, MPU team members feel that their module needs to refer to the ALU team's work in progress, not just its releases.

To create a hierarchical reference to the ALU configuration that contains work in progress, the MPU team leader uses the **hcm addhref** operation:

```
% stclc
stcl> hcm addhref -fromtarget
sync://mpu.ABCo.com:2647/Projects/MPU -totarget
sync://alu.ABCo.com:2647/Projects/ALU
```

Because the team leader did not specify a configuration for the MPU or the ALU module, the HCM software creates the reference from the MPU module's default configuration to the ALU module's default configuration.

Because the team leader did not specify a **-relpath** option, the addhref operation uses `ALU` as the **-relpath** value. (By default, the operation uses the **-totarget** module name as the name of the submodule's base directory and places it directly below the upper-level module base directory.)

The addhref operation creates the reference and stores the information with the MPU module default configuration on the SyncServer.

Note: Although this scenario shows use of a module's default configuration as the work-in-progress configuration, any branch configuration can be used for this purpose. For example, instead of using the default configuration, the ALU team could branch their files and use **hcm mkconf** to create an HCM configuration called `ALU@Work` that the team uses for development work in progress. For an example scenario of creating a configuration from branched files, see [The CPU Team Creates a Work-in-Progress Configuration from a Release](#).

When an MPU designer uses the **hcm get -recursive** operation to get the MPU module, the operation fetches the latest files on the `Trunk` branch of the MPU module and the latest files on the `Trunk` branch the ALU module.

Related Topics

hcm addhref Command

hcm get Command

hcm mkconf Command

The CPU Team Creates a Work-in-Progress Configuration from a Release

In the scenario A Designer Creates a Configuration for Experimentation, a designer on the CPU team used the CPU module's default configuration to create a configuration to experiment with the CPU design. The CPU team decides to use this method to create a configuration they can use to solve a problem with a release.

The CPU team has discovered a problem on release R5 of the CPU module, which is in wide-scale use by other teams. In addition, CPU team members have already made significant code changes during development of release R6. To make bug fixes to CPU@R5 code, the team decides it needs to create a branch of the CPU@R5 module configuration.

The CPU team leader (Robert) creates the branch of the configuration:

1. Robert is going to create the configuration in his work area. He starts by creating a new work area directory and changing directory to it:

```
% stclc
stcl> mkfolder /dev/users/Robert/CPU_R5_Bug_Fix
stcl> cd /dev/users/Robert/CPU_R5_Bug_Fix
```

2. Next he uses the **hcm get** operation to fetch to his work area only the upper-level CPU module of the CPU@R5 release configuration. (**Note:** The **hcm get** operation is nonrecursive by default.)

```
stcl> hcm get -target
sync://cpu.ABCo.com:2647/Projects/CPU@R5
```

Since Robert did not specify the **-path** option, the operation fetches the module to his current work area directory.

3. He then uses the DesignSync **mkbranch** command to branch the files, calling the branch R5_Bug_Fix.

```
stcl> mkbranch -recursive R5_Bug_Fix .
```

The **mkbranch** operation creates new branches of all files in the folder and subfolders contained in the upper-level CPU module Robert fetched to his work area with **hcm get**.

Note: Using the **mkbranch -recursive** command is not generally recommended with HCM module configurations; however, its use is acceptable in this particular case. See Using DesignSync with HCM for more information.

4. Robert creates a branch configuration of the new R5_Bug_Fix branch, giving the configuration the same name as the branch:

```
stcl> hcm mkconf -branch R5_Bug_Fix -name R5_Bug_Fix -
target sync://cpu.ABCo.com:2647/Projects/CPU -description
"WIP configuration for R5 bug fix"
```

The mkconf operation:

- Creates the configuration from the R5_Bug_Fix branch
 - Includes the description (specified with the **-description** option) in the ProjectSync project. (If the **-description** option is not specified, the mkconf operation provides a default description of "HCM configuration".) Team members can view the description in ProjectSync and in the output of the **hcm showconfs -report script** operation.
5. Robert uses the `clone.tcl` script to copy all of the CPU@R5 hierarchical references to the new configuration, CPU@R5_Bug_Fix. (To view the script, see A Designer Creates a Configuration for Experimentation.)

```
stcl> clone sync://cpu.ABCo.com:2647/Projects/CPU@R5
sync://cpu.ABCo.com:2647/Projects/CPU@R5_Bug_Fix
```

At this point, CPU team members get the CPU@R5_Bug_Fix module configuration to their work areas and begin fixing the code. They can modify files or hierarchical references. They also can branch any of the submodules, if need be, to fix problems in a submodule's code.

Related Topics

[hcm get Command](#)

[hcm mkconf Command](#)

[mkbranch Command](#)

[mkfolder Command](#)

[Using DesignSync with HCM](#)

[A Designer Creates a Configuration for Experimentation](#)

The CPU Team Develops a Checking Script Using HCM

Through some trial and error, the CPU team discovers that their release process has not caught some critical errors in the submodules of CPU. By doing queries of the module hierarchy, the team was able to find all defects that were logged; however, one variety of defect escaped detection until integration time.

The team discovers that a day can be lost assembling a massive chip and simulating it. However, half the time of running the simulation and analyzing the results can be saved if the data input to the simulation is itself validated before running the simulation.

To better identify defects during the development process, the team establishes release criteria that each submodule must meet:

- Every module must have a `test` directory and a `doc` directory in its hierarchy .
- The `test` directory must contain a file that can be parsed in order to determine that all the tests passed.
- The `test` directory's file must have a later date than one of the RTL files. In addition, the date of the test results file must be later than all of the RTL files for that module (thereby confirming that the test results are based on all of the latest RTL files).
- The `doc` directory must have a non-empty file named after the module.

Robert, the CPU team leader, develops a script called `mod_check` to check that each submodule meets the criteria. The script makes some assumptions about the CPU module's vault structure:

- The `test` and `doc` directories contain files under DesignSync revision control.
- Also under revision control are the test results file and the `doc` directory's file named after the submodule.
- To compare the dates of the files, the checking script must be able to locate the modification date of the test files and of the RTL files against which it is checking.
- Project leaders use the **`hcm get`** operation to fetch files into their work areas so that files can be checked and subsequently released.

Robert develops a script (`project_check`) to perform the submodule checks at each level of the module hierarchy. This script:

- Identifies submodules (by using the **`hcm showhrefs`** operation without the **`-target`** option) to display hierarchical references that reside with the module's local data
- Executes the `mod_check` script at each level of the module hierarchy to check for the existence of directories, files, and test results required to meet the criteria
- Uses DesignSync utilities such as **`url properties`** to determine the checkin time of the files. These checkin times can be used to determine that the test results were produced after the latest time of the RTL files for the module being checked.

- (If all checks are successful) performs the **hcm release** operation on the module and initiates regression tests.

Robert then performs the submodule checks:

1. Since he will perform the submodule checks and release the module from his work area, Robert uses the **hcm get -recursive** operation to fetch the CPU module hierarchy to his work area. For example:

```
% stclc
stcl> cd /dev/users/Robert/CPU
stcl> hcm get -recursive
```

Because he has used the **hcm get** operation on this module before, Robert does not need to specify the **-target** option. (The HCM software stored the information he specified when he first got the CPU module and recalls it.)

Because Robert did not specify a **-path** option, the operation uses his current work area directory.

2. Robert then runs the `project_check` script, which checks each submodule against the release criteria and, if all checks are successful, releases the module and initiates regression tests.

Related Topics

hcm get Command

hcm release Command

hcm showhrefs Command

url properties

The CPU Team References a Tools Module

The CPU team is developing the CPU module, which relies on other submodules, as is typical in HCM usage. However, the team also relies on some critical tools that are not part of the CPU design project, for example, the Synopsys Design Compiler version 4.3.

While the Design Compiler tool doesn't have source files to be managed, the CPU team wants to use ProjectSync to track any defects that affect the tool's use on projects going on in the company. When team members perform a ProjectSync hierarchical query on the CPU module, they want the query to show Design Compiler defects as well as those on other contributing modules. They also want a hierarchical subscription for notes on the CPU module to include Design Compiler notes.

To accomplish these goals, the team decides to create an HCM module for the Design Compiler tool and include it as a submodule in the CPU module's hierarchy.

Robert, the CPU team leader, creates the module and makes it a submodule of CPU:

1. Robert uses the **hcm mkmod** operation to create a module for the Design Compiler tool:

```
% stcl
stcl> hcm mkmod -target
sync://cpu.ABCo.com:2647/Projects/DesignCompiler -
description "hcm module for Design Compiler tool"
```

The **mkmod** operation:

- Defines the DesignCompiler vault folder as a module and creates a default configuration for it.
 - Creates a ProjectSync project for the DesignCompiler module. This action allows ProjectSync to recognize the module.
 - Includes the description (specified with the **-description** option) in the ProjectSync project. This information is also available in HCM through the **hcm showmods -report script** operation.
2. Next, he uses the **hcm mkconf** operation to create the v4.3 configuration of the DesignCompiler module:

```
stcl> hcm mkconf -target
sync://cpu.ABCo.com:2647/Projects/DesignCompiler -name v4.3
```

3. To make DesignCompiler a submodule of CPU, Robert uses the **hcm addhref** operation to create a hierarchical reference from the CPU module to the DesignCompiler module.

In addition, Robert specifies an empty value (" ") for the **-relpath** option. This empty value indicates that when the submodule is fetched with **hcm get**, the operation should not fetch the submodule's contents (files and directories). However, ProjectSync hierarchical subscription and queries on the upper-level CPU module include the referenced DesignCompiler@v4.3 module configuration.

```
stcl> hcm addhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://cpu.ABCo.com:2647/Projects/DesignCompiler@v4.3 -
relpath ""
```

Because Robert specified only the CPU module (without a configuration) as the **-fromtarget**, the **hcm addhref** operation creates a reference from the CPU

module's default configuration to the DesignCompiler@v4.3 module configuration.

The **hcm addhref** operation creates the reference and stores the information with the CPU module default configuration on the SyncServer.

To change or remove the reference, Robert would use the **hcm rmhref** operation. For an example scenario, see The CPU Team Changes a Reference to a New ALU Release.

Related Topics

hcm addhref Command

hcm rmhref Command

hcm mkconf Command

hcm mkmod Command

The MPU Team Removes a Module

(Removing a Module and Its Contents from the SyncServer)

When HCM was first installed, MPU team members wanted to experiment with its functionality within their design environment. They created a module (TEST) for that purpose and, in the course of their experiments, several configurations of TEST were created as well.

The team has finished testing HCM functionality and no longer needs the TEST module. They decide to remove the module and all its configurations from their server.

Note: This scenario assumes that the MPU team leader has access privileges to remove modules. For information, see Access Controls on HCM Operations.

Jeanne, the MPU team leader, uses the **hcm rmmod** operation to remove the TEST module:

1. Because the **rmmod** operation removes all configurations of a module, Jeanne first uses the **hcm showconfs** operation to display the TEST module's configurations:

```
% stclc
stcl> hcm showconfs -target
sync://mpu.ABCo.com:2647/Projects/TEST
```



```
Configurations of module
sync://mpu.ABCo.com:2647/Projects/TEST
```

NAME	TYPE	OWNER	SELECTOR/ALIASED RELEASE
<Default>	Branch	Jeanne	Trunk:Latest
C1	Selector	Jeanne	test031902
C2	Selector	Jeanne	test032502
R1	Release	Jeanne	
R2	Release	Jeanne	

2. Satisfied that the team no longer needs any of the TEST module's configurations, Jeanne uses the **hcm rmmod** operation to remove the module from the SyncServer:

```
stcl> hcm rmmod -target
sync://mpu.ABCo.com:2647/Projects/TEST -vaultdata -notes
```

The rmmod operation:

- Removes the TEST module and all its configurations from the mpu server.

Note: This operation does not affect the data in Jeanne's work area. For example, suppose that Jeanne had fetched the TEST module to her work area and then removed the module from the server. An **hcm showstatus** operation comparing her work area to the server would show the deleted TEST module configurations in her work area but report that they do not exist on the server.

- Detaches the ProjectSync notes that are attached to the TEST module and its configurations, except for the RevisionControl note generated by the rmmod operation.
- Generates a RevisionControl note. Users who have subscribed to email notification of RevisionControl notes on the TEST module or its configurations receive email that the module has been removed.
- Removes all files (even locked files) in the vault folder in which the TEST module resides. The operation also removes the vault folder and any folders in the path specified. (This behavior is a result of the specifying the **-vaultdata** option.)
- Deletes those notes that were detached and not attached to other objects (a result of specifying the **-notes** option).

Now when the MPU or other teams use **hcm showmods** to display a list of the modules on the server, the TEST module does not appear on the list.

3. The `rmmod` operation does not delete Jeanne's subscription to email on the TEST module, so she deletes that subscription. (For an example scenario, see Robert Deletes an Email Subscription.)

The MPU Team Leader Removes a Module Created By Mistake

(Removing a Module But Not Its Contents)

The MPU team decides to create a module that contains their design tools. Jeanne, the MPU team leader, uses the `hcm mkmod` operation to create the module. However, after creating the module, she realizes that she created the module with a target name of `Scripts` instead of `tclScripts`.

1. Jeanne uses `hcm rmmod` to remove the module from the SyncServer. Because she wants to remove just the `Scripts` module but not its contents or vault folder, Jeanne does not specify the `-vaultdata` option:

```
stcl> hcm rmmod -target
sync://mpu.ABCo.com:2647/Projects/Scripts
```

The `rmmod` operation:

- Removes the `Scripts` module from the mpu server. The operation also removes the hierarchical references of each of the module's configurations (if any). (**Note:** This operation does not affect the data in Jeanne's work area.)
- Detaches the `ProjectSync` notes attached to the module.
- Generates a `RevisionControl` note. Users subscribed to receive email notification of `RevisionControl` notes on either the module or its configurations receive notification that the module was removed.

Note: Because Jeanne did not specify the `-vaultdata` option, the `rmmod` operation removes the `Scripts` module but does not remove its contents, vault folder, or configurations. An `hcm showconfs` operation on the `Scripts` module would still list the configurations of the `Scripts` module (if any), even though the module was removed. In addition, because the `Scripts` vault folder resides in the `Projects` folder on the server, `ProjectSync` still displays the `Scripts` module and its configurations as a project.

2. Jeanne then uses the `hcm mkmod` operation to create the module from the correct vault folder (`tclScripts`).

Related Topics

`hcm rmmod` Command

The ALU Team Creates a Module from a Vault

A Designer Removes a Configuration

In the scenario A Designer Creates a Configuration for Experimentation, Shirley, a CPU designer, created a configuration of the CPU module so that she could experiment with a submodule release not included in the CPU default configuration.

Shirley has finished her experiment and no longer needs the configuration she created (CPU@Shirleyconf). She decides to remove the configuration from the SyncServer.

Note: This scenario assumes that Shirley has access privileges to remove configurations. For information, see Access Controls on HCM Operations.

1. Shirley uses the **hcm showconfs** operation to display the configurations of CPU on the cpu SyncServer:

```
% stcl
stcl> hcm showconfs -target
sync://cpu.ABCo.com:2647/Projects/CPU

Configurations of module
sync://cpu.ABCo.com:2647/Projects/CPU
```

NAME	TYPE	OWNER	SELECTOR/ALIASED	RELEASE
<Default>	Branch	Robert	Trunk:Latest	
R1	Release	Robert		
R2	Release	Robert		
Shirleyconf	Branch	Shirley	Trunk:Latest	

2. Next she uses the **hcm rmconf** operation to remove the CPU@Shirleyconf module configuration from the SyncServer:

```
stcl> hcm rmconf -target
sync://cpu.ABCo.com:2647/Projects/CPU@Shirleyconf -notes
```

The rmconf operation:

- Removes the CPU@Shirleyconf configuration (a branch configuration) from the server.

Note: This operation does not affect the data in Shirley's work area. For example, suppose that Shirley had fetched the CPU@Shirleyconf configuration to her work area and then removed the configuration from the server. An **hcm showstatus** operation comparing her work area to the

server would show the CPU@Shirleyconf module configuration in her work area but report that it does not exist on the server.

- Detaches any ProjectSync notes attached to the CPU@Shirleyconf configuration.
- Deletes those notes that were detached and not attached to other objects (a result of specifying the **-notes** option). **Note:** Notes for the Shirleyconf configuration may not be deleted, even though Shirley specified the **-notes** option. This behavior occurs when notes were created and attached through the ProjectSync GUI, rather than some other means, such as a server-side script. When you use the ProjectSync GUI to attach a note to a configuration, ProjectSync also attaches it to the project (module). The **rmconf -notes** operation does not delete the notes because they are attached to the CPU project as well as its Shirleyconf configuration.
- Generates a RevisionControl note. Users who have subscribed to email notification of RevisionControl notes on the CPU@Shirleyconf configuration or the CPU module receive email that the configuration has been removed.

Notes:

- The rmconf operation cannot remove a release or an alias.
 - The rmconf operation is not recursive. For example, it does not follow hierarchical references to submodules of CPU@Shirleyconf and remove those submodule configurations. To remove the entire CPU@Shirleyconf module hierarchy, Shirley would use the rmconf operation to remove each submodule configuration in the hierarchy.
3. The rmconf operation does not delete Shirley's subscription to email on the CPU@Shirleyconf configuration, so Shirley deletes that subscription. (For an example scenario, see Robert Deletes an Email Subscription.)

Notes:

- The rmconf operation does not remove hierarchical references that other modules have to the CPU@Shirleyconf configuration. For this reason, it is important for users to subscribe to email notification of RevisionControl notes on such HCM operations as **hcm mkconf** and **hcm rmconf**. If users receive notification of creation and removal of configurations, they can update their hierarchical references to the configurations accordingly. (For an example scenario, see Robert Subscribes to RevisionControl Notes on HCM Operations.)
- Removing the reference to the CPU@Shirleyconf does not remove the data or contents of CPU@Shirleyconf from users' work areas. However, users can remove the contents of a removed configuration from their work areas in either of two ways:
 - Use the **hcm get -recursive** operation. If the removed configuration was part of a module hierarchy, when users again fetch the hierarchy (using an

- hcm get –recursive** operation), the get operation deletes the contents of the removed configuration from their work areas.
- Use the DesignSync **rmfolder** and **rmfile** commands. If the removed configuration was not part of another module's hierarchy, users can use the DesignSync **rmfolder** and **rmfile** commands to remove the configuration's files and folders from their work areas. See *Deleting Design Objects in DesignSync Help* for information.

In Shirley's case, the CPU@Shirleyconf configuration is not part of another module's hierarchy. After removing CPU@Shirleyconf from the server, Shirley no longer wants the its contents taking up space in her work area. So she uses the DesignSync **rmfolder** and **rmfile** commands to remove its files and folders.

Related Topics

A Designer Creates a Configuration for Experimentation

hcm rmconf Command

The ALU Team Removes an Alias

The ALU team has two aliases that it uses to identify releases of the ALU module at different stages of the design process:

- ALU@SILVER identifies the Beta release of ALU. Currently, ALU@SILVER points to ALU@R13.
- ALU@GOLDEN identifies the release of ALU that is ready for other teams to use. Currently, ALU@GOLDEN points to R12.

After thoroughly testing R13, the ALU team decides that the GOLDEN alias should point to R13 instead of R12.

In addition, at this time, no other release has passed the necessary test criteria to qualify as the SILVER release. So the ALU team decides that the ALU@SILVER alias should be removed from the server.

Anne, the ALU team leader, makes the changes to the aliases. (**Note:** This scenario assumes that Anne has access privileges to remove aliases. For information, see *Access Controls on HCM Operations*.)

1. Anne first uses **hcm mkalias** to change the GOLDEN alias to point to R13:

```
% stclc
stcl> hcm mkalias -target
sync://alu.ABCo.com:2647/Projects/ALU@R13 -name GOLDEN
```

The `mkalias` operation changes the `ALU@GOLDEN` alias to point to the `ALU@R13` release.

- To confirm the alias change, Anne uses the `hcm showconfs` operation to display the configurations of the ALU module:

```
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU
```

```
Configurations of module
sync://alu.ABCo.com:2647/Projects/ALU
```

NAME	TYPE	OWNER	SELECTOR/ALIASED	RELEASE
<Default>	Branch	Anne	Trunk:Latest	
C1	Selector	Anne	valid12.06.01	
GOLDEN	Alias	Anne	R13	
R1	Release	Anne		
.				
.				
R12	Release	Anne		
R13	Release	Anne		
SILVER	Alias	Anne	R13	

- Anne uses `hcm ralias` to remove the `ALU@SILVER` alias from the server:

```
stcl> hcm ralias -target
sync://alu.ABCo.com:2647/Projects/ALU@SILVER
```

The `ralias` operation:

- Removes the `ALU@SILVER` alias from the server but does not delete the release to which the alias pointed (in this case, `ALU@R13`).

Note: The `ralias` operation does not affect the data in users' work areas. For example, suppose that `ALU@SILVER` is a submodule of `CPU@V0402` and that Anne had fetched the `CPU@V0402` hierarchy to her work area before she removed the `ALU@SILVER` alias from the server. An `hcm showstatus` operation comparing her work area to the server would show the `ALU@SILVER` alias configuration in her work area but report that it does not exist on the server.

- Detaches the ProjectSync notes that are attached to the `ALU@SILVER` alias.
- Generates a RevisionControl note. Users who have subscribed to email notification of RevisionControl notes on the `ALU@SILVER` alias or the owning module receive email that the alias has been removed.

4. Anne uses ProjectSync to delete her subscription to the release to which ALU@SILVER pointed. (The release is identified by its **Alias** value of SILVER in Anne's Subscription status summary.) Then she subscribes to email on ALU@GOLDEN. (For an example scenario, see Robert Updates Email Subscriptions.)

Notes:

- The `rmalias` operation does not remove hierarchical references that other modules have to the ALU@SILVER configuration. For this reason, it is important for users to subscribe to email notification of RevisionControl notes on such HCM operations as **hcm mkalias** and **hcm rmalias**. If users receive notification of creation and removal of aliases, they can update their hierarchical references to the aliases accordingly. (For an example scenario, see Robert Subscribes to RevisionControl Notes on HCM Operations.)
- Removing the reference to the ALU@SILVER submodule does not remove the data or contents of ALU@SILVER from users' work areas. However, when users again fetch the CPU@C1 hierarchy (using an **hcm get –recursive** operation), the operation deletes the contents of ALU@SILVER from their work areas.

Related Topics

`hcm rmalias` Command

Quick Steps

Quick Steps

This topic presents a summary of steps and commands used in Scenarios for Using HCM. This topic is intended to be a quick reminder of the commands for performing tasks with HCM and assumes you are familiar with the scenarios.

Creating a Module

Creating a Module from a DesignSync Vault

```
% stclc
stcl> hcm mkmod -target sync://alu.ABCo.com:2647/Projects/ALU -
description "hcm module for alu"
```

Related scenario: The ALU Team Creates a Module from a Vault

Creating a Module and Its Contents

1. Create the module on the server.
2. Create a folder for the module and change directory to it.
3. Fetch the module to the work area.
4. Create files and add them to the module. (The example shows the operation for a branch configuration. If you have a selector configuration, see Adding Files to a Selector Configuration.)

```
% stclc
stcl> hcm mkmod -target sync://cpu.ABCo.com:2647/Projects/PLL -
description "PLL design for chip407"
```

```
stcl> mkfolder PLL
stcl> cd PLL
```

```
stcl> hcm get -target sync://cpu.ABCo.com:2647/Projects/PLL
```

```
stcl> ci -new -comment "adding procedure notes" procnotes.txt
```

Related Scenario: The PLL Team Creates a Module and Its Contents

Working with a Single Module

Fetching the Files of a Module for the First Time

1. Show the modules on the server.

2. Show the module's configurations.
3. Fetch the files of the module configuration to the work area.

```
% stcl
stcl> hcm showmods -target sync://alu.ABCo.com:2647

stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU

stcl> hcm get -target sync://alu.ABCo.com:2647/Projects/ALU -
path /data/devel/users/Thomas/ALU
```

Related scenario: An ALU Designer Gets Files of a Module

Updating Your Work Area (Fetching the Same Module Subsequent Times)

```
% stcl
stcl> cd /data/devel/users/Thomas/ALU
stcl> hcm get
```

Related scenario: An ALU Designer Gets Files of a Module

Putting Modified Files of a Single Module Back on the Server (and Locking Them for Edit)

1. Change directory to the base directory of the module in your work area.
2. Identify unmanaged files that you want to be part of the module and check in the files. (The example shows the operation for a branch configuration. If you have a selector configuration, see Adding Files to a Selector Configuration.)
3. Put the module configuration back on the server.

```
% stcl
stcl> cd /data/devel/users/Thomas/ALU

stcl> hcm showstatus -files
stcl> ci -new -comment "adding mult8 logic" mult8.gv mult8.v

stcl> hcm put -lock
```

Related scenario: A Designer Puts Files of a Module Back on the Server

Creating a Release of a Single Module

1. Change directory to the base directory of the module configuration you want to release.
2. Identify unmanaged or modified files and check them in (using **ci -new** for unmanaged files and **hcm put** for modified files). See Adding Files to a Module

Configuration or Putting Modified Files of a Single Module Back on the Server
(and Locking Them for Edit).

3. Create a release of the module configuration.

```
% stclc
stcl> cd /data/devel/users/Anne/ALU

stcl> hcm showstatus -files
stcl> hcm put

stcl> hcm release -name R1 -norecursive -description "Initial
release of ALU"
```

Related scenario: The ALU Team Makes a Release Available

Adding Files to a Module Configuration

Identify whether the configuration is a branch or selector configuration.

```
% stclc
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU
```

Adding Files to a Branch Configuration

1. Change directory to the base directory of the configuration.
2. Identify the unmanaged files in your work area.
3. Check in the unmanaged files.

```
stcl> cd /data/devel/users/Thomas/ALU

stcl> ls -recursive -unmanaged

stcl> ci -new -comment "adding mult8 logic" mult8.gv mult8.v
```

Adding Files to a Selector Configuration

1. Change directory to the base directory of the configuration.
2. Identify the unmanaged files in your work area.
3. Check in the unmanaged files.
4. Tag the new file versions with the selector(s) for the configuration.

```
stcl> cd /data/devel/users/Thomas/ALU

stcl> ls -recursive -unmanaged
```

```
stcl> ci -new -comment "new files for mult8 logic" -branch Trunk
mult8.gv mult8.v
```

```
stcl> tag valid12.06.01 mult8.gv mult8.v
```

Related scenario: A Designer Adds Files to a Configuration

Removing Files from a Module Configuration

Identify whether the configuration is a branch or selector configuration.

```
% stclc
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU
```

Removing Files from a Branch Configuration

Use the DesignSync retire command to retire the files.

```
stcl> retire AluAddEx.v AluDelEx.v
```

Removing Files from a Selector Configuration

1. Change directory to the base directory of the configuration.
2. Make sure that all of the files in your work area are up-to-date.
3. Delete the version tag from the files you want to remove from the configuration.

```
stcl> cd /data/devel/users/Thomas/ALU
stcl> ls -recursive -report SNGHR
stcl> tag -delete valid12.06.01 file4.v file2.v
```

Working with a Module Hierarchy

Adding a Submodule to a Module Hierarchy (Creating a Hierarchical Reference)

1. Create a hierarchical reference from the upper-level module configuration to the module configuration that will be the submodule.
2. Show the module's hierarchical references to all its submodules.

```
% stclc
stcl> hcm addhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://alu.ABCo.com:2647/Projects/ALU@R1 -relpath ALU

stcl> hcm showhrefs -target
sync://cpu.ABCo.com:2647/Projects/CPU
```

Related scenario: The CPU Team References the ALU Module

Creating a Reference to an IP Gear Deliverable

1. In IP Gear, search for the deliverable and determine its deliverable number.
2. In HCM, create a hierarchical reference from the upper-level module configuration to the IP Gear deliverable.
3. Show the module's hierarchical references.

```
% stclc
stcl> hcm addhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028 -relpath IOSTAT

stcl> hcm showhrefs -target
sync://cpu.ABCo.com:2647/Projects/CPU
```

Related scenario: The CPU Team References an IP Gear Deliverable

Changing a Reference

1. Identify the module configuration you want to add a hierarchical reference to.
2. Identify the reference to the submodule configuration that you want to remove.
3. Remove the existing hierarchical reference.
4. Add the new hierarchical reference.

```
% stclc
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU

stcl> hcm showhrefs -target
sync://cpu.ABCo.com:2647/Projects/CPU

stcl> hcm rmhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://alu.ABCo.com:2647/Projects/ALU@R1 -relpath ALU

stcl> hcm addhref -fromtarget
sync://cpu.ABCo.com:2647/Projects/CPU -totarget
sync://alu.ABCo.com:2647/Projects/ALU@R2 -relpath ALU
```

Related scenario: The CPU Team Changes a Reference to a New ALU Release

Fetching the Files of a Module Hierarchy for the First Time

```
% stcl
stcl> hcm get -recursive -target
sync://cpu.ABCo.com:2647/Projects/CPU -path /dev/users/Marie/CPU
```

Related scenario: A CPU Designer Gets a Module Hierarchy

Updating Your Work Area with a Module Hierarchy (Fetching the Hierarchy Subsequent Times)

1. Change directory to the base directory for the hierarchy.
2. Update the hierarchy in the work area.

```
% stcl
stcl> cd /dev/users/Marie/CPU

stcl> hcm get -recursive -incremental
```

Putting a Module Hierarchy Back on the Server

1. Change directory to the base directory for the upper-level module.
2. Identify unmanaged files that you want to be part of the module and check in the files. (The example shows the operation for a branch configuration. If you have a selector configuration, see Adding Files to a Selector Configuration.)
3. Put the hierarchy back on the SyncServer.

```
% stcl

stcl> cd /dev/users/Marie/CPU

stcl> hcm showstatus -files -recursive
stcl> ci -new -comment "adding mult8 logic" mult8.gv mult8.v

stcl> hcm put -recursive
```

Related scenarios:

A Designer Puts a Module Hierarchy Back on the Server

A Designer Adds Files to a Configuration

Creating a Release of a Module Hierarchy

1. Change directory to the base directory for the upper-level module.
2. Show the status of the work area hierarchy as compared to the hierarchy on the server.

3. Identify unmanaged or modified files and check them in (using **ci -new** for unmanaged files and **hcm put** for modified files). See Adding Files to a Module Configuration or Putting a Module Hierarchy Back on the Server.
4. Create the release of the hierarchy.

```
% stclc
stcl> cd /dev/users/Robert/CPU

stcl> hcm showstatus -recursive -files
stcl> hcm put

stcl> hcm release -name R1 -description "Baseline release of CPU
hierarchy"
```

Related scenario: The CPU Team Creates a Release

Fetching from the Module Cache

1. Show the contents of the module cache to see if releases are there.
2. Fetch the configuration, using module cache mode.

```
% stclc
stcl> hcm showmcache -mcachepaths /A543/cache1

stcl> hcm get -recursive -target
sync://cpu.ABCo.com:2647/Projects/CPU -path /dev/users/Marie/CPU
-mcachepaths /A543/cache1 -mcachemode link
```

Note: If the DesignSync registry settings for default module cache paths and default module cache mode have been set, you need to specify the **-mcachepaths** and **-mcachemode** options only if you want to override the defaults.

Related scenario: Designers Use the Module Cache

Creating or Changing an Alias for a Release

1. Identify the release you want to create the alias for or change an existing alias to.
2. Create the alias.

```
% stclc
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU

stcl> hcm mkalias -target
sync://alu.ABCo.com:2647/Projects/ALU@R11 -name GOLDEN -
description "Ready for distribution to other teams"
```

Related scenario: The ALU Team Creates an Alias for a Release

Upgrading to HCM

1. Determine which DesignSync vault directories are good candidates to be HCM modules.
2. Upgrade each vault directory.

```
% stclc
stcl> url projects sync://mpu.ABCo.com:2647

stcl> hcm upgrade -target sync://mpu.ABCo.com:2647/Projects/MPU
```

Related scenario: The MPU Team Upgrades to HCM

Working with Configurations

Creating a Branch Configuration

```
% stclc
stcl> hcm mkconf -branch Trunk -name Shirleyconf -target
sync://cpu.ABCo.com:2647/Projects/CPU -description "WIP
configuration for R5 bug fix"
```

Related scenario: A Designer Creates a Configuration for Experimentation

Creating a Branch Configuration from a Release

1. Create a folder in your work area for the configuration and change directory to the folder.
2. Get the release to your work area.
3. Branch the files of the release.
4. Create a configuration from the branch.
5. Use a script (`clone.tcl`, in the example) to recreate the hierarchical references of the release in the new configuration. (This script is explained in The CPU Team Creates a Work-in-Progress from a Release.)

```
% stclc
stcl> mkfolder /dev/users/Robert/CPU_R5_Bug_Fix
stcl> cd /dev/users/Robert/CPU_R5_Bug_Fix

stcl> hcm get -target sync://cpu.ABCo.com:2647/Projects/CPU@R5

stcl> mkbranch -recursive R5_Bug_Fix .

stcl> hcm mkconf -branch R5_Bug_Fix -name R5_Bug_Fix -target
```



```
sync://cpu.ABCo.com:2647/Projects/CPU
```

```
stcl> clone sync://cpu.ABCo.com:2647/Projects/CPU@R5  
sync://cpu.ABCo.com:2647/Projects/CPU@R5_Bug_Fix
```

Related scenario: The CPU Team Creates a Work-in-Progress from a Release

Referencing a Work in Progress Configuration

```
% stclc  
stcl> hcm addhref -fromtarget  
sync://mpu.ABCo.com:2647/Projects/MPU -totarget  
sync://alu.ABCo.com:2647/Projects/ALU -relpath ALU
```

Related scenario: The MPU Team References the ALU Work in Progress

Referencing a Tools Module

These steps create a module configuration and add it to a hierarchy, but specify that its contents never be fetched to users' work areas. This method is useful for including a tool in a module hierarchy, not to manage its files but to track notes or defects on it as part of the ProjectSync project associated with the module.

1. Create the module.
2. Create the configuration.
3. Add the hierarchical reference from the upper-level module to the submodule but specify that contents will never be fetched (`-relpath ""`).

```
% stclc  
stcl> hcm mkmod -target  
sync://cpu.ABCo.com:2647/Projects/DesignCompiler -description  
"hcm module for Design Compiler tool"  
  
stcl> hcm mkconf -target  
sync://cpu.ABCo.com:2647/Projects/DesignCompiler -name v4.3  
  
stcl> hcm addhref -fromtarget  
sync://cpu.ABCo.com:2647/Projects/CPU -totarget  
sync://cpu.ABCo.com:2647/Projects/DesignCompiler@v4.3 -relpath  
""
```

Related scenario: The CPU Team References a Tools Module

Removing a Module

Removing a Module and Configurations, Its Contents, and Notes

1. Identify the module's configurations, since they will be removed with the module.
2. Remove the module.

```
% stcl
stcl> hcm showconfs -target
sync://mpu.ABCo.com:2647/Projects/TEST

stcl> hcm rmmmod -target sync://mpu.ABCo.com:2647/Projects/TEST -
vaultdata -notes
```

Removing a Module but Not Its Configurations, Contents, or Notes

```
% stcl
stcl> hcm rmmmod -target
sync://mpu.ABCo.com:2647/Projects/Scripts
```

Related scenario: The MPU Team Removes a Module

Removing a Configuration

1. Identify the configuration you want to remove.
2. Remove the configuration.

```
% stcl
stcl> hcm showconfs -target
sync://cpu.ABCo.com:2647/Projects/CPU

stcl> hcm rmconf -target
sync://cpu.ABCo.com:2647/Projects/CPU@Shirleyconf
```

Related scenario: A Designer Removes a Configuration

Removing an Alias

1. Identify the alias you want to remove.
2. Remove the alias.

```
% stcl
stcl> hcm showconfs -target
sync://alu.ABCo.com:2647/Projects/ALU

stcl> hcm rmalias -target
sync://alu.ABCo.com:2647/Projects/ALU@SILVER
```

Related scenario: The ALU Team Removes an Alias

HCM Administration

Overview of Administration Tasks

After you install the HCM software, you need to perform some tasks to set up HCM for use and to ensure that it operates successfully:

- Add the `hcmNoteAttach` trigger. You must perform this task on each HCM SyncServer. See [Adding the hcmNoteAttach Trigger](#).
- Upgrade DesignSync REFERENCES to HCM modules, if necessary. You should perform this task in order to take advantage of HCM capabilities, such as hierarchical queries and subscriptions. For an example scenario, see [The MPU Team Upgrades to HCM](#).
- Set up email notification of HCM RevisionControl notes. The HCM installation process performs the first step of this task (enabling HCM RevisionControl notes). For more information, see [Setting Up Email Notification of HCM RevisionControl Notes](#).
- Map note types, if necessary. You should perform this task if submodule SyncServers use note types different from those of the upper-level module's SyncServer. For information, see [Mapping Note Types](#).
- Store logins, if necessary. You should perform this task if your users do not have the same login on all SyncServers they access in their design work.

A simple test of whether you need to store logins is to set up the module hierarchy and then perform a query. If the query succeeds, then you do not need to store logins. For more information, see [Storing Logins](#).

- Decide on and implement a policy regarding access to hcm commands. For information see [Access Controls on HCM Operations](#).
- Optionally, customize the display of HCM commands and command options in the DesignSync GUI. For information, see [Commands and Command Options in SyncAdmin Help](#).
- Optionally, set up a module cache. For information, see [Setting Up a Module Cache](#).
- Consider how DesignSync settings affect the behavior of HCM operations and adjust settings if necessary. In most cases, HCM follows DesignSync settings. For example:
 - Minimum checkin comment length. The **hcm put** operation follows this setting. For information about the minimum checkin comment length setting, see *ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide: The General Tab-Administrator View*.
 - The registry setting for keeping version identifiers (`RmVaultKeepVid`). The **hcm rmmmod -vaultdata** operation follows this setting. For information about this setting, see *ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide: Keep Last-Version Information (RmVaultKeepVid)*

- The registry setting for choosing to delete or retain empty folders during a populate (`PopulateNoEmptyDirs`). The **hcm get** operation follows this setting. (**Note:** If this registry setting is not specified, the default behavior of the get operation is to remove empty directories.) For information about this setting, see *ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide: Advanced Empty Directory Defaults (PopulateNoEmptyDirs)*.
- The registry setting for Cadence non-collection members (`Cadence View Folder`). The **hcm get** and **hcm put** operations follow this setting when working on modules that include Cadence collections. For information about this setting, see *ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide: Advanced Registry Settings*.

However, HCM does not follow these DesignSync settings:

- Default fetch state. HCM does not support the default fetch state of **Always point to the latest version (Mirror)**. If a default fetch state of Mirror has been set in DesignSync, you must change that setting at the site or project level in order for users to use the **hcm get** or **hcm put** operations. For information about the default fetch state setting, see *ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide: The General Tab-Administrator View*.
- Settings that specify dereferencing of symbolic links to files or directories.

Note: To operate on symbolic links to files or directories, the **hcm put** operation requires links to be both under DesignSync management and modified.

The table shows how each symbolic link setting is handled:

For this symbolic link setting...	The hcm put operation...
<p>Dereference links and operate on the file/directory pointed to by the link (the default setting for symbolic links)</p>	<p>Obeys the symbolic link setting for links to files only if the link is managed and has been modified. Otherwise, the put operation does not follow the link.</p> <p>The put operation does not follow links to directories. In HCM, the contents of a configuration reside on a single server in a single vault folder. Symbolic links might point to directories containing objects that do not reside in the vault folder of the configuration. If the put operation</p>

	followed directory links, problems with maintaining configuration content could occur.
Store links to files as links... Dereference links to directories...	Checks in links to files as links, as specified by the setting. The put operation does not follow links to directories. In HCM, the contents of a configuration reside on a single server in a single vault folder. Symbolic links might point to directories containing objects that do not reside in the vault folder of the configuration. If the put operation followed directory links, problems with maintaining configuration content could occur.
Store both links to files and directories as links and remember where they point to on check out	Checks in both links to files and links to directories as links, as specified by the setting.

Note: Symbolic link settings do not affect the **hcm get** operation.

For information about the symbolic link mode setting, see The Symbolic Links Tab in SyncAdmin Help.

Enabling Legacy Module Support

In order to take advantage of the functionality of HCM modules, you must enable legacy module mode.

To enable legacy module mode:

1. Open SyncAdmin.
2. Open the **General** -> **Modules** tab.
3. Check the **Activate legacy hcm command set** option to enable legacy module mode and click **Apply** or **OK**.
4. Reset the DesignSync server and restart any open DesignSync clients.

Legacy Module Triggers and Email Notification

Legacy modules have two predefined triggers that synchronize notes attached to an alias with those attached to the module selector or release configuration to which the alias points.

Note: Modules do not use these predefined triggers.

You should disable both the triggers in order to stop Release or Alias note attachment activity after you've upgraded your modules.

This Trigger...	Is Enabled When...
hcmNoteAttach	<p>You add the trigger to the Note Objects triggers. See the steps in the section To add the hcmNoteAttach trigger.</p> <p>To disable the trigger, use the DesignSync Web interface to edit the trigger and deselect its Active property. See <i>ENOVIA Synchronicity DesignSync Data Manager: Editing Note Object Triggers</i> for information.</p>
hcmMakeAlias	<p>The first time the hcm mkalias operation is performed. You do not need to take any action to enable this trigger.</p> <p>To disable the trigger, include the following line in a server-side script and run the script:</p> <pre data-bbox="495 903 1031 934">trigger disable hcmMakeAlias</pre> <p>To enable the trigger again (after disabling it), include the following line in a server-side script and run the script:</p> <pre data-bbox="495 1081 1015 1113">trigger enable hcmMakeAlias</pre> <p>For information about server-side scripts, see Working with Server stcl Scripts in the Synchronicity stcl Programmer's Guide.</p>

If you use legacy modules concurrently with new modules, you may need to create email notification triggers for legacy module notes. To do so, you must add the hcmNoteAttach trigger and create email notification for legacy module actions.

To add the hcmNoteAttach trigger:

Note: These steps assume that the email trigger has already been activated. See *ENOVIA Synchronicity DesignSync Data Manager: Activating the Email Trigger*.

1. Invoke the DesignSync Web interface.
2. In the **Admin | Triggers** menu, click **Add**. The DesignSync web displays the Add Trigger: Define Attributes panel.
3. At the Add Trigger: Define Attributes panel, in the **Name** field, type:
hcmNoteAttach.
4. In the **TCL File** field, type: hcmNoteAttachTrigger.tbc.
5. For **Command**, select **attach**.

6. Select **Atomic**.
7. In the **Object Type** field, select **Project Configuration**.
8. Click **Submit**.

This adds the trigger to its list of triggers and activates the trigger. The Operation Successful panel appears with a table that summarizes the properties of your new trigger.

To enable email notification of legacy module actions:

1. Configure email notifications in the DesignSync Web interface. See the *DesignSync Data Manager Administrator's Guide: Configuring Email Notifications* for details.
2. From the **Email Administrator Index of Operations** page, select the **Revision Control** link under **Note Type Specific Email Formatting Attributes**. DesignSync displays the **Revision Control Specific Settings** panel.
3. For the **Fields Included in Notifications** field, select **Tag or HCM sub-cmd** from the pull-down menu.
4. Click **Review** to review your changes. Then click **Commit Changes** to have the settings take effect.

Users can then subscribe to email for RevisionControl notes, as they would for other note types. The email contains the RevisionControl note's contents and a list of objects (the URL of the relevant legacy modules, or legacy configurations affected by the operation). For an example of a legacy module scenario, see *The CPU Team Subscribes to Email on a Hierarchy*.

Mapping Note Types

When HCM is installed on a SyncServer, users can use ProjectSync to query for notes on an entire module hierarchy (an HCM module and its submodules), regardless of where submodules reside. However, submodule servers may have different ProjectSync note types, which may contain different field names and field values. If submodule servers use different note types, you, as Synchronicity administrator, must map the upper-level module note types to those used on the submodule servers. Mapping note types ensures the successful operation of not only queries but also subscriptions to notes on a module hierarchy.

Mapping note types is required in order to successfully query for notes on a module hierarchy containing IP Gear deliverables. By mapping a ProjectSync note type (SyncDefect, for example) to an IP Gear Ticket (a note based on the ProjectSync CustTicket note type), you ensure that a query on SyncDefects displays Tickets for the IP Gear deliverable as well. For example mappings, see the `<SYNC_DIR>/share/config/hcmNoteMap.conf` file.

Note: Mapping of note types ensures successful queries and subscriptions to a module hierarchy only when the user's login is the same as on the referenced server, or when a login has been stored for the user. See Storing Logins for information.

To map note types:

1. Use the **hcm showhrefs** operation to identify submodules that reside on servers different from the upper-level module. For example:

```
stcl> hcm showhrefs -target sync://cpu.ABCo.com:2647/Projects/CPU
Target: sync://cpu.ABCo.com:2647/Projects/CPU
```

REFERENCE PATH	URL	RELATIVE
ALU@R1	sync://alu.ABCo.com:2647/Projects/ALU@R1	ALU
CACHE@R4	sync://cpu.ABCo.com:2647/Projects/CACHE@R4	CACHE
Deliverable/1028	sync://ipgsrvr1.IOCo.com:2647/Deliverable/1028	IOSTAT

2. On the upper-level module's SyncServer, use the ProjectSync NoteType Manager to examine the note type structure. At the NoteType Manager, choose **Modify an active note type** and use information in the first two screens displayed.
3. For each submodule on a different SyncServer, log in to that server. Then use ProjectSync NoteType Manager to view note types and their fields. (To get information on choice and state machine field values, use the Property Type Manager on the NoteType Manager panel.)

From the information displayed, determine the note types and fields you need to map. To complete this step, you will have to know the semantics of how fields are used.

For example, the **hcm showhrefs** operation for the CPU module's default configuration lists the IOSTAT submodule as residing on an IP Gear Publisher server. You know that the Publisher server uses the Ticket note type (based on the ProjectSync CustTicket note type). You want a query based on the DateFixed field of the SyncDefect note type to use the ClosedDate field in the CustTicket note type.

View SyncDefect 2

id	Creation Date	Author
2	2004-09-30 10:50:54	Robert Bruce
Title	Problem with integration	

Project [CPU](#)

Configuration [Rel2.0](#)

Attached To

State closed

Severity serious

Responsible Robert Bruce

Platform SunOS

Browser Netscape 4.X

Customer Synchronicity

Class sw-bug

Date Fixed 2004-09-30 10:51:57

View Ticket

Ticket Id	Date Submitted	Author
31	2004-03-22 11:32:16	Master Administrator

Externally Visible Attributes

Attached To [System Behavioral Model \[1.2\]](#)
[sUSB-HC \[1.0\]](#)

Group/Site [DSP Group](#)

Title No databook

Ticket State closed

Priority Medium

Closed Date 2004-03-22 11:32:54

Customer CC List

Supporting Data

Resolution Data

Customer Dialog Original text by **masteradmin** on 2004-03-22 11:32:15-0500 :
No databook.

Ticket Type Problem Report

Analyst Assigned Analyst for DSP Group

Waiting On Nobody

From this information, you know that on the `sync://cpu.ABCo.com:2647` server, you must map:

- The SyncDefect note type to the CustTicket note type
 - The DateFixed field to the ClosedDate field
4. On the SyncServer where the upper-level module resides (`sync://cpu.ABCo.com:2647` in the example), create a configuration file named `hcmNoteMap.conf` to contain note type maps. (**Note:** The upper level module's SyncServer has the SyncDefect note type installed.)

To create the file, copy the Synchronicity-supplied file `<SYNC_DIR>/share/config/hcmNoteMap.conf` to the custom area of your Synchronicity installation directory.

To apply values to...	Copy the hcmNoteMap.conf file to this location...
Your entire site	<code><SYNC_SITE_CUSTOM>/share/config/hcmNoteMap.conf</code> (The default definition of the environment variable <code>SYNC_SITE_CUSTOM</code> is <code><SYNC_CUSTOM_DIR>/site</code>)
A specific SyncServer	<code><SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/config/hcmNoteMap.conf</code> where <code><host></code> and <code><port></code> are the server's name and port number (The default definition of the environment variable <code>SYNC_CUSTOM_DIR</code> is <code><SYNC_DIR>/custom.</code>)

Note: Do not edit the Synchronicity-supplied `<SYNC_DIR>/share/config/hcmNoteMap.conf` configuration file. Any changes you make to this file will be lost upon upgrading your SyncServer.

5. Modify the site or server `hcmNoteMap.conf` file you created to map note types, fields, and values. (The file contains syntax and examples to help you.)

For the example (Step 3) showing that the `SyncDefect` note type should be mapped to the `CustTicket` note type, you would edit the `hcmNoteMap.conf` file on the CPU module's SyncServer (`sync://cpu.ABCo.com:2647`) and add the following lines:

```
lappend SyncNoteTypeMap(sync://ipgsrvr1.IOCo.com:2647) { \
localntype SyncDefect \
refntype CustTicket \
localfield DateFixed \
reffield ClosedDate}
```

Notes:

- Specify the server URL exactly as it appears in the output of the **hcm showhrefs** operation, minus the module path.
 - For additional examples, see the `hcmNoteMap.conf` file.
6. Reset the server. In the Server section of the ProjectSync menu, select **Reset Server**. For more information, see ProjectSync Help: Resetting the SyncServer.

The mapping takes effect the next time a query is performed.

Limitations of Note Type Mapping:

- You can map multiple elements (note types, fields, or values) on the origin server to one element on the referenced server. But you cannot map one element on the origin server to multiple elements on the referenced server. For example, you can map the BugReport and SyncDefect note types on the origin server to the ProblemReport note type on the referenced server. But you cannot map the SyncDefect note type on the origin server to both the ProblemReport and TroubleNote note types on the referenced server.
- You cannot map substrings for text searches. For example, suppose that on the `sync://cpu.ABco.com:2647` server you map the SyncDefect note type's EVALUATED keyword to the REVIEWED keyword of the BugReport note type on `sync://srvr1.ABco.com:2647`:

```
lappend SyncNoteTypeMap(sync://srvr1.ABco.com:2647) { \  
localntype SyncDefect \  
refntype "BugReport" \  
localfield KeyWords \  
valuemap {EVALUATED REVIEWED}
```

A Standard Query of SyncDefect for the CPU module, specifying **This object and one level below** or **This object and all levels below**, plus Key Words EVALUATED will not yield any matches.

Related Topics

[hcm showhrefs Command](#)

Storing Logins

In order for DesignSync Web UI Query operations to work on HCM submodules, the SyncServer for the upper-level module (the origin server) must have authorization to connect to servers where submodules reside (referenced servers). (Authorization takes place through the ProjectSync login mechanism, which uses a username and password.)

In many cases, you do not need to set up stored logins for the Query operations to work. You do not need to store a login if users within a corporation have the same username and password for the different servers they access.

You must store a login if a referenced server requires a login different from the one a user uses to connect to the origin server. For example, if your login on your team's cpu server (the origin server) is `john`, but your login on the alu server (the submodule server) is `jdonne`, you need to store a login on the cpu server for the alu server. If you do not store a login on the origin server, Query operations on a module hierarchy fail.

As Synchronicity administrator or project leader, you can use the **hcm addlogin** operation to store:

- A specific login for a user to access a submodule server
- A specific login for all users of the server to access a submodule server
- A default login the SyncServer uses when no stored login exists for a submodule server

To store a login:

1. Identify the SyncServers on which submodules reside. To display a list of submodules and their URLs, use the **hcm showhrefs** operation on the upper-level module:

```
% stcl
stcl> hcm showhrefs -recursive -report brief -target
sync://cpu.ABCo.com:2647/Projects/CPU
```

2. Use the **hcm addlogin** operation to store a login for a specific server. For example, suppose John (with username `john` on the `cpu` server) wants to store a login to access the `alu` server, where he has the username `jdonne`. He would enter:

```
stcl> hcm addlogin -fromtarget sync://cpu.ABCo.com:2647 -
totarget sync://alu.ABCo.com:2647 -fromuser john -touser
jdonne
```

Notes:

- The value you specify for the **-totarget** option must be exactly the same as the value displayed by **hcm showhrefs** operation.
- Each user for whom you are storing a login must have an account (username/password) on the submodule server. In the example, user `john` already has a `jdonne` account on the `alu` server.

To store a login that allows all users on the `cpu` server (other than those that have specific stored logins) to query for notes on the `alu` server, specify the **-fromallusers** option. For example, suppose as Synchronicity administrator for the `cpu` server, you want to set up a login that all `cpu` users can use to access the `alu` server. On the `cpu` server you would enter:

```
stcl> hcm addlogin -fromtarget sync://cpu.ABCo.com:2647 -
fromallusers -totarget sync://alu.ABCo.com:2647 -touser
cputeam
```

Note: The account (`cputeam` in the example) must exist on the submodule server for the stored login to work.

To store a login that the query uses on all referenced servers instead of reusing the user's own login, use the **-toalltargets** option instead of the **-totarget** option. For example:

```
stcl> hcm addlogin -fromtarget sync://cpu.ABCo.com:2647 -
fromallusers -toalltargets -touser guest
```

This example stores the login as a kind of fallback login that the `cpu` server uses whenever it contacts submodule servers that don't have a specific login stored.

Note: For this stored login to work completely, the `guest` account first must exist on each referenced server.

3. When the **hcm addlogin** operation prompts you, supply the password. For example:

```
stcl> hcm addlogin -fromtarget sync://cpu.ABCo.com:2647 -fromuser john
-totarget sync://alu.ABCo.com:2647 -touser jdonne
Password for jdonne@alu: ****
Login stored.
User john on server cpu:2647 will contact referenced server
sync://alu:2647 using username 'jdonne'.
stcl>
```

Modifying a Stored Login

To modify a stored login, use the **hcm addlogin** operation to store the login again. For example, suppose John (with username `john` on the `cpu` server) stored a login to access the `alu` server with the username `jdonne`. John now wants to change the login he stored to use the username `tester`. To make the change, he uses **hcm addlogin** and specifies the new username:

```
stcl> hcm addlogin -fromtarget sync://cpu.ABCo.com:2647 -totarget
sync://alu.ABCo.com:2647 -fromuser john -touser tester
Password for tester@alu: ****
Stored login changed.
User john on server cpu:2647 will contact referenced server sync://alu:2647
using username 'tester'.
stcl>
```

Listing a Server's Stored Logins

To list all stored logins that a server uses to access other referenced (submodule) servers, use the **hcm showlogins** command. For example, to show the logins stored for the `cpu` server to access other submodule servers:

```

stcl> hcm showlogins -fromtarget sync://cpu.ABCo.com:2647
fromuser  totarget                touser
-----
john      sync://alu.ABCo.com:2647  jdonne
ALLUSERS  sync://alu.ABCo.com:2647  cputeam
ALLUSERS  ALLTARGETS                 guest

```

In this example:

- Item 1 shows that on the cpu server, user `john` has a stored login (`jdonne`) for querying the alu server.
- Item 2 shows that all users on the cpu server (except `john`) query the alu server with the stored login `cputeam`. (The from user `ALLUSERS` usage indicates that the **-fromallusers** option was used.)
- Item 3 shows that the cpu server uses the `guest` login whenever it contacts submodule servers that don't have a login. (The fromuser `ALLUSERS` usage indicates that the **-fromallusers** option was used. The totarget `ALLTARGETS` usage indicates that the **-fromalltargets** option was used.)

Removing a Stored Login

To remove a stored login:

1. Use the **hcm showlogins -report** command operation to display the stored logins. This option displays the logins in a form that is easy to cut and paste into your **hcm rmlogin** command. For example:

```

stcl> hcm showlogins -fromtarget sync://cpu.ABCo.com:2647 -
report command
-fromtarget sync://cpu.ABCo.com:2647 -fromuser john -
totarget sync://alu.ABCo.com:2647 -touser jdonne
-fromtarget sync://cpu.ABCo.com:2647 -fromallusers -
totarget sync://alu.ABCo.com:2647 -touser cputeam
-fromtarget sync://cpu.ABCo.com:2647 -fromallusers -
toalltargets -touser guest

```

2. Use the **hcm rmlogin** operation to remove each login you don't want.

To remove a stored login of a specific user for a specific server:

```

stcl> hcm rmlogin -fromtarget sync://cpu.ABCo.com:2647 -
fromuser john -totarget sync://alu.ABCo.com:2647

```

To remove a stored login that allows all users on the origin server to query the referenced (submodule) server:

```
stcl> hcm rmlogin -fromtarget sync://cpu.ABCo.com:2647 -  
fromallusers -totarget sync://alu.ABCo.com:2647
```

To remove a stored login that allows the origin server to query all referenced (submodule) servers:

```
stcl> hcm rmlogin -fromtarget sync://cpu.ABCo.com:2647 -  
fromallusers -toalltargets
```

Note: As Synchronicity administrator or project leader, you can set up access controls on the **hcm addlogin**, **hcm rmlogin**, and **hcm showlogins** operations. For information, see Access Controls on HCM Operations.

Related Topics

hcm addlogin Command

hcm rmlogin Command

hcm showlogins Command

Setting Up Email Notification of HCM RevisionControl Notes

A RevisionControl note is a ProjectSync note created automatically when a DesignSync revision control operation (or in this case, an HCM operation) takes place. The HCM installation process enables the generation of RevisionControl notes for these HCM operations: **hcm addhref**, **hcm mkalias**, **hcm mkconf**, **hcm mkmod**, **hcm put**, **hcm release**, **hcm ralias**, **hcm rmconf**, **hcm rmod**, and **hcm rmhref**.

As Synchronicity administrator, perform these steps to set up email notification of HCM RevisionControl notes:

1. The HCM installation process enables the generation of RevisionControl notes for HCM operations, so you do not need to perform this step as you do in DesignSync.

Note: Unlike DesignSync RevisionControl notes, RevisionControl notes on HCM operations cannot be enabled or disabled through the SyncAdmin RevisionControl tab.

2. Configure email notifications in ProjectSync, if you have not done so. You specify the behavior and format of email notifications using the Email Administrator. For more information see *DesignSync Data Manager Administrator's Guide: Configuring Email Notifications*.

3. Use the Email Administrator to include the **Tag or HCM sub-cmd** field among the email notification fields for RevisionControl notes:
 - a. In the **Triggers** section of the ProjectSync menu, click **Email Administrator**.
 - b. Click **Index**. Then from the Index of Operations, select the **Revision Control** link under **Note Type Specific Email Formatting Attributes**. ProjectSync displays the **Revision Control Specific Settings** panel.
 - c. For the **Fields Included in Notifications** field, select **Tag or HCM sub-cmd** from the pull-down menu.
 - d. Click **Review** to review your changes. Then click **Commit Changes** to have the settings take effect.

Users can then use ProjectSync to subscribe to email for RevisionControl notes, as they would for other note types. The email contains the RevisionControl note's contents and a list of objects (for HCM operations, the URL of the relevant modules and configurations affected by the operation). For an example scenario, see [The CPU Team Subscribes to Email on a Hierarchy](#).

Access Controls

Access Controls on HCM Operations

By default, HCM software allows all users access to HCM operations except for **hcm rmalias**, **hcm rmconf**, and **hcm rmod** operations. (For these operations, access is denied to everyone by default.)

As Synchronicity administrator, you can customize access controls for all HCM operations. For information, see the [Synchronicity Access Control Guide](#).

Module Caches

Setting Up a Module Cache

To reduce fetch time of large configurations and to save disk space, a design team can set up one or more module caches. The caches contain released configurations for the design the team is working on. Designers can then fetch those releases from the cache or link to them to save disk space. (For a scenario describing ways to use a module cache, see [Designers Use a Module Cache](#).)

Note: A module cache is different from a DesignSync cache. See [A Comparison of Module Caches and DesignSync Caches](#) for information.

To set up a module cache:

1. Create a directory to contain the cache. For example:


```
% mkdir /A543/cache1
```

- From your UNIX shell, set the permissions on the cache directory to read-only. Grant write access only to the person who will update the cache (the owner), usually the team leader or release librarian. For example, if your team has UNIX groups set up to control access, you might specify:

```
% chmod 750 /A543/cache1
```

- Change directory to the cache directory. Then use the **hcm get -recursive** command to fetch each release to the module cache. Use the **-path** option to specify a base directory directly below the module cache directory. (If the base directory is more than one level below the cache directory, the get operation does not find the release in the module cache.) For example:

```
% cd /A543/cache1
% stcl
stcl> hcm get -recursive -target
sync://cpu.ABCo.com:2647/Projects/CPU@Rel1 -path CPU
stcl> hcm get -recursive -target
sync://srvr1.ABCo.com:2647/Projects/STDLIB@RelA -path
STDLIB
```

The get operation:

- Fetches the CPU@Rel1 configuration to the work area directory specified with the **-path** option. If the specified path does not exist, the get operation creates it. Then the get operation fetches the STDLIB@RelA configuration in the same way.
- Because the **-recursive** option was specified, the get operation fetches all of the files for the module and its submodules to the module cache.

Note: You do not have to use **hcm get -recursive** to fetch the entire hierarchy of the release to the module cache. You can instead use a nonrecursive get operation to fetch only the upper-level module of the release. However, for a release to be fetched from the module cache, the hierarchy of that release in the module cache must match the hierarchical nature of the **hcm get** operation being performed. Otherwise, the user's get operation will not find the release in the module cache. For example, suppose the cache contains only the upper-level module of the release (fetched to the cache using **hcm get** without **-recursive**). If a user specifies a recursive fetch of a configuration, the get operation determines that the entire hierarchy of the release is not in the module cache and fetches that release from the server.

To make sure that users' fetches from the module cache succeed, whether they are recursive or nonrecursive, you might create two caches, one with recursively fetched releases and one with nonrecursively fetched releases. Then you might set the default module cache paths to include both cache paths. As an alternative to setting up two module caches, you might have one module cache but follow a naming convention that identifies releases fetched recursively from those that were not. For example, you might recursively fetch Chip@Rel1 to path Chip.hier and nonrecursively fetch Chip@Rel1 to path Chip.nohier.

4. Optionally, set a module cache mode or list of module cache paths that the **hcm get** operation uses by default. If you set up these defaults, users need to specify the **-mcachepaths** and **-mcachemode** options only if they want to override the defaults. For information, see [Setting the Default Module Cache Path or Mode](#).

To set up another module cache, repeat these steps.

To view the hierarchy of the release in the module cache, use the **hcm showmcache** operation. For example:

```
% stcl
stcl> hcm showmcache -mcachepaths /A543/cache1

Mcachepaths search order:
/A543/cache1

Configurations found:
PATH                                TARGET                                AVAILABLE  HIERARCHY
-----
---
/A543/cache1/CPU                    sync://cpu.ABCo.com:2647/Projects/CPU@Rel1  yes
  yes
/A543/cache1/STDLIB                 sync://srvr1.ABCo.com:2647/Projects/STDLIB@RelA  yes
  yes
```

As the cache owner, it is up to you to add releases to and remove them from the module cache as necessary. For guidelines on performing this task, see [Updating a Module Cache](#).

Related Topics

[hcm get Command](#)

[hcm showmcache Command](#)

Setting the Default Module Cache Path or Mode

The **default module cache path** is a DesignSync registry setting identifying the path to one or more module caches. The **hcm get** operation uses this path to locate the module cache when a user does not specify it with the **-mcachepaths** option.

When a user fetches a release with the get operation but does 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 cache. If neither **-mcachepaths** nor its registry setting is specified, the get operation fetches releases from the server. (**Note:** To override the default module cache paths, users can use the **-mcachepaths** option to specify the module cache paths they want.)

The **default module cache mode** is a DesignSync registry setting identifying the method (mode) that the **hcm get** operation uses to fetch releases when users do not specify the **-mcachemode** option. (The mode can be link, copy, or server.)

When a user fetches a release with the get operation but does not specify the **-mcachemode** option, the get operation uses the mode specified in the default module cache mode registry setting to determine the mode to use for the fetch. If neither **-mcachemode** nor its registry setting is specified, the mode that the get operation uses by default depends on the platform on which HCM is running:

- On Unix platforms, the get operation uses link mode by default.
- On Windows platforms, the get operation uses copy mode by default.

To override the default module cache paths, users can use the **-mcachemode** option to specify the mode they want. (**Note:** Using **-mcachemode link** is not allowed on Windows platforms.)

As a Synchronicity administrator or project leader, you can define a registry setting for the default module cache path or mode so that the setting applies to all users at a site or users working on a project. Individual users can also define these registry settings for their own use.

To set the default module cache path or mode, use the SyncAdmin tool. For information, see *ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide: Module Options*.

Related Topics:

Designers Use the Module Cache

Setting Up a Module Cache

Updating a Module Cache

hcm get Command

Updating a Module Cache

As module cache owner, you are responsible for updating the module cache with new releases and removing releases no longer referenced in configurations that your design team uses. To update the module cache, follow these guidelines:

- Subscribe for email notification of new releases of modules referenced in the configuration hierarchies that your team works with. For information, see *The CPU Team Subscribes to Email on a Hierarchy*.
- When you receive notification of a new release, use the **hcm get** operation to fetch the release to the module cache. Specify a base directory (with the **-path** option) that has a unique name and is located directly below the cache directory. **Note:** For an example of a script that determines the releases that need to be fetched to the module cache and then fetches those releases, see *Example Script for Updating a Module Cache*.
- A nonrecursive fetch of a release to the module cache must have a different base directory from the same release fetched recursively. For example, suppose you have the entire hierarchy of a release in the module cache (fetched with **hcm get -recursive**) but your team wants to create links to just the upper-level module of the release. When you fetch the release to the module cache (using **hcm get** without the **-recursive** option), you must specify a base directory that is different from the one you specified when you fetched the release recursively.
- When a release is no longer referenced in a configuration's hierarchy, you can use the DesignSync **rmfolder -recursive** command to remove the release's folder and contents from the module cache. See *Deleting Design Objects in DesignSync Help* for information.
- When releases are removed from a module cache, links from users' work areas to those releases are broken. Cache users need to update their work areas by using **hcm get -recursive** to refetch the module configurations containing the releases. This update removes any broken links. After you remove release from the module cache, it is a good idea to send email reminding cache users to update their work areas.
- When you fetch a release hierarchy to the module cache (with **hcm get -recursive**), you should guard against overwriting one release with another.

Overwriting can occur because the get operation allows you to fetch a different configuration of the same module into an existing directory. For example, if you fetch a configuration containing the submodule ALU@R3 to the module cache and the ALU directory already contains ALU@R1, the get operation overwrites ALU@R1 with the contents of ALU@R3. This behavior (the default for the get operation) lets users update their work areas just by refetching a module configuration.

In addition, in fetching to the module cache, it is possible for one or more of the release's submodules to be fetched to the top of the cache directory or even to a directory outside of the module cache directory. (This situation can occur

because the **hcm addhref** command allows specification of any relative path, even one that falls outside of the directory hierarchy of the upper-level module.) For example, some teams create module hierarchies that, when fetched to a work area, organize directories such that all of the configurations in the design are peers in a single, flat directory. (The hierarchical references all contain relative paths of the form `./<submodule>`.) Because all releases must reside as at the top level of the module cache, a fetch of such a module to the cache might overwrite releases in the cache with submodule releases being fetched to the same directory.

To identify releases with the potential of overwriting each other in the module cache, use the **hcm showhrefs** operation to review a configuration's hierarchy before fetching it to the module cache. If a configuration looks as though it may overwrite another, fetch it into a different module cache.

Example Script for Updating a Module Cache

This example script fetches the releases of a module into a module cache. The script fetches only those releases that are not in the module cache, are not available, or were not fetched to the module cache hierarchically.

You can use this example script to help you write your own module cache update script. To use this example script:

1. Write a script to call this procedure for each module for which all releases should be fetched.
2. Start a cron job to call the script.

Example Script:

```
#####
# Fetch the releases of a module into a module cache. Only
# those releases that are not in the module cache, are not
# available, or were not fetched to the module cache
# hierarchically are fetched.
#
# To use:
# 1. Write a script to call this procedure for each module
#    for which all releases should be fetched.
# 2. Start a cron job to call the script.
#####

proc populateMCache {mcachePath moduleUrl} {
    global errorInfo

    #
    # Get a set of the releases for the module by fetching the
    # configurations for the module from the server and filtering
    # out all but those with a configuration type of "Release".
    #
```

```

if [catch {hcm showconfs -target $moduleUrl -report script} configList] {
    error "Failed to get configuration list from server." $errorInfo
}
foreach {configName configPropList} $configList {
    array unset configPropArray
    array set configPropArray $configPropList
    if {[string equal "Release" $configPropArray(type)]} {
        set releaseArray($configName) $configPropList
    }
}

#
# If there are no releases for the module, we're done.
#
if {![info exists releaseArray]} {
    puts "No releases to fetch to the module cache."
    return
}

#
# Now get a list of the releases of the provided module that
# are available in the module cache and have been hierarchically
# fetched. These releases do not need to be refetched. Any
# release of the module that is not in the cache, not available,
# or non-hierarchical needs to be fetched to the module cache.
#
if [catch {hcm showmcache -mcachepaths $mcachePath -report script}
cacheEntries] {
    error "Failed to read module cache." $errorInfo
}

set availableReleases ""
foreach cacheEntry $cacheEntries {
    array unset cacheEntryPropArray
    array set cacheEntryPropArray $cacheEntry
    if {![string match "$moduleUrl*" $cacheEntryPropArray(target)]} {
        continue
    }
    if {!$cacheEntryPropArray(hierarchical) ||
!$cacheEntryPropArray(available)} {
        continue
    }
    lappend availableReleases $cacheEntry
}

#
# Get a list of the releases that need to be fetched to the module
# cache. This is done by removing from the list of releases
# available on the server, every release that is already available
# in the cache. What is left is a list of releases that are not
# available in the cache.
#
foreach cacheRelease $availableReleases {

    #
    # Since we're dealing with releases of a single module, it is
    # sufficient to compare the release names only. The release

```

ENOVIA Synchronicity DesignSync Data Manager HCM User's Guide

```
# names will be unique within the context of a single module.
# So, get the release name for the cached release and look for
# it in the list of releases from the server.
#
array unset cacheEntryPropArray
array set cacheEntryPropArray $cacheRelease
set release [file tail $cacheEntryPropArray(target)]
set releaseName [lindex [split $release @] 1]

unset releaseArray($releaseName)
}

#
# The releaseArray array now contains the releases that need to be
# fetched.
#
if {![array size releaseArray]} {
    puts "No new releases to fetch to the module cache."
    return
}

#
# Start fetching the releases that need to be fetched. Place
# each in a directory name of <module-name>@<release-name>
#
set failed 0
foreach {releaseName releasePropList} [array get releaseArray] {
    set target [join [list $moduleUrl $releaseName] @]
    set pathTail [file tail $target]
    set path [file join $mcachePath $pathTail]
    puts "\nFetching release to module cache."
    puts "\ttarget: $target"
    puts "\tpath:    $path\n"
    if [catch {hcm get -target $target -path $path -recursive -mcacheMode
server} result] {
        puts "Failed to fetch release."
        puts "\ttarget: $target"
        puts "\ncontinuing ..."
        set failed 1
    }
}

if {$failed} {
    error "Failed to fetch all releases."
}

return
}
```

Related Topics

[hcm get Command](#)

[hcm showhrefs Command](#)

Setting Up a Module Cache

Command Reference

Accessing the Command Reference

The HCM Command Reference is part of the Synchronicity Command Reference, which is available in its own help system.

When you follow this link, the Synchronicity Command Reference displays in its own window and you have full access to Contents, Index, and Search.

Getting Assistance

Using Help

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

Note:

Use SyncAdmin to change your default Web browser, as specified during Synchronicity tools installation. See SyncAdmin Help for details.

The Hierarchical Configuration Manager (HCM) has a command-line interface. You can bring up HCM command help from the *ENOVIA Synchronicity Command Reference* from the command line.

This help is available as text only within the command line interface. The invoked help displays in the standard output window.

To bring up the online help from the tool you are using, do one the following:

- Type **help <command>** from the tool you are using. This displays the full command description, usage (syntax), explanation of options, examples, and error messages. If the command contains a prefix, enclose the command in double quotes, for example: `help "hcm mkmod"` displays help for the `hcm mkmod` command.
- Type **<command> -help** from the tool you are using. This displays the full command description, usage (syntax), explanation of options, examples, and error messages.
- Type **<command> -usage** from the tool you are using. This displays the short command description and the command syntax.

The *DesignSync Data Manager HCM User's Guide* is not tied to a GUI application and is always opened as stand-alone help.

To display the *DesignSync Data Manager HCM User's Guide* in a browser:

- Enter the following URL from your Web browser:

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

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

- Enter the following URL from your Web browser:

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

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.

Finding Information in Online Help

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

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

Within each topic, there are the following navigation buttons:

- **Show** and **Hide**: Clicking these buttons toggles the display of the navigation (left) pane of WebHelp, which contains the Contents, Index, and Search tabs. Hiding the navigation pane gives more screen real estate to the displayed topic. Showing the navigation pane gives you access to the Contents, Index, and Search navigation tools.
- **<<** and **>>**: Clicking these buttons moves you to the previous or next topic in a series within the help system.

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

Related Topics

Accessing the HCM Command Reference

hcm version Command

Getting a Printable Version of Help

The *DesignSync Data Manager HCM User's Guide* is available in book format from the ENOVIA Documentation CD or the DSDocumentationPortal_Server installation available on the 3ds support website (<http://media.3ds.com/support/progdir/>). The content of the book is identical to that of the help system. Use the book format when you want to print the documentation, otherwise the help format is recommended so you can take advantage of the extensive hyperlinks available in the DesignSync Help.

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

Related Topics

Using Help

Contacting ENOVIA

For solutions to technical problems, please use the 3ds web-based support system:

<http://media.3ds.com/support/>

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

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

enovia.matrixone.help@3ds.com

Related Topics

Using Help

Index

A

Access Controls

- HCM 146

Alias

- HCM handling 25

- moving to another release 42

- naming 21

- overview 14

- removing 45

C

Configurations 9

- deleting 45

- status 40

- viewing 31

D

DesignSync

- comparing caches 28

- HCM commands 31

- with HCM 15

H

HCM 1

- access controls 146

- administration tasks 133

- command reference 155

- commands in DesignSync GUI 31

- introduction 3

- operations 5

- scenarios 51

Help

- contacting ENOVIA 159

- printing 158

- using 157

Hierarchical References 11

- removing 48

L

Legacy Modules 7

- command sets 5

- deleting 45

- enabling 135

- fetching to your work area 35

- relationship to ProjectSync projects 22

Login

- storing 141

M

Module Cache 26

compared to DesignSync caches 28

setup 146

showing the contents 44

updating 150

Module Hierarchy

viewing 34

N

Note Types

mapping 137

P

ProjectSync

projects 22

using with HCM 17

R

Releases 9

operation 23

RevisionControl Notes 145

T

Triggers 135