

CA Harvest Software Change Manager

Administrator Guide

Release 12.5



This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2013 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Technologies Product References

This documentation set references the following CA Technologies products:

- CA Harvest Software Change Manager (CA Harvest SCM)
- CA Software Delivery
- CA IT Client Automation (formerly, CA Desktop and Server Management, CA IT Client Manager)
- CA Clarity Agile
- CA Clarity Requirements

Documentation Changes

The following documentation updates have been made since the last release of this documentation:

- [Define a User-Defined Process](#) (see page 92)—Updated this section to include the security hole in UDP feature information.
- [Configuring SCM Project Lifecycle and Options](#) (see page 236)—Updated this section to include the enforce package association in PV/AV information.
- [Database Tables](#) (see page 180)—Updated this section to include the new database table information for generating the Project Dashboard Reports.
- [Copy the Harvest SCM Project](#) (see page 191)—Added this section to include the copy or move the SCM project information.
- [Verify the Project Data Movement](#) (see page 195)—Added this section to include the copy or move the SCM project information.
- [Delete the Source Harvest SCM Project](#) (see page 197)—Added this section to include the copy or move the SCM project information.

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At <http://ca.com/support>, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at <http://ca.com/docs>.

Contents

Chapter 1: Getting Started 15

CA Harvest SCM.....	15
Sample Software Development Process	16
Audience	17
Administrative Tasks	17
Before You Begin	18
Log In to the Administrator Application.....	18

Chapter 2: Understanding Projects and Lifecycles 21

Projects and Lifecycles	21
Lifecycles	22
Baselines.....	22
Repositories.....	23
Repositories in the Baseline	23
Multiple Repositories in a Baseline	24
Working Views	25
Working Views and the Baseline	26
Snapshot Views	28
Snapshot Views and Baselines	29
Snapshots and Multiple Projects.....	30
States and Views	31
Promote and Demote.....	31
Package Movement Through the Lifecycle	32
Multiple Next States.....	33
Multiple Approval Processes	34

Chapter 3: Administering Users 35

Users.....	35
Authentication and Users	36
Create a User.....	37
User Manager Utility	38
Password Policy.....	39
How to Disable and Enable User Accounts	39
How to Unlock User Accounts.....	39
Modify the Properties of a User	40
User Groups.....	41

Create a User Group.....	42
Add a User to a User Group	42
Remove a User From a User Group.....	43
Delete a User Group.....	44

Chapter 4: Administering Projects and Lifecycles 45

How to Define a Lifecycle	46
Lifecycle Considerations	46
Create a Project.....	47
Copy Project	48
How to Organize Projects.....	49
Change Project Status	49
Create a Working View.....	49
Create a State.....	50
Reorder States in a Project Lifecycle	50
Define a Process	51
Add a Process Note	52
Configure Baseline	52
Delete a Baseline.....	54
Create a Snapshot View	54
Project Lifecycle Diagram	55
Reports	56

Chapter 5: Defining Lifecycle Processes 59

Processes.....	59
Process Types	60
Process Aspects	62
Linked Processes	62
Process Access.....	63
Process Execution	63
Define an Approve Process	64
Define a Promote Process	65
Define a Demote Process	66
Define a Check-In Process	67
Check-In Properties Options	68
Define a Check-Out Process	70
Check-Out Properties Options	71
Define a Compare View Process.....	73
Define a Create Package Process	74
How to Specify Default Package and Form Names	76
Define a Cross Project Merge Process.....	77

Define a Delete Package Process	79
Define a Delete Version Process	79
Define a List Version Process	80
Define a Move Item Process	81
Define a Move Package Process	82
Define a Move Path Process	83
Define a Notify Process	84
Define a Remove Item Process	86
Define a Remove Path Process	87
Define a Rename Item Process	88
Define a Rename Path Process	89
Define a Switch Package Process	90
Define a Take Snapshot Process	91
Define a User-Defined Process	92

Chapter 6: Administering Repositories 95

CA Harvest SCM Repositories	95
File Extensions	96
Case Issues	96
File Conversion	97
Repository Access	97
Create a Repository	98
Load Repositories	99
Load a Repository	99
How to Move, Remove, or Rename an Item or Item Path	101
Create an Item Path	101
Delete an Item Path	102
Delete an Item	102
Duplicate a Repository	103
Delete a Repository	104

Chapter 7: Controlling Object Access 105

Object Access	105
Object Access Relationships	106
SCM-Level Access	107
Set SCM-Level Access	108
Object-Level Access	109
Project Access	109
State Access	110
Process Access	111
Item Path and Item Access	111

Repository Access	115
Form Type Access.....	116
Set Object-Level Access.....	117
Generate Access Reports	118
Access Summary.....	118
Access Hierarchy Summary	119
Access Hierarchy List	120
Administrator Access Summary	120

Chapter 8: Setting Up the Mail Utilities 121

Mail Utility Programs.....	121
How to Set Up hmail	121
How to Set Up hsmtp	122
Set Up Variables for the Email Subject with hsmtp	124

Chapter 9: Creating and Modifying Form Types 125

Default Form Types	125
How to Create Form Types.....	127
How to Design Form Types.....	128
Control Types	129
Multiple Page Forms	131
A Sample Compilation of Form Fields	132
Create a Form Type	133
How to Add or Modify Form Fields	135
Preview and Test a New Form Type.....	136
View or Modify Form Type Properties	137
Edit a Form Type Using the Form Wizard.....	138
Convert Form Definition Files to XML Files	139
Store the XML Templates From the Form Reference Folder	139
How to Modify a Form Type by Editing the XML File	140
Define a Multi-Column Form Page Layout	141
Define Required Text Fields	142
Define Text Field Pattern Validation	143
Define an Image Field.....	145
Define a Hyperlink Field	145
Define a Rich Label	146
Enable the Harweb-Based Form Editor	147
Redirect a Form	148
How Forms Use JavaScript Scripts.....	149
Use Initialization Scripts	150
Use Field Validation Scripts.....	151

Use Field Event Scripts	152
Print a Form Type	153
Add the Form Table to Your CA Harvest SCM Database	153
Form Type Deletion	155
Delete a Form Type From the Database	156
Delete a Custom Form Type From the Administrator Application	156

Chapter 10: Administering Data 159

Oracle Database Maintenance	159
BLOB Data Storage	160
Oracle Export	160
Oracle Import	161
SQL Server Database Maintenance	161
Relational Database Query Utility	162
Access Issues	163
hsqldb Standard Output Example	163
hsqldb Tab-Delimited Output Example	165
hsqldb Specific Report Example	166
hsqldb Calculations Example	167
hsqldb UDP Creation Example	168
Audit Log	168
Audit Resource Types	169
Audit Events	170
Resource Relationships and Levels	172
HARAUDITLOGVIEW	172
Audit Log	176
Database Tables	180
How to Copy or Move a Harvest SCM Project from One Database to Other	189
Prerequisites	191
Copy the Harvest SCM Project	191
Verify the Project Data Movement	195
Delete the Source Harvest SCM Project	197

Chapter 11: Synchronizing Reference Directories 199

CA Harvest SCM Reference Directory Synchronization	199
Terminology	199
hsync Directory Synchronization	200
Full View Approach	201
Active View Approach	202
hsync Item Selection Options	202
Check-Out Version Selection	203

File Deletion	203
When Subdirectories Are Deleted	204
Purge Option and Exclusion Lists	204
PurgeOpts and PurgeArg Tokens	205
Synchronous/Asynchronous Execution Mode	206
Use Case-hsync Stand-Alone Execution	207
hsync for Promotional File Management Example	207
hrefresh Command	209
How hrefresh Selects Items	210
hrefresh Post-Linked to Processes Use Cases	210
Server, Client, and Agent Configuration	215
How to Set Up hrefresh	215
How to Define the hrefresh Configuration File	216
hrefresh Argument File Definition	218
Semaphore Locks	220
Encrypted Password Files	221
hrefresh UDP Definitions	222

Chapter 12: Repository Cache on Remote Site 225

CA Harvest SCM Repository Cache on Remote Site Overview	225
Cache Terminology	226
How the Cache Works	226
Cache Management	226
Cache Maintenance	227
Repository Cache on Remote Site Configuration	227
Configure the Repository Cache Using the Workbench	227
Configure the Repository Cache Manually	228

Chapter 13: Administering Peer Reviews 229

Peer Review Verification and Comment Purging – hpeerreviews UDP	229
Specify hpeerreviews Parameters	230
Sample Promote Processes Using Peer Review UDPs	230
Copy and Edit Sample Processes	231
Sample UDP Input	231
Sample Verifications	231
Sample Purge Comments	233
Configure Peer Review Notification	233

Chapter 14: Configuring SCM for CA Vision Integration 235

CA Harvest SCM Project and Lifecycle Configuration	236
--	-----

Configuring SCM Project Lifecycle and Options	236
Mapping CA Vision Users to SCM Users.....	238
Associating CA Vision Product Releases with CA Harvest SCM Projects	239
Synch Server.....	240

Appendix A: Predefined Lifecycle Templates 241

Predefined Lifecycle Templates	241
Software Release Development	242
Deploy Release	242
How CA Software Delivery Integration Works	243
Uninstall and Rollback.....	246
Package Statuses and Processes	248
HARUSDHISTORY Table	249
How to Display Software Deployment Package Data Records.....	249
A Typical Scenario	250
How to Complete Forms and Form Attachments	251
USD Package Information Form	251
How to Specify USD Platform Information Parameters	253
How to Use the USD Platform Information Form or Configuration File	253
Format of the Platform Information Configuration File.....	254
USD Platform General Information	255
USD Platform Installation	256
Targets for USD Platform Information Form Only	258
Targets for USD Platform Information Configuration File, ini Only	259
Targets for USD Platform Information Configuration File, xml Only.....	260
Procedure Options	261
Job Options	263
Uninstallation Options	264
Sample USD Platform Information Configuration Files.....	265
Formal Problem Tracking	266
How Formal Problem Tracking Works.....	267
New Development	268
How New Development Works.....	269
New Development - Alternatives	269
OnDemand Project.....	270
How Administrators Set Up an OnDemand Project	271
How to Use an OnDemand Project Life Cycle	272
Packaged Application Change Management.....	274
How Packaged Application Change Management Works	274
Parallel Development.....	275
How Parallel Development Works	275

Production.....	277
How Production Works	277
How to Identify and Demote Emergency Packages	279
Check for Emergency Changes UDP	279
Check for Branch Version UDP	284
Make Part of Emer Pkg Group UDP	286
Remove from Emergency UDP	287
Validate All on Branch UDP	288
Release	291
How Release Works	292
Release with Emergency	295
How Release with Emergency Works.....	296
Standard Problem Tracking	298
How Standard Problem Tracking Works	299
Third Party Tool	299
How Third Party Tool Works	299
Version Control	300
How Version Control Works.....	301
Peer Review Sample UDPs	301

Appendix B: Using the System Variables 303

System Variables	303
clientpath and viewpath Variable Usage.....	305
Lists.....	305
Password System Variable	306

Index 309

Chapter 1: Getting Started

This section contains the following topics:

[CA Harvest SCM](#) (see page 15)

[Sample Software Development Process](#) (see page 16)

[Audience](#) (see page 17)

[Administrative Tasks](#) (see page 17)

[Before You Begin](#) (see page 18)

[Log In to the Administrator Application](#) (see page 18)

CA Harvest SCM

CA Harvest Software Change Manager (formerly known as CA Harvest Change Manager) helps you synchronize development activities throughout the application development lifecycle on multiple platforms and across your enterprise. CA Harvest SCM provides you with the following functionality:

- **Adaptability**—You can easily adapt CA Harvest SCM to use your own development lifecycle processes. You can update or change CA Harvest SCM lifecycle templates as new process improvements are identified and approved. This adaptability keeps software changes under control, schedules on track, and your team up-to-date.
- **Security**—You can control access to administrative functions and protect sensitive data. Options are provided for the secure use of command-line interface (CLI) and remote agent technology.

CA Harvest SCM authenticates users. CA Harvest SCM can control the execution of lifecycle processes and secure archived items. You can define each process in the CA Harvest SCM lifecycle to provide a flexible security model.
- **Usability**—The CA Harvest SCM graphical user interface (the Workbench) provides you with ease-of-use and minimal training time.
- **Concurrent Development Support**—More than one developer can work on the same area of code simultaneously without risk of overwriting changes. Developers can use merge utilities to resolve any conflicts between versions.
- **Parallel Development Support**—You can maintain multiple releases of the same application. For example, you can work on emergency fixes for a release and at the same time work on the next release of that application. If necessary, you can merge changes made for the emergency into the main application development cycle.
- **Change and Defect Tracking**—You can track changes and testing defects in associated packages and forms. Using reports, you can obtain current and historical information about changes and packages. Your application production environment provides a history of the development process, including specific changes and the reasons they were made.

- **Tool Integration**—Interfaces from integrated development environments (IDEs) let developers perform routine SCM functions without leaving the IDE.

CA Harvest SCM enables Workbench users to plug in third-party difference and merge tools, such as Araxis Ltd. Merge and Guiffy Software, Inc., in place of internal difference and merge tools.

- **Complete Solution**—All functional groups involved in the development and maintenance process can benefit from CA Harvest SCM, not only programmers. CA Harvest SCM can control the entire development process, including problem tracking, change management, builds, testing, quality assurance, documentation, and release. Extensive management reporting capabilities support auditing functions throughout the entire process.

Sample Software Development Process

This sample lifecycle presents a typical, simplified, software development process. The team that uses it consists of two developers, one quality assurance (QA) engineer, and one development manager. The software application being worked on is already in the market, so the purpose of this project is to support maintenance work on the current release of that software.

The five phases (states) in the development process are as follows:

1. **Assign**—Various personnel such as QA engineers, customer support personnel, and the development manager, create modification requests (packages). The development manager assigns a priority to each request and passes the request to the development team (promotes the request packages to the Coding state).
2. **Coding**—Developers determine what code must be modified. Using the packages, the developers check out the code to their respective client computers, make changes, and check in the code. When all code affected by the request is modified, the development manager approves the packages and promotes them to the Test state.
3. **Test**—The QA engineer reviews the requests and tests the code to verify that it works according to the requests. If errors are found in the code, the engineer returns the packages to the developers (demotes it to the Coding state). If no errors are found, the engineer promotes the packages to the Release state.
4. **Release**—Code that is ready to be released resides here. Packages are not demoted from this state. When the code is ready to be released a type of picture named a snapshot is taken of the code that is included in the release.
5. **Snapshots**—All snapshots reside here. Snapshots make it easy to go back and look at the content of any release.

More information:

[Projects and Lifecycles](#) (see page 21)

[States and Views](#) (see page 31)

[Working Views and the Baseline](#) (see page 26)

[Processes](#) (see page 59)

[Package Movement Through the Lifecycle](#) (see page 32)

[Snapshot Views](#) (see page 28)

Audience

This guide presents CA Harvest SCM concepts that help you understand how to perform administrative tasks, such as designing and creating a lifecycle. Step-by-step procedures guide you through the tasks you must perform to set up and maintain CA Harvest SCM.

The following persons may have specific roles or combined roles during a project lifecycle:

- Administrators use the information in this guide to create lifecycles, maintain projects, and administer CA Harvest SCM users. For example, information in this guide helps administrators design a lifecycle and custom form types.
- Database administrators use the information in this guide to help administer CA Harvest SCM database tables. For example, information in this guide helps database administrators use the archive function to move data from the CA Harvest SCM database tables to another location for archive purposes.
- Build managers use the information in this guide to build applications from source code. For example, information in this guide helps build managers manage reference directories on multiple operating systems.

The *Administrator Guide* benefits anyone who wants to understand how lifecycles and processes work in CA Harvest SCM.

Administrative Tasks

A software development process is a structure (lifecycle) imposed on the development of a software product. The lifecycle that your site uses determines the tasks or activities that take place during the software development process to control changes that are made to software throughout the lifecycle.

The administrator sets up the lifecycle and must perform some tasks before users can access and use CA Harvest SCM to manage their changes. Such tasks include setting up user groups, defining the lifecycle that you want to use, setting up the repository, and setting access permissions. Administrators use the Administrator application to perform these tasks and later to administer and maintain the software development process.

Before You Begin

You use the Administrator application to set up and administer CA Harvest SCM projects and lifecycles. Some setup tasks must be completed before you can use the Administrator application effectively. Depending on your knowledge and responsibilities, you may be able to perform some or all these tasks yourself. A system administrator may need to complete some of them.

Before you begin to set up CA Harvest SCM, do the following:

1. Install CA Harvest SCM on both the server and the client.
2. Verify that the relational database is running.
3. Verify that the CA Harvest SCM broker process is running and a server process is available.
4. Know the login credentials for the initial CA Harvest SCM user that was created during installation.

Note: For information about Steps 1 through 4, see the *Implementation Guide*.

5. (Optional, but recommended) Read the following chapters:
 - "Understanding CA Harvest SCM Basics" in the *User Guide*.
 - "Understanding Versions" in the *User Guide*.
 - "Understanding Projects and Lifecycles" in this guide.

Becoming familiar with this material, can help you design, implement, and manage your projects and lifecycles.

Log In to the Administrator Application

The Administrator application lets you use CA Harvest SCM functions that are suited to CA Harvest SCM administrators, build engineers, and other administrators and development managers.

Follow these steps:

1. Select CA Harvest SCM Administrator from the program group.
The CA Harvest SCM login dialog appears.
2. Enter your user name, password, and the broker location.
3. (Optional) Select the Save Password check box to retain the password you enter for your next CA Harvest SCM session.

4. Click OK.

The main window shows the Administrator application, which provides you with access to all administrative functions.

Chapter 2: Understanding Projects and Lifecycles

This section contains the following topics:

[Projects and Lifecycles](#) (see page 21)

[Lifecycles](#) (see page 22)

[Baselines](#) (see page 22)

[Repositories](#) (see page 23)

[Working Views](#) (see page 25)

[Working Views and the Baseline](#) (see page 26)

[Snapshot Views](#) (see page 28)

[States and Views](#) (see page 31)

[Promote and Demote](#) (see page 31)

[Package Movement Through the Lifecycle](#) (see page 32)

[Multiple Next States](#) (see page 33)

[Multiple Approval Processes](#) (see page 34)

Projects and Lifecycles

The term *project* refers to the control framework in CA Harvest SCM that supports a particular development or maintenance activity. A *lifecycle* is the part of a project that determines what activities can take place, when, and by whom. It consists of states, processes, views, and access settings. The lifecycle can be considered the heart of a project, because it controls the flow of development life in it.

More information:

[Sample Software Development Process](#) (see page 16)

Lifecycles

A lifecycle describes the path that changes take as development progresses in terms of an ordered set of states. In each state, processes define the various activities that can occur. This means that each state can have a uniquely defined scope of work. The demote and promote processes have special significance in a lifecycle because they determine how states are related and how changes can move between them. Changes travel through the lifecycle by means of packages. You can link Notify and UDP processes to the promote and demote processes so that the linked processes execute either before or after the package is moved. Data views (baseline, working views, and snapshot views) determine which item versions are accessible to users, and let administrators capture software inventories.

Baselines

When you create a project, it automatically contains an empty *baseline*. The baseline defines the software inventory that users can access. You configure the baseline to include repositories when you want to use the project to track changes. The repositories contain the physical data items that CA Harvest SCM controls. When you configure a baseline, you can select from among the available snapshot views, each snapshot view contains one or more repositories at a specific stage of development. If more than one repository was included in the snapshot view when it was created, you can add any or all of them to the baseline. If a project is designed for an activity such as problem tracking that does not require changing data, the baseline can remain empty.

Note: The name baseline is a reserved name; no other view can have this name.

The *Baseline* view is a view that CA Harvest SCM automatically creates to reflect the condition of a repository before any changes have been made to it. You can access a new repository, or an existing repository in its original condition, by selecting the Baseline view.

More information:

[Snapshot Views and Baselines](#) (see page 29)

[Configure Baseline](#) (see page 52)

Repositories

CA Harvest SCM maintains the data under its control in one or more hierarchical sets of directories called *repositories*. The term *item* refers to a component of a repository to distinguish it from a file in the external directory structure. To clarify the distinction between data outside the repository and data within it, a further convention is used—external directories are referred to as client directories. Directories within the repository are referred to as item paths, or view paths if a particular view is being referenced.

The repository contents are determined by you. You can put one or more applications in a single repository, or you can split up an application over several repositories.

A repository is independent of any particular environment and is the basis for creating baselines and views. Baselines and views can include items from more than one repository, so there is not a disadvantage to using multiple repositories.

For security and ease of access, repositories are typically located on a server.

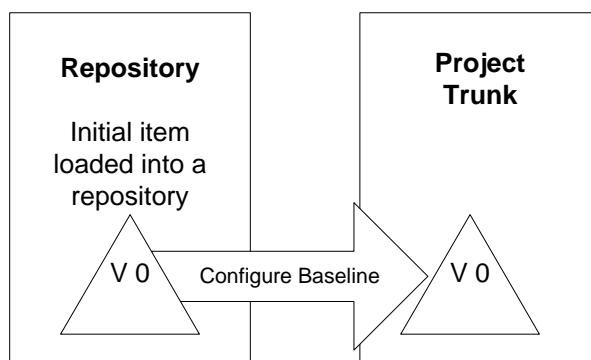
Repositories in the Baseline

When you configure a repository in the baseline of a project, CA Harvest SCM creates a trunk for each item in the repository. Each item is numbered as version zero (V0) and all versions for that project accumulate on their own trunk.

To facilitate the management of shared items, you can specify repositories in the baseline as read-only. Items in a read-only repository can be checked out for browse, but they cannot be changed.

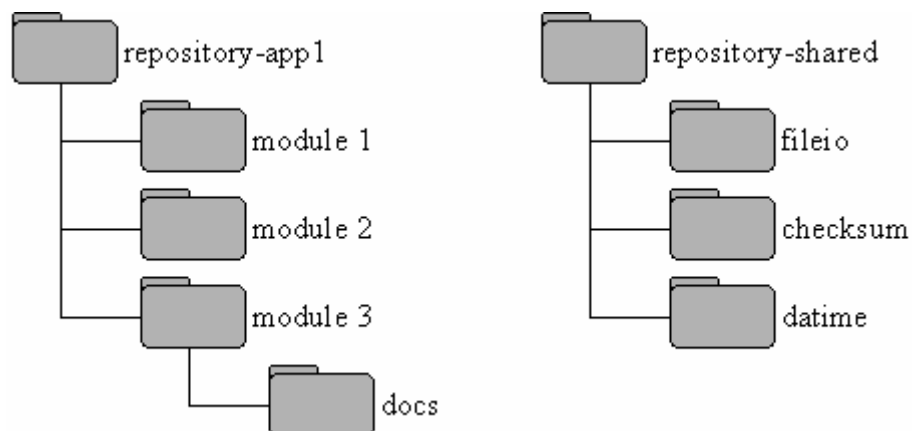
Important! Because the repository item inventory is determined when you configure the baseline, use the load repository function before you configure the baseline. If you load the repository *after* the baseline is configured, the items are *not* visible in the project.

The following illustration shows how an initial item (V0) in the repository is numbered when it becomes the initial item (V0) in a project's trunk:

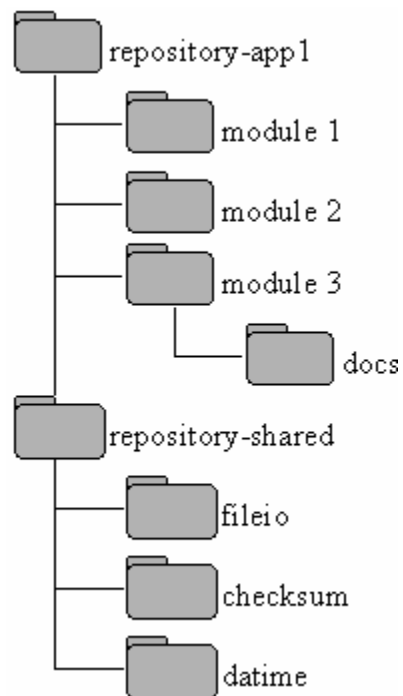


Multiple Repositories in a Baseline

A baseline that was created from two or more repositories has a top-level directory for each repository. The files under the top-level directories match the files in the corresponding repositories. For example, the following illustration shows two repositories (repository-app1 and repository-shared) with different structures:



In the following illustration, a project baseline is configured to include repository-app1 and repository-shared; these repositories are the top-level directories.

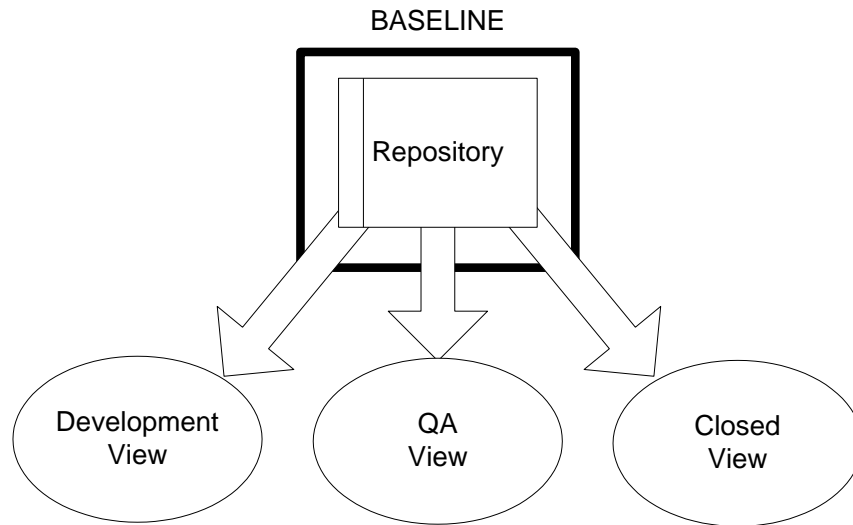


Working Views

Working views have two basic purposes:

- They allow users in a project to access data.
- They allow changes to be isolated and managed.

The number of views you create and how those views are associated with stages in the lifecycle depends largely on how you want to isolate changes. The following illustration shows a project with three working views.



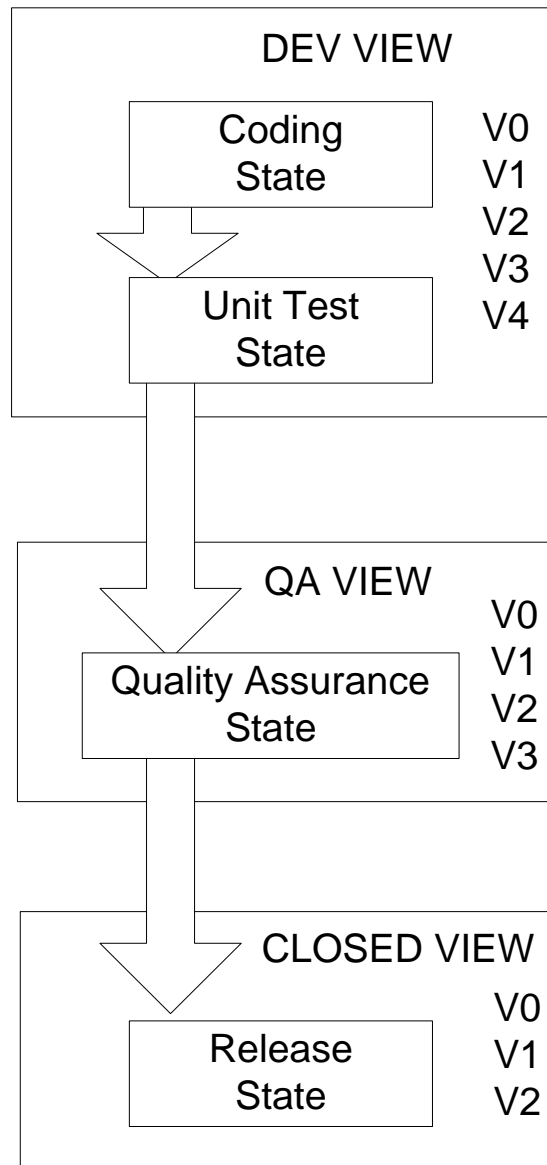
Working Views and the Baseline

Each working view in a project is based on the project's baseline, and inherits its structure and items. Each working view looks at the same paths and items as the baseline, but not necessarily the same versions of items. The ability of views to isolate versions is a part of change management. For example, a user in a Quality Assurance state can check out the latest versions of all items in the QA view to build a release of the product that only includes versions associated with packages that have been promoted to Quality Assurance.

You can create working views before or after you add a repository to the baseline. If the baseline is empty when you create the working views, and then you add a repository to the baseline, all the working views in the project are automatically updated to include the items available in the baseline. Working views that are created after a repository is included in a baseline initially include *only* the base versions of items in the repository.

Example: Working Views in a Lifecycle

The following example illustrates how a lifecycle uses working views. Four states (Coding, Unit Test, Quality Assurance, and Release) are associated with three views: DEV, QA, and CLOSED.



All these views are based on the baseline and the same items are visible in each view. Each view changes as versions are created and packages progress in the lifecycle. For example, versions created in the Coding state are initially visible only in the DEV view. When the package associated with the versions is promoted to the Unit Test state, the versions are still visible to the Coding state because Unit Test and Coding share the DEV view. When the package is promoted to the Quality Assurance state, the versions become visible in the QA view, and so on.

More information:

[Sample Software Development Process](#) (see page 16)

Snapshot Views

A snapshot view is a read-only image of a working view at a specific point in time. Snapshot views let administrators capture a software inventory at significant points in its development, such as a release. You can base snapshot views on any working view of a project, and you can capture the view at the current or previous time—the snapshot view acts as a reference to the versions in the working view that were most recent at the time specified by the snapshot view. Updating items in the working view does not affect the snapshot view, because the snapshot cannot be altered after it is created.

The All Snapshot Views option in the State Properties dialog lets you access snapshot views in the project from which they were created. When you select this option, the state has as its view all the snapshot views that exist in that project. The versions that the snapshot view captures are visible in the state and can be checked out, but because snapshot views are read-only, items cannot be checked in or modified from that state.

You can use snapshot views in the following ways:

- As a reference.
- As the baseline for other projects.
- To recreate an application as it was at the time that the snapshot was taken.
- On a temporary basis. For example, a build manager might regularly take a snapshot view of a testing view when a build is performed. If the build is successful, the build manager can delete the previous snapshot view. If the build presents new problems, the build manager can revert development to the versions captured by the previous snapshot view.

More information:

[Sample Software Development Process](#) (see page 16)

Snapshot Views and Baselines

Snapshot views achieve their greatest usefulness when they are used as the baseline view of other projects. By capturing software development at significant moments, snapshot views enable new development cycles to begin at any point in an existing lifecycle.

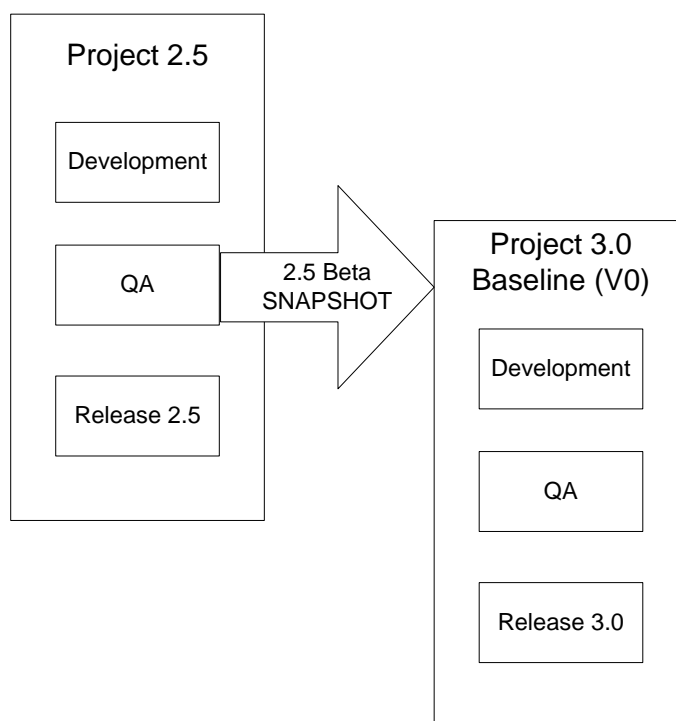
The following example shows how an administrator uses a snapshot as a baseline:

1. In Project 2.5, the administrator uses the take snapshot process to take a snapshot view of the QA working view.

The administrator names the snapshot view 2.5 Beta SNAPSHOT.

2. In Project 3.0, the administrator configures 2.5 Beta SNAPSHOT as the baseline.

The latest versions that exist in the Project 2.5 QA become the base versions (V0) in Project 3.0.



More information:

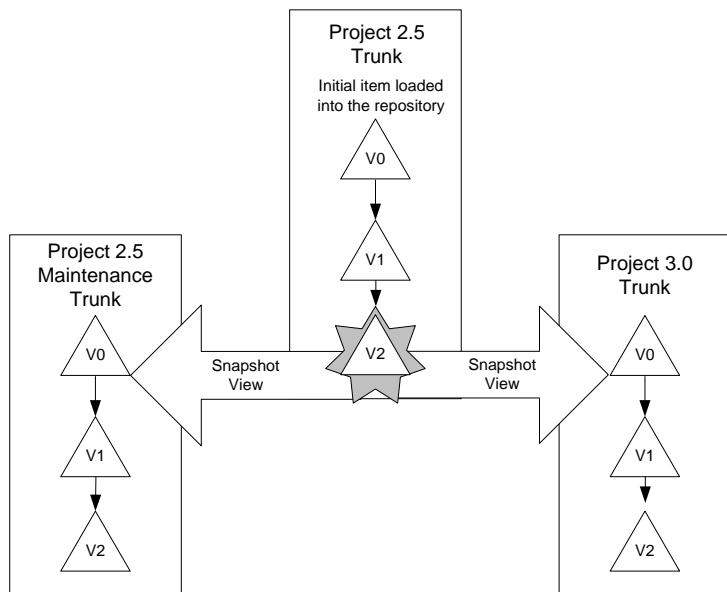
[Baselines](#) (see page 22)

Snapshots and Multiple Projects

Snapshot views let administrators create divergent projects that are initially based on the same application items. When you want the Configure Baseline dialog to list a snapshot view during the setup of other projects, select the Visible to Other Projects option when you use the take snapshot process. Typically, snapshots that represent significant phases of development are made visible.

Example: How Multiple Projects Use a Snapshot as a Baseline

In the following example, an item in Project 2.5 has versions 0, 1, and 2 (V0, V1, V2) on the trunk. The administrator takes a snapshot view of the working view that includes V2, and configures the snapshot view as the baseline for two new projects. V2 becomes the initial version (V0) in the new projects. Subsequently, executing a check-out and check-in process for the item in the new projects creates versions on the project trunks.



States and Views

States are distinct work areas in which certain activities can take place as packages move from creation to completion. A lifecycle can include any number of states. A state is typically associated with a view, which allows users to “see” the data stored in a CA Harvest SCM repository. All users in a state share one view. This setup means that they see the same items, and changes made to an item in a state are immediately visible to all other users in that state.

You can define a state without a view when access to items is not needed. For example, a lifecycle could have an initial state named Assigned Changes, which is simply a holding area for newly created change requests. This state does not have a view and its processes are limited to those processes that do not access repository data, such as approve, promote, and create package.

After you define a lifecycle, including states and views, the Workbench Explorer View shows the following objects:

- A States folder that shows the states that are defined for a project.
- A Views folder that shows the paths and items which make up the repository under a view. The items represent the logical contents of the repository; therefore, when an item has been renamed, only the current item name shows.

More information:

[Sample Software Development Process](#) (see page 16)

Promote and Demote

The order in which changes progress through the lifecycle is determined by two processes, promote and demote. A promote process moves change packages from one state to the next. A demote process returns change packages to a previous state.

When the from and to states in a promote process have different views, the second view is modified to “see” the changes associated with the promoted package. The from view is not changed by the promote process. In a corresponding fashion, when the from and to states in a demote process have different views, the from view is modified to no longer see the changes associated with the promoted package. The destination view is not changed by the demote process unless the demoted package did not previously exist in that view.

Important! If you delete a state that was the target of a demote or promote process, the process must be either deleted or redefined to point to a new target state.

Package Movement Through the Lifecycle

Packages typically travel sequentially from one state to the next in a lifecycle. Returning packages to a previous state also usually occurs sequentially. For example, consider a simple lifecycle with four states: Development, Unit Test, QA, and Release. The following promote processes move a package sequentially through this lifecycle:

1. Development to Unit Test
2. Unit Test to QA
3. QA to Release

Corresponding demote processes return a package sequentially:

1. Release to QA
2. QA to Unit Test
3. Unit Test to Development

The sequential movement of packages is the most simple and straightforward way to manage changes; however, you might want to demote a package across multiple states.

Using the current example, if QA detects and diagnoses a problem, it might be simplest to return the package directly to Development for additional work, rather than demoting it to Unit Test and then Development. This scenario is possible only if both Unit Test and Development share the same view. In this scenario, only two demote processes would be required:

1. Release to QA
2. QA to Development

Skipping states only makes sense because Development and Unit Test share the same view. Consider what would happen if they had different views. A package is promoted from Development to Unit Test and then to QA, updating each view with changes. Then the package is demoted from QA to Development. The demote process removes the changes from QA, but they still appear in Unit Test, although the current state of the package is Development.

Skipping states in the lifecycle can create a confusing situation in which package changes do not appear in the view associated with its current state.

If Development, Unit Test, and QA all have their own views, a similar situation could occur if you promote a package from Development to QA. The promote process would update QA with the package changes and change the state of the package to QA, but it would not update Unit Test. The state of the package would be QA, but its changes would not appear in the Unit Test view.

To avoid this confusion, follow this rule:

Packages should always move forward or backward sequentially through the lifecycle from one state to the next, unless two states share the same view.

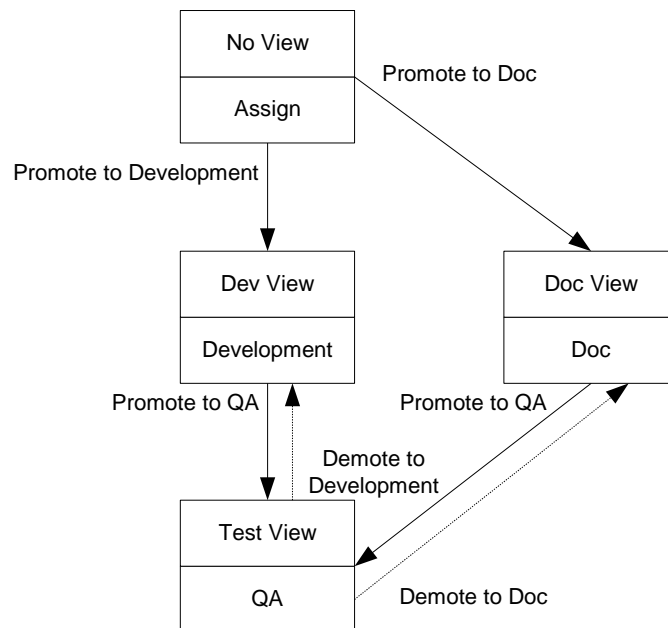
More information:

[Sample Software Development Process](#) (see page 16)

Multiple Next States

You can set up a lifecycle to move packages to more than one next state by defining multiple promote processes. You can use this setup to move packages through different paths.

The following illustration shows how users in the Assign state can send a package to either Development or Doc, depending on whether the CR pertains to software or to documentation. Packages in both Development and Doc are then promoted to QA for quality assurance testing and can be demoted if the packages need more work.



You have complete freedom in designing a lifecycle. You can create sequences that do not make sense, such as promoting a package in a circular path. You can also create highly complex lifecycles, such as a lifecycle that includes a state without a view, following a state with a view, or multiple promote processes to states with different views. Although it is possible to set up and design these lifecycles, change management in such a lifecycle becomes more complex.

Note: For more information about how packages move through the lifecycle, see the *Workbench User Guide*.

Multiple Approval Processes

You can define more than one approval process for a particular state. When you define multiple approval processes for a state, the approval processes are treated as alternatives to one another.

Typically, you can promote or move a package as soon as the requirements for the approval process are complete. In emergencies where the usual approval process cannot be completed, you can execute a special process to bypass the typical approval requirements. This second approval process then acts as an emergency override to the first approval process; you can define it to require approval by one senior-level manager.

Chapter 3: Administering Users

This section contains the following topics:

[Users](#) (see page 35)

[User Groups](#) (see page 41)

Users

Users must be defined in CA Harvest SCM before they can log in to the product. When CA Harvest SCM is installed, an initial user is automatically created. This user must add other users who need access to CA Harvest SCM.

Note: The record in the HARUSER table whose USROBJID field has a value of 1 identifies the initial user created during the installation. This user is always a CA Harvest SCM administrator and always exists in CA Harvest SCM, even if this user does not exist in the external authentication server. However, when you use external authentication, this user (like all other CA Harvest SCM users) must exist in the external authentication server to log in to CA Harvest SCM.

Creating users and user groups are closely related tasks. If you create users first, you can add them to the groups when you define the groups. If you create groups first, you can add the users to them when you define the users.

Users and user groups exist at the CA Harvest SCM level. This level means that they are available to all projects defined in a CA Harvest SCM installation. A user can belong to any number of user groups, and the groups imply no hierarchy. For example, a user in the Development Manager group does not implicitly belong to the Developer group.

Note: For more information about the CA Harvest SCM initial user, see the *Implementation Guide*.

Authentication and Users

The User Properties dialog lets you define CA Harvest SCM users.

If your site uses internal authentication (CA Harvest SCM authentication), you can edit all the fields on the User Properties dialog.

If your site uses external authentication such as Microsoft Active Directory, in the personal information area of the User Properties dialog, you can edit only the Name and Note fields. You cannot edit any other fields (Password, Real Name, Phone #, Ext, Fax# and E-mail). In the Security area, you can enable the options including Disabled and Single Work Station Login while the other options Locked and Change Password on Next Login remain disabled.

For both internal and external authentication, during login, create user, and update user operations:

- If the value of the CASESENSLOGIN field in the HARTABLEINFO table is N, the user name authentication is not case sensitive.
- If the value of the CASESENSLOGIN field in the HARTABLEINFO table is Y, the user name authentication is case sensitive.
- If your site uses external authentication, the CASESENSLOGIN field in the HARTABLEINFO table must be set to match the behavior of the external authentication server.

The following considerations apply when you create a user:

- If your site uses internal authentication, a user is created only if the user name entered does not exist in CA Harvest SCM.
- If your site uses external authentication, such as Microsoft Active Directory, a user is created only if the user name entered does not exist in CA Harvest SCM but does exist in the authentication server, or if the CA Harvest SCM server has been configured for Mixed Authentication Mode and the user name entered does not exist in CA Harvest SCM.

The following considerations apply when you update a user's name definition:

- If your site uses internal authentication, changes to the Name field are permitted only if the new name does not exist in CA Harvest SCM.
- If your site uses external authentication, such as Microsoft Active Directory, changes to the Name field are permitted only if the new name does not exist in CA Harvest SCM but does exist in the authentication server, or if the CA Harvest SCM server has been configured for Mixed Authentication Mode and the External Authentication attribute for the user is reset.

Create a User

The User Properties dialog lets you define CA Harvest SCM users.

If your site uses internal authentication (CA Harvest SCM authentication), you can edit all the fields on the User Properties dialog.

If your site uses external authentication such as Microsoft Active Directory, in the personal information area in the User Properties dialog, you can edit only the Name and Note fields. You cannot edit any other fields (Password, Real Name, Phone #, Ext, Fax# and E-mail). In the Security area, you can enable the options including Disabled and Single Work Station Login while the other options Locked and Change Password on Next Login remain disabled.

Follow these steps:

1. Click the User Groups tab of the Administrator application.
2. Right-click the Users folder, and select New User from the shortcut menu.

The User Properties dialog appears.

3. Define the properties of the user.

Note: The notify process uses this email address when this user is designated to receive notifications.

4. (Optional) Specify authentication/security properties for this user. Only a CA Harvest SCM Administrator or a user with Administrative User access can set these options:

Disabled

Disables the user account.

Single Workstation Login

Restricts the user from having multiple active sessions with the same broker from multiple workstations. However, all users (including single-workstation-login restricted users) can have multiple active sessions with the same broker from the same workstation and can also have multiple active sessions with different brokers from any combination of workstations.

External Authentication

Specifies whether the user account uses internal or external authentication.

Locked

Unlocks an internally authenticated account which has been locked because it has reached the maximum consecutive failed login attempts or the password has expired and the user is not permitted to change the password.

Change Password on Next Login

Forces the user to change the password upon the next login. This option is disabled for external authentication. For mixed mode authentication, it is enabled for internal users and disabled for external users.

5. Click the User Groups tab.

If a user already belongs to one or more user groups, those groups are displayed in the User Groups list. New users are automatically added to the Public group.

Note: The Administrator group is visible in this list only if the current user has Administrator rights. This security measure helps ensure that only an Administrator can grant Administrator rights to another user.

6. Click OK.

The user is created and appears in the Users folder.

User Manager Utility

The User Manager (husrmgr) utility is a toolkit for the CA Harvest SCM administrator who maintains user profiles in CA Harvest SCM. You execute the User Manager utility from the command line. The utility provides user maintenance functions: import user, delete user, rename user, and update user.

Note: For information about the User Manager utility, see the *Command Line Reference Guide*.

Password Policy

Command line utilities, `hpolget` and `hpolset`, provide a configuration file-based interface to CA Harvest SCM Password Policy. Use `hpolget` to generate a configuration file containing the current policy. To change policy, edit the configuration file and then run `hpolset`.

The utility, `hchu`, provides a command-line interface for changing a password. Other command-line utilities do not prompt for a new password when the user's password has expired.

Note: For details about the command-line utilities, see the *Command Line Reference Guide*.

How to Disable and Enable User Accounts

A user with a disabled account cannot access CA Harvest SCM; any login attempt using a disabled account always fails. Unlike locking a user account (which is automatically triggered by reaching the maximum failed login attempts), enabling or disabling a user account is a manual procedure.

CA Harvest SCM Administrators or users with Admin User Access can enable or disable user accounts by using the User Properties Account Disabled check box. This user property is not synchronized with the external authentication server. To log in to CA Harvest SCM, in addition to the conditions implemented by external authentication, the user account must not be disabled in CA Harvest SCM.

How to Unlock User Accounts

Unlock methods unlock an internally authenticated user account that has been locked because it has reached the maximum consecutive failed login attempts or the password has expired and the user is not permitted to change the password.

The following methods are available to unlock user accounts:

- The Locked check box on the User Properties dialog allows a CA Harvest SCM Administrator or a user with Administrative User access to unlock an internally authenticated user account that has been locked because of the following:
 - The user attempts have met the maximum consecutive failed login count.
 - The password has expired and the user is not permitted to change the password.

- A command-line utility, `husrunk`, lets an administrator unlock users whose accounts have been locked out.

Note: For details about the `husrunk` command-line utility, see the *Command Line Reference Guide*.

- If all users in the Administrator user group are locked out, no CA Harvest SCM user can unlock them. You can execute one of the following scripts to unlock a user:

Oracle:

`HUsrUnlk.sql`—This script accepts two arguments: CA Harvest SCM user name (case sensitive) and output log file name.

SQL Server:

`HUsrUnlk_sqlserver.sql`—This script cannot pass arguments. Edit the following identical two lines in the script before running it:

```
AND u.username = 'harvest')
```

Manually replace `harvest` with the username that you want to unlock.

If the user's failure count has exceeded the maximum failed login attempts before lockout value, executing the script resets the user's failure count to 0. This action lets the user attempt to log in again; otherwise, the user receives the override, Change Password at Next Login.

`HUsrUnlk.sql` and `HUsrUnlk_sqlserver.sql` are located in the directory Database under `CA_SCM_HOME`. To run the scripts successfully, you must have access to update the database using the relational database management system (RDBMS). To run the script, use the following syntax:

Oracle:

```
sqlplus Harvest owner/password @HUsrUnlk.sql username log_file
```

SQL Server:

```
osql -d HarvestDBname -i HUsrUnlk_sqlserver.sql -U owner -P password -e -b -o  
HUsrUnlk_sqlserver.log
```

Modify the Properties of a User

The User Properties dialog lets you modify the properties of a CA Harvest SCM user.

Follow these steps:

1. Click the User Groups tab of the Administrator application.
2. Expand the Users folder, right-click the user you want to modify, and select Properties from the shortcut menu.

The User Properties dialog appears.

3. Modify the user properties and click OK.

The user properties are modified.

User Groups

User groups can have any number of users, and you can assign users to any number of groups. User groups are the basis for access control and are nonhierarchical, which means the access available to one group does not imply access for any other. You can use user groups with notifications and approvals. For example, a user can belong to the Development group in Project A and the Testing group in Project B.

You can use user groups in the following areas:

Access control

Grants user groups permission to perform a particular action on an object. For example, specific users might be in a group named Developer, which has permission to check out and check in items, delete versions, and promote packages.

Notifications

Notifies specific users or all members of one or more user groups of certain events in the lifecycle. Notifications can be sent manually or automatically. For example, a notify process can alert the QA user group that a package has successfully been promoted from Development to QA.

Approvals

Specifies that certain users or user groups must give their approval before a package can progress from one state to another through the lifecycle.

CA Harvest SCM has the following predefined user groups with special meaning:

Public—All users implicitly belong to the Public group. If the Public group is added to the access list of a method, everyone can execute that method.

Administrator—Any user belonging to an Administrator group is exempt from security checks. As a security measure, only members of the Administrator group can grant Administrator rights to other users.

Important! Take extreme care to verify that at least one member of the Administrator group always exists.

Note: You can define other groups.

Create a User Group

The User Group Properties dialog lets you create and define CA Harvest SCM user groups.

Follow these steps:

1. Click the User Groups tab of the Administrator application.
2. Right-click the User Groups folder, and select New User Group from the shortcut menu.

The User Group Properties dialog appears.

3. Name the user group. User group names must be unique in a CA Harvest SCM installation.
4. (External Authentication only) Select External Authentication to specify external authentication for the user group; otherwise, the user group is created as internal authentication.

Note: To create an external user group, it must exist in the external authentication server.

5. Click the Users tab.

The Users list shows the names of users who belong to this user group. For a new group, the list is empty.

6. Click Add.

The Select Users for User Group dialog appears.

7. Select the user you want to add, and click OK.

The user name appears in the list.

8. (Optional) Select the user you want to remove, and click Remove.

The user name is removed from the list.

9. Click OK.

The user group is created and appears in the User Groups tab.

Add a User to a User Group

When adding users to a new user group, the Select Users for User Group dialog lets you locate and select users from a list. The Select Users for User Group dialog is invoked from the User Groups Properties dialog.

When the Select Users for User Group dialog initially displays, the From User Group(s) field lists all user groups and the Select User(s) list is empty. After you search for users, the Select User(s) list shows the results and you can return the users to the calling dialog.

You can search for users to populate the Select User(s) list in the following ways:

- To locate the users that belong to a specific user group, select the user group and click Find. This lists the users that belong to that group in the Select User(s) field.
- To locate all users in all user groups, select All User Groups in the From User Group(s) list and click Find. This action lists all users in all user groups in the Select User(s) field.

Note: If you have selected a user group but have not clicked Find, the Selected User(s) list still shows the previous selection.

Select one or more users and click OK to return the users to the calling dialog.

Follow these steps:

1. Click the User Groups tab of the Administrator application.
2. Expand the User Groups folder, right-click the user group to which you want to add a user, and select Properties from the shortcut menu.

The User Group Properties dialog appears.

3. Click the Users tab.

The users list shows the users that belong to the group.

4. Click Add.

The Select Users for User Group dialog appears.

5. Select the user you want to add and click OK.

Note: To return all users in the Select User(s) list to the calling dialog, click Select All and then click OK.

The users list includes the user that you added to the group.

6. Click OK.

The user is added to the user group.

Remove a User From a User Group

The User Group Properties dialog lets you remove a user from a user group.

Follow these steps:

1. Click the User Groups tab of the Administrator application.
2. Expand the User Groups folder, right-click the user group to which you want to remove a user, and select Properties from the shortcut menu.

The User Group Properties dialog appears.

3. Click the Users tab.

The tab lists the users that belong to the user group.

4. Select the user you want to remove and click Remove.

The users list does not include the user that you removed from the group.

5. Click OK.

The user is removed from the user group.

Delete a User Group

You can delete user groups. Delete User group is an auditable event.

Important! Use extreme caution when you delete user groups because access-related problems can occur.

Follow these steps:

1. Click the User Groups tab of the Administrator application.
2. Right-click the user group you want to delete, and select Delete from the shortcut menu.

A confirmation dialog appears.

3. Verify that it is the user group that you want to delete, and click Yes.

The user group is deleted.

Chapter 4: Administering Projects and Lifecycles

This section contains the following topics:

- [How to Define a Lifecycle](#) (see page 46)
- [Lifecycle Considerations](#) (see page 46)
- [Create a Project](#) (see page 47)
- [Copy Project](#) (see page 48)
- [How to Organize Projects](#) (see page 49)
- [Create a Working View](#) (see page 49)
- [Create a State](#) (see page 50)
- [Reorder States in a Project Lifecycle](#) (see page 50)
- [Define a Process](#) (see page 51)
- [Configure Baseline](#) (see page 52)
- [Create a Snapshot View](#) (see page 54)
- [Project Lifecycle Diagram](#) (see page 55)
- [Reports](#) (see page 56)

How to Define a Lifecycle

The order in which you perform the activities to define a lifecycle may vary, but the following basic steps are ordered to make it easy to follow and quick to set up a software development lifecycle:

1. [Plan the lifecycle](#). (see page 46)
2. [Create user groups and users](#) (see page 37).
3. [Create a project](#). (see page 47)
We recommend that you base your project on an existing one, or on a lifecycle template by copying it and then editing it.
4. [Create working views](#) (see page 49).
5. [Create the lifecycle states](#) (see page 50).
6. [Define the lifecycle processes](#) (see page 51).
The promote and demote processes depend on the existence of a target state; therefore, define all the states first, and then the processes.
7. [Create a repository](#) (see page 98).
8. [Load the repository](#) (see page 99).
9. [Configure the baseline](#) (see page 52).
10. [Set access permissions for the various project objects](#) (see page 105).
11. (Optional) [Set up the mail utilities for notifications](#) (see page 121).
12. (Optional) [Create custom form types](#) (see page 127).

Lifecycle Considerations

Analyzing your current software development and maintenance activities and mapping them into a lifecycle is a crucial first step to implementing CA Harvest SCM successfully. Consider the following questions and plan your lifecycle on paper before attempting to set it up in CA Harvest SCM.

- How many functional groups are involved in the lifecycle? Of these groups, which must have their activities isolated from one another and which can share a common working area?
- How many separate phases of development can you identify? At what point are approvals required and from whom? Who should be notified when a change is ready to move from one phase to another?

- What activities take place in each separate phase of development and who must do them?
- How is problem tracking be used with development? Do problems have their own lifecycle in a support project or are they created in a development project?

Create a Project

The Project Properties dialog lets you create a project.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Right-click Active Projects, Inactive Projects or Lifecycle Templates, and select New Project from the shortcut menu.

The Project Properties dialog appears.

3. Define the properties of the project.

Name

Names the project.

Active

Specifies whether the project is active and usable. For a new project, this option is enabled by default. The new project shows in the Active Projects folder.

Inactive

Specifies whether the project is inactive. The new project shows in the Inactive Projects folder.

Create User Group

Creates a user group with the same name as the project and assigns “use” access for the project. If such a user group exists, project creation succeeds by assigning “Use” access to the existing user group and the output log shows a warning message that says that the group exists. This option is disabled for update or delete project operations.

4. Click Apply.
Saves the project definition, but does not close the dialog.
5. Click the Access tab and set access for the project. Click OK.
The project is created and appears in the Lifecycles tab.

Copy Project

The Copy Project dialog lets you copy a project and select a destination folder for it.

The following properties are copied when you copy a project:

- Access (if it is selected on the Copy Project dialog)
- Note
- Package group notes
- Package groups
- Process access
- Process notes
- State notes
- State processes
- State views
- States
- Working views

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Right-click the project or template you want to copy, and select Copy To from the shortcut menu.

The Copy Project dialog appears.

3. Name the project, and select the destination folder from the Copy To drop-down list.
4. (Optional) If you want access control duplicated, select the Duplicate Access Control check box. Click OK.

The project is duplicated in the destination folder.

How to Organize Projects

The Lifecycles tab on the Administrator application includes the following folders that let you organize your projects:

- **Active Projects**—Contains all projects that are designated as active, but not as templates.
- **Inactive Projects**—Contains all projects that are not designated as active or as templates.
- **Lifecycle Templates**—Contains all projects that are designated as templates.

Project status determines the folder location of the project.

To organize projects, select the project that you want to move to a different folder and use its Properties to assign a different status.

The project is moved to the appropriate folder (Active, Inactive, or Lifecycle Templates) in the Lifecycles tab.

Change Project Status

The Lifecycles tab on the Administrator application contains three folders-Active Projects, Inactive Projects, and Lifecycle Templates. Project status determines the folder location of the project. The Project Properties dialog lets you change project status.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Right-click the project that you want to change, and select Properties from the shortcut menu.

The Project Properties dialog appears.

3. In the Project Attribute field, select the attribute (status) that you want to enable. Click OK.

The project status is changed and the project is moved to the appropriate folder (Active, Inactive, or Lifecycle Templates) in the Lifecycles tab.

Create a Working View

The Working View Properties dialog lets you name a working view.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.

2. Navigate to the Data Views folder for your project, right-click the Working Views folder, and select New Working View from the shortcut menu.

The Working View Properties dialog appears.

3. Name the view, and Click OK.

The working view is created and appears in the Lifecycles tab.

Create a State

The State Properties dialog lets you create and define a state. The dialog also lets you view modify state attributes.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Right-click the States folder, and select New State from the shortcut menu.

The State Properties dialog appears.

3. Define the properties of the state.

Name

Names the state. Each state in a project must have a unique name.

View

Shows all the currently defined working views in the project. To associate the state with a particular view, select the view from this list.

All Snapshot

Accesses all snapshots in a project in a read-only mode. This option automatically disables the View drop-down list and all processes that are used to update items this state.

4. Click Apply.
Saves the state definition, but does not close the dialog.
5. Click the Access tab and set access for the state. Click OK.

The state is created and appears in the Lifecycles tab.

Reorder States in a Project Lifecycle

The Project Properties dialog lets you reorder states in a project lifecycle.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.

2. Right-click the project that has the states you want to reorder, and select Project Properties from the shortcut menu.

The Project Properties dialog appears.

3. Click the State Order tab.

The project states are listed the Change State Display Order field.

4. Select the state you want to reorder and use the arrow icons to move the state up or down in the project lifecycle.

A numbered sequence column identifies the order of the states and the states' associated views are listed.

5. Click OK.

The updated state order shows in the workspace and the list view.

Define a Process

The processes defined for a state determine what activities users can perform in that state. Each process type has its own associated Properties dialog. The Properties dialogs let you define processes and establish default values for them.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Navigate to the state in which you want to create the process, right-click the Processes folder, and select New, *process name* from the shortcut menu.

The Properties dialog appears.

3. Define the properties of the process and click Apply.
Saves the process definition, but does not close the dialog.
4. Click the Access tab and set access for the process. Click OK.

The process is created and appears in the Processes folder.

More information:

[How to Define a Lifecycle](#) (see page 46)

Add a Process Note

You can enter comments or instructions to the process execution dialog that are helpful to users executing processes.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Navigate to the process for which you want to add a note, right-click the process, and select Properties from the shortcut menu.

The Properties dialog appears.

3. Click the Note tab and enter any comments or instructions that would be helpful to users executing the process. Click OK.

When a user executes the process, the user can view the note, but not change it without the proper access.

Configure Baseline

When you create a project, it automatically contains an empty baseline. The name *baseline* is a reserved name; no other view can have this name. An administrator must specify the repositories to include in the baseline. The repositories contain the physical data items that CA Harvest SCM controls. The Configure Baseline dialog lets you configure a baseline.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Navigate to the Data Views folder for your project, right-click Baseline, and select Configure Baseline from the shortcut menu.

The Configure Baseline dialog appears.

3. Complete the dialog fields:

View Type

Lists the available repositories of that type after you select a view type from the drop-down list.

- **ALL**—Selects all repositories or a limited set as the starting point for this new development.
- **Base**—Accesses a new repository that has been loaded. CA Harvest SCM automatically creates this view to reflect the repository before changes are made to it.

- **Snapshot**—Specifies a snapshot view. When a baseline is defined, you can select from among the global snapshot views available as the starting point for further development. These global snapshots typically represent significant points in an application's history, such as a release. After you select a snapshot view, the repositories that the snapshot includes are listed. Although these repositories have the same names as those included in the Base view, the items in that repository are the versions included in that snapshot view when it was created, and not in the original version of the repository.

Available Repository/View List

Lists repositories that are associated with your server and are available to add to the baseline.

The first entry in the list is Base. Selecting Base returns a list of all available repositories in their original condition to the Available Repository/View List, where you can select them for inclusion in the baseline.

If any repositories have been added to the baseline of the current project, their names are displayed in this list. Selecting repositories from the list and clicking Add shows your selection in the Selected Repository List.

Important! You can add more repositories to the baseline at any time; however, after development has begun in a project and items have been changed, you cannot remove a repository from the baseline.

Selected Repository List

Displays your repository selections in a list. Remove is enabled when a repository or a directory in this list is selected. Select a repository or a directory, and click Remove to remove the selection from the baseline. If a repository has been changed in this project, it cannot be removed from the baseline.

Read/Write or Read Only

Works with the Selected Repository List and lets you change the access type of a repository. When you select a repository in the Selected Repository List, the access type of your selection enables this option. To change the access type of a repository, select the repository in the Selected Repository List, and click this button.

Click OK.

The baseline is configured and appears in the Lifecycles tab.

Delete a Baseline

You can delete a baseline if none of its items have been changed.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Navigate to the Data Views folder for your project, right-click the Baseline you want to delete, and select Delete from the shortcut menu.

A confirmation dialog appears.

3. Click Yes.

The baseline is deleted and no longer appears in the Lifecycles tab.

Create a Snapshot View

The Snapshot View Properties dialog lets you define a snapshot view. Consider the following rules when create a snapshot view:

- You cannot take a snapshot if the parent view has any reserved or merge-tagged versions. Reserved versions must be checked in and merged versions must be resolved before taking the snapshot.
- Branch versions are never included in a snapshot because they do not affect views.
- You can only modify the Visible to Other Projects option in the snapshot view properties. The From View and As of Date values are set at creation time.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Navigate to the Data Views folder for your project, right-click the Snapshot Views folder, and select New Snapshot View from the shortcut menu.

The Snapshot View Properties dialog appears.

3. Name the view and define it:

From View

Specifies the view on which it is based—because a snapshot is a read-only image of a working view. The snapshot is based on versions of items in this working view. You select a parent view from the list of working views available on this drop-down list.

Visible to Other Projects

Makes the snapshot view visible to other projects. You can include these snapshots in the baseline of other projects. When you copy a snapshot to the repository, it is automatically visible to other projects. (Working views are part of a project and are only accessible from within it.)

Latest Versions

Includes the latest versions of the current working view in the snapshot view.

As of Modified Version Date

Specifies a date and time. You can use the drop-down lists or accept the current date and time defaults. The take snapshot process captures versions in the current working view that were modified before or on the specified date and time.

Important! This behavior deviates from AllFusion Harvest Change Manager 4.x. The date and time are compared with the version modification time. This option does not refer to the date and time that versions were present in the working view. For a description of how to simulate AllFusion Harvest Change Manager 4.x behavior, see tech note TEC293284 at <http://ca.com/support>. AllFusion Harvest Change Manager 4 is no longer supported; however, we provide this information as a courtesy to our AllFusion Harvest Change Manager 4 clients. For information about CA policy for unsupported products, see <http://ca.com/support> or contact your Account Representative.

Click OK.

The snapshot view is created and appears in the Lifecycles tab.

Project Lifecycle Diagram

The Project Lifecycle diagram gives you a graphical view of a project lifecycle and lets administrators modify lifecycles. The diagram lets you easily determine the relationship of states and the impact of connections between them including promote, demote, and approval processes. The graphical layout diagram reflects any changes that administrators make to a project lifecycle.

The Project Lifecycle diagram works in the following ways:

- You can reorganize the layout by dragging a state box to a new position. When you drag a state box, all of its attached connections also move.
- You can view or modify state or process properties by double-clicking the respective box or line; modifications are reflected in the Administrator application.
- A shortcut menu provides you with options to Zoom, and Save as JPG.

You can access the Project Lifecycle Diagram from the Workbench or the Administrator application. On the CA Harvest SCM Web Interface, you can access the Project Lifecycle diagram from the Project Properties Lifecycle tab page. To display or update your lifecycle diagrams in the Web Interface, an administrator must perform setup steps.

Note: For information about setting up lifecycle diagrams in the Web Interface, see the *Implementation Guide*.

Project states are depicted as boxes with a colored bar across the top that indicates the View type. The name of the state and view are also shown on the box. If the state has an approve process, an approval icon shows in the lower right corner of the box.

CA Harvest SCM promote and demote processes are represented as connecting, directed lines. A double arrowhead indicates multiple promote processes, multiple demote processes, or whether both a promote process and a demote process exist between two states with the transition in the same direction (same from-state and same to-state). The double arrowhead exists in the latter case even if only a promote process or only a demote process shows. Both pre-linked and post-linked notify and user-defined processes are shown as icons on the transitions.

Reports

Using the Reports menu, you can generate SCM-level and project-level reports. Initially all project-level report options are disabled, you must click a project to activate the reports options. In addition, if you do not have access privileges the option is disabled. You can also access the project-level report options from project shortcut menu. All reports display in the output log unformatted.

The Administrator application reports incorporate project management information from CA Harvest SCM. These reports let you quickly determine a project's progress. The reports provide information about project access and process activities. You can locate problem areas quickly to identify and isolate small problems before they become critical.

The following table lists the Administrator application reports and provides a brief description of each report.

Level	Report Name	Description
SCM	Project Summary	Lists detailed information about each project.
	Repository Summary	Lists detailed information about each repository.
	CA Harvest SCM Access	Lists what access the specified user has to the objects.
	Users	Lists detailed information about each user.

Level	Report Name	Description
Project	User Groups	Lists detailed information about each user group.
	Lifecycle Definition	Lists all process and process types for each state in the selected project.
	Project Access	Lists all project and state access belonging to a selected project.
	Approval Definition	Lists all approve processes with approval user/user group in a selected project.
	User List	Lists all users in each user group.

Chapter 5: Defining Lifecycle Processes

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

This section contains the following topics:

- [Processes](#) (see page 59)
- [Define an Approve Process](#) (see page 64)
- [Define a Promote Process](#) (see page 65)
- [Define a Demote Process](#) (see page 66)
- [Define a Check-In Process](#) (see page 67)
- [Define a Check-Out Process](#) (see page 70)
- [Define a Compare View Process](#) (see page 73)
- [Define a Create Package Process](#) (see page 74)
- [Define a Cross Project Merge Process](#) (see page 77)
- [Define a Delete Package Process](#) (see page 79)
- [Define a Delete Version Process](#) (see page 79)
- [Define a List Version Process](#) (see page 80)
- [Define a Move Item Process](#) (see page 81)
- [Define a Move Package Process](#) (see page 82)
- [Define a Move Path Process](#) (see page 83)
- [Define a Notify Process](#) (see page 84)
- [Define a Remove Item Process](#) (see page 86)
- [Define a Remove Path Process](#) (see page 87)
- [Define a Rename Item Process](#) (see page 88)
- [Define a Rename Path Process](#) (see page 89)
- [Define a Switch Package Process](#) (see page 90)
- [Define a Take Snapshot Process](#) (see page 91)
- [Define a User-Defined Process](#) (see page 92)

Processes

A *process* is a command that executes an action. CA Harvest SCM includes a predefined set of process types. You can define a set of processes for each state in the lifecycle. The processes that are defined for a state determine what activities users can perform in that state. Administrators have complete control over which processes are available in each state of the lifecycle, and which users can execute these processes. For example, you can set up the lifecycle to allow check-in and check-out in the Development state, but only allow check-out in the Test state. You can add more than one process of one type to a state. For example, you can have one check-out process for updating items and another check-out process for read-only.

Each process has a type and a name. CA Harvest SCM predefines the process types. A process name is a more precise label that you can specify for added clarity and to distinguish between multiple processes of the same type. For example, promote is one of the predefined process types in CA Harvest SCM. A promote process that has been set up for promoting a package to a QA state could be named Promote to QA. If a name for the new process is not specified, it defaults to Process-*nn* (*nn* is the next process record number). The Processes menu in the States folder lists processes in alphabetical order.

You can define user-defined processes (UDPs) to invoke user-supplied programs from CA Harvest SCM. This feature lets the administrator integrate the use of existing development tools into the lifecycle.

The properties dialog for a process displays associated Access, Pre-Links, and Post-links.

- Access shows the type of access that is available to a process: Execute Access.
- Pre-Links and Post-Links list the pre-linked and post-linked processes for the selected process.

More information:

[Sample Software Development Process](#) (see page 16)

Process Types

The following table provides a summary of each process type. All processes can have UDP and notify processes linked to them with the Pre-Link and Post-Link options.

Approve

Approves or rejects a specified package for promotion to the next state.

Check-in

Creates a version of an item by bringing the changes made to a file into the current view path, and optionally releases the reserved status of the parent version in the repository.

Check-out

Copies a version of an item to the file system, optionally reserving the version in the current view or on a branch.

Compare Views

Displays item differences between two views.

Concurrent Merge

Merges all the changes a package has made on a branch back to the trunk.

Create Package

Creates a package and form with the same name and automatically associates them.

Cross Project Merge

Merges all the changes a package has made in one project into a package in another project.

Delete Package

Differentiates users who can edit a package from those who are allowed to delete a package.

Demote

Returns a package to a defined previous state in the lifecycle.

Delete Version

Removes the last change made to an item.

Interactive Merge

Shows branch versions, and conflicts between versions so you can resolve them.

List Version

Lists information associated with deltas.

Move Item

Logically moves an item from the current path to another path.

Move Package

Moves a package from a state in one project to a state in another project.

Move Path

Logically moves a path from the current location to another path.

Notify

Sends mail to specified users or user groups.

Promote

Moves a package to the next defined state in the lifecycle.

Remove Item

Logically deletes an item as part of a package's changes.

Remove Path

Logically removes a path.

Rename Item

Logically renames an item.

Rename Path

Logically renames a path.

Switch Package

Moves versions from a source package to a target package.

Take Snapshot

Creates a snapshot view from the working view that includes the current state.

UDP

Executes a user-defined process on the client or server computer.

Process Aspects

A process has the following aspects:

Definition—Process properties dialogs let administrators define processes and establish default values for them during the setup and maintenance of a lifecycle. Each process type has an associated properties dialog.

Execution—Users execute processes by selecting the process from the menu at the state or package context. When the process execution dialog appears, some fields can be populated with default values set by the administrator. Users can accept these defaults or change them before executing the process; however, users cannot change all the properties of the process this way. Some properties are part of the process definition and only the administrator can change them using the process properties dialog.

Linked Processes

Each process in a lifecycle can have one or more processes linked to it. The following methods of process linking are available:

- *Pre-linked* processes execute *before* the start of the parent process. For example, processes pre-linked to a promote process execute before packages are promoted to the next state in the lifecycle.

Note: The pre-linked method is unavailable for the command-line utility except for the promote and demote processes.

- *Post-linked* processes execute *after* the start of the parent process. For example, post-linked processes execute only after the packages are successfully promoted.

The order of display in the Pre-Linked or Post-Linked tabs determines the order of the linked process execution. Your CA Harvest SCM session is suspended during execution of linked processes. If multiple processes are linked to one parent process, the execution of each depends on the success of the previous one. If a linked process or the parent process fails, none of the remaining processes execute including the parent process.

Note: You must set up linked processes so that they do not require any input from the user upon execution. If the input is insufficient for execution, the process fails.

Process Access

Processes are considered objects in CA Harvest SCM, so the same rules that govern other object access apply to processes with one exception: Processes that are linked to another process have no access control of their own. They are secured through the access control associated with the process to which they are linked.

More information:

[Set Object-Level Access](#) (see page 117)

Process Execution

The client computer executes all CA Harvest SCM processes, with the exception of UDPs. If a process must run on a computer other than the client, you can define a UDP to run it.

CA Harvest SCM can only access directories on computers where a CA Harvest SCM agent is running. For example, during check-in processes verify that the files you want to check in are located on an appropriate client. When the files you want are on remote computers, you can use operating system tools, such as the Network File System from Sun Microsystems or the Distributed File System from the Open Software Foundation, to distribute the files.

External processes invoked through UDPs or notifications are executed in synchronous mode. After a user-defined or notification process starts, you cannot perform keyboard actions until the process completes.

Define an Approve Process

The Approve Properties dialog lets you define the users and user groups who must give their approval before they can promote or move a package out of the associated state.

Note: To prevent users from changing a package while it is pending approval, we recommend that you create a state named Approve to which users can promote packages. Do not define any processes in the state that would allow users to change packages (for example, check-out for update, check-in, delete versions, and so on).

For a post-linked process to run, all packages must be approved. If the approval of any selected package fails, post-linked processes do not execute. If any of the pre-linked processes fail the approvals do not execute.

A user can reject a package when it does not meet standards. If a member of a group rejects a package, that particular user must approve the package to remove the rejection; the approval of another user in the group does not override it. If it is necessary to override a rejection, you can add a second approval process.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Approve Process from the shortcut menu.

The Approve Properties dialog appears.

3. Name the process and use the following fields to add approvers to the process:

Approval Users

Specifies all users who are responsible for approving packages. You can add one or more users to the approval list by clicking Add to open the Add Users dialog. You can delete users by selecting one or more names from the Users list, and clicking Delete.

Approval User Groups

Specifies all user groups who are responsible for approving packages. You can add one or more user groups to the approval list by clicking Add to open the Add User Groups dialog. You can delete user groups by selecting one or more names from the User Groups list, and clicking Delete.

Approvers are defined for the process.

4. Click Apply.

Saves the process definition, but does not close the dialog.

5. Click the Access tab and set object-level access for the process. Click OK.

The approve process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Promote Process

The Promote Properties dialog lets you define a promote process. The setting of the options on this dialog determine the choices the user can make when executing the promote process:

- If the option is not selected on the Promote Properties dialog, the option is not selected on the *promote* process dialog. The user executing the promote process can decide to select the option.
- If the option is selected on the Promote Properties dialog, the option is selected by default on the *promote* process dialog. When the promote process is executed the option is enforced and the user cannot override it.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.
2. Right-click the Processes folder, and select New, Promote Process from the shortcut menu.

The Promote Properties dialog appears.
3. Name the process and specify a destination state.
4. Specify the options that you want to be available to users on the process execution dialog:

Enforce Package Bind

Forces all the packages belonging to a bound package group to be promoted together; otherwise, an error occurs.

Enforce Package Merge

Prohibits packages from being promoted to the next state if they are associated with branch versions. If you enforce that packages must be merged, the latest change for the package must be on the trunk, not on a branch. If the promote process is in a state with no view, or if that state's view is the same as the one in the Promote To state, this option is not enforced.

Verify Package Dependency

Disallows promotion of packages that depend upon other packages. Dependency is based on versions in the view. In the current state, a package with a higher item-version cannot be promoted without also promoting the packages with the lower item-versions in the current view, unless the lower item-versions exist in the view of the Promote To state. If the promote process is in a state that shares the same view as the one in the Promote To state, this option is not enforced.

Note: The lower item-version causes a dependency error only when it is on the trunk.

The process is defined.

5. Click Apply.

Saves the process definition, but does not close the dialog.

6. Click the Access tab and set access for the process. Click OK.

The promote process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Demote Process

The Demote Properties dialog lets you define a demote process. The setting of the options on this dialog determine the choices the user can make when executing the demote process:

- If the option is not selected on the Demote Properties dialog, the option is not selected on the Demote Process Execution dialog. The user executing the demote process can decide to select the option.
- If the option is selected on the Demote Properties dialog, the option is selected by default on the Demote Process Execution dialog. When the demote process is executed the option is enforced and the user cannot override it.

Pre-Linked processes execute before packages change states, and Post-Linked processes execute after packages change states. The failure of a Pre-Linked process generates an error and causes the demote process to fail. The state of the packages changes only after the successful completion of all Pre-Linked processes. Typically, a notify process is pre-linked to the demote process.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Demote Process from the shortcut menu.

The Demote Properties dialog appears.

3. Name the process and specify a destination state.
4. Specify the options that you want to be available to users on the process execution dialog:

Enforce Package Bind

Enforces all the packages belonging to a bound package group be demoted together; otherwise, an error occurs.

Verify Package Dependency

Prevents demotion of packages upon which other packages depend. Dependency is based on versions in the current view. In the current state, a package with a lower item-version cannot be demoted without also demoting the packages with the higher item-versions in the current view. If the demote process is in a state that shares the same view as the one in the Demote To state, this option is not enforced.

Note: The higher item-version causes a dependency error when it is on the trunk or on the branch.

The process is defined.

5. Click Apply.
Saves the process definition, but does not close the dialog.
6. Click the Access tab and set access for the process. Click OK.

The demote process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Check-In Process

The check-in process creates a version of an item by bringing the changes made to a file into the current view path, and optionally releases the reserved status of the parent version in the repository. The Checkin Properties dialog lets you define the check-in process.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Checkin Process from the shortcut menu.

The Checkin Properties dialog appears.

3. Name the process and specify options.

The options will be available to users on the process execution dialog.

4. Click the Defaults tab and specify default values for options.

The process dialog will display the default values and the user can override them.

5. Click Apply.

The process definition is saved.

6. Click the Access tab and set access for the process. Click OK.

The check-in process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

More information:

[Check-In Properties Options](#) (see page 68)

Check-In Properties Options

The Checkin Properties dialog lets you select options to be available to users on the process execution dialog and specify defaults for various execution options. Following are the available options and their descriptions:

New or Existing Items

Checks in files when the package reserves them or when they do not exist in the repository. This filter does not require the corresponding item to be in the current view. However, if it is in the current view, it must have a reserved version in the current package.

New Items Only

Limits the check-in to files that do not have corresponding items in the current view.

- If an item has been removed from the current view, the file can be checked in using this filter.
- If the corresponding item was renamed in the current project, the file cannot be checked in. The file must be renamed before it can be checked in.

Existing Items Only

Limits the check-in to files that have corresponding items reserved by the package. Any files without corresponding items are skipped.

Check In by Owner Only

Prevents a user from releasing a version if that user did not initially reserve it. By default, any user can release a reserve-tagged version by checking in a copy of the reserved item.

Enforce Change Description

Forces the user to enter a description to check in successfully.

Note: If you select the Enforce change description option and if the Display silent check in command in menus option is selected in Visual Studio .NET, users will get an error message and the check-in process may not be successful.

New Item on Trunk

Limits the check-in to files that do not have corresponding items in the current view and creates trunk versions.

New Item on Branch

Limits the check-in to files that do not have corresponding items in the current view and creates branch versions. If an item exists on the branch, a user can check in the item to a branch in a different view. This option supports concurrent development of the same file using different packages. For example, if user1 uses package1 to check in file1.txt, a x.1.1 version is created. User2 can check in the same file (file1.txt) using package2 to the same view on a branch as x.2.1 version.

Note: The New Items on Trunk and New Items on Branch options are disabled when the user selects Existing Items Only; otherwise, one or both can be selected. If the user only selects New Items on Trunk, new items can only be checked in to the trunk. If the user only selects New Items On Branch, new items can only be checked in to the branch. If the user selects both, new items can be checked in to either the trunk or the branch.

The Defaults tab lets you specify default options. When a user opens the process execution dialog, the default values are displayed and the user can override them.

Update and Release

Updates or creates the item in the view and it is no longer reserved by the package. Sets the file permission to read-only so that changes cannot be made to the file until it is checked out again.

Update and Keep

Updates or creates the item in the view . The current package keeps the item reserved and checked out so that the user can modify the item.

Release Only

Releases the reserved tag, no check-in is performed, and the item is not updated. The file permission is set to read-only, indicating that changes should not be made to the file until it is checked out.

Item Filter

Specifies the default item filters; a disabled filter value cannot be selected as a default.

Note: Select the Existing Items Only for the default item filter to prevent your repository from being cluttered up with unwanted files. It is common during development to create temporary files or templates in a working directory. If all files are selected during check-in, these unwanted files can be included, although they are not a meaningful part of the application being controlled.

Preserve directory structure

Checks in selected files to paths with names that correspond to their client directory location, if these paths currently exist.

Preserve and create path structure

Checks in selected files to paths with names that correspond to their directory location, and creates any view paths that do not currently exist.

All files to same view path

Checks in all selected files to the same path in the destination view, ignoring the client directory structure.

Delete Files After Check In

Controls whether files checked in should be deleted from the client file system at the completion of the check-in process.

Description

Provides up to 2,000 characters of descriptive text for the check-in process. This text shows on the process execution dialog and lets you inform users what the process does. You are not required to enter anything in this field. If the process is linked to another, the description is invisible during execution.

Define a Check-Out Process

The check-out process copies a version of an item to the file system, optionally reserving the version in the current view or on a branch. The Checkout Properties dialog lets you select the check-out mode or modes to allow, and specify default values for various execution options.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.
The Processes folder is listed in the workspace.
2. Right-click the Processes folder, and select New, Checkout Process from the shortcut menu.
The Checkout Properties dialog appears.
3. Name the process and specify options.
The options will be available to users on the process execution dialog.
4. Click the Defaults tab and specify default values for options.
The process dialog will display the default values and the user can override them.
5. Click Apply.
The process definition is saved.
6. Click the Access tab and set access for the process. Click OK.
The check-out process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Check-Out Properties Options

The Checkout Properties dialog lets you select options to be available to users on the process execution dialog and specify defaults for various execution options. Following are the available options and their descriptions:

Browse

Checks out the items to the destination directory but does not reserve the item and does not allow the user to check the files back in. The read-only attribute is set on files checked out in Browse mode.

Update

Copies selected versions to the destination client directory and creates a reserved version on each item's trunk, allowing the corresponding files to be checked in. The permission on a read-only file is changed to normal (write access) when this mode of check-out is used.

Reserve only

Creates a reserved version on each item's trunk, but does not move any data to external directories.

Synchronize

Identifies the versions of the files in the client file system by using the signature file. CA Harvest SCM versions are checked out only if the signature file shows:

- Client files that differ from their corresponding CA Harvest SCM versions.
- Client files whose timestamps differ from their corresponding timestamps in the signature file.

Items are checked out in a read-only mode. The check-out for Synchronize mode is especially useful in the build process.

Concurrent update

Copies selected versions to the destination client directory and creates a reserved version on a package branch for each item. All package updates accumulate on this branch. The permission on a read-only file is changed to normal (write access) when this mode of check-out is used.

Notification is sent to all users that have checked out the item for concurrent update, including items updated in different CA Harvest SCM projects.

Note: Creating a check-out process that allows only Browse mode can be useful in a QA or testing state where no updates are made. To design a lifecycle with enforced package branching, allow only check-out for Concurrent update in certain states.

Shared Working Directory

Replaces read-only files on the UNIX file system regardless of file ownership; this option is useful when two or more developers share the same working directory. By default, CA Harvest SCM replaces read-only files on a UNIX client system only if the user executing the check-out process is the owner of the files.

This option also affects the ownership of client directories created by the check-out process when the Preserve and create directory structure option is used. Without Shared working directory selected, directories created during check-out have write access only to the user who executes the check-out process. When Shared Working Directory is selected, all new directories created during check-out have group write access.

Note: All directories being checked out to must have group write access, and all users sharing the working directories must be in the same UNIX group.

Preserve file timestamp

Restamps the file with the time at check-out; otherwise, the current system time is used.

The Defaults tab lets you specify default options. When a user opens the process execution dialog, the default values are displayed and the user can override them.

Preserve View Path Structure

Checks out all selected items into corresponding client directories, if they exist.

Preserve and Create Dir. Structure

Checks out all selected items into corresponding client directories and creates any client directories that do not currently exist.

All Items to Same Client Dir.

Places all selected items directly beneath the specified client directory, ignoring the path structure in the repository.

Replace Read Only Files

Controls whether the checked-out files should replace existing read-only files on the client file system. This option lets you replace files that you previously checked out as read-only or checked in.

Note: On UNIX platforms, the Replace Read-Only Files option replaces files only if the user who is executing the check-out process owns them.

Files that are not read-only on a Windows file system or that have write permission on the UNIX file system are *never* overwritten. CA Harvest SCM assumes that such a file has been checked out for Update and not checked back in yet. Overwriting such a file might cause a user to lose unsaved changes.

If no corresponding file exists on the host, the item is checked out regardless of the value for this option.

Define a Compare View Process

The compare view process generates an overview report showing a summary of the differences between any two views, either snapshot or working, that exist in any project. This report generates three lists:

- Items that exist only in the first view
- Items that exist only in the second view
- Items that exist in both views but have conflicting changes

You can also use the process to see the difference between two items or sets of items, rather than an entire view.

- The detailed Compare View Report shows the differences found for all listed items.
- The Visual Difference dialog shows the differences between the two versions of a single changed item.

Pre-Linked and Post-Linked processes execute before and after you click Compare View on the process execution dialog.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Compare View Process from the shortcut menu.

The Compare View Properties dialog appears.

3. Name the process and click Apply.

Saves the process definition, but does not close the dialog.

4. Click the Access tab and set access for the process. Click OK.

The compare view process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Create Package Process

The create package process lets users create packages and optionally, corresponding forms. The Create Package Properties dialog lets you define a create package process and specify whether a form should be created and automatically associated with the package created by this process, and if so, what kind of form. You can also specify an initial state for the package to be created in and a default name for packages created with this process.

You can link notify and UDP processes to a create package process using the Pre-Linked or the Post-Linked process. Linked processes execute after OK or Apply is clicked on a package/form combination.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.
The Processes folder is listed in the workspace.
2. Right-click the Processes folder, and select New, Create Package Process from the shortcut menu.
The Create Package Properties dialog appears.
3. Name the process and specify the options that you want to be available to users on the process execution dialog.

Initial State

Lists all the states in the lifecycle of the current project. Select the state that packages created by this process should begin the lifecycle in. When the user executes the process, the package is created in this state regardless of the state from which the process was invoked.

Type

Lists the currently installed form types. If you check the Create Automatically option, this field becomes active and you can select the type of form to be created.

Create automatically

Creates a form with the same name as the package created by this process and associates them.

4. Click the Defaults tab and specify default options:

Default Name

Specifies a default name for the package and form to be created by this process. The user executing the process can modify the name.

Prevent Name Change During Package Creation

Prohibits the user from changing the package name. This field should be disabled for text-only template names; otherwise, users will not be able to create more than one package with the process.

Note: A blank Default Name is allowed for DBMS triggers even if this option is selected.

Options are defined for the process.

5. Click Apply.
Saves the process definition, but does not close the dialog.
6. Click the Access tab and set access for the process. Click OK.

The create package process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

More information:

[How to Specify Default Package and Form Names](#) (see page 76)

How to Specify Default Package and Form Names

You can set the Create Package Properties dialog to specify a default name for the package and form that the create package process creates. For example, you might want all package names to begin with a certain prefix such as Bug #.

The default name can include formats to generate dates or numbers automatically by including database management system (DBMS) format models in the Default Name field, as shown in the following table.

Default Name	Comment	Results
CR-%N('99')	Insert an incremental number	CR- 1, CR- 2, CR- 3,...
MR-%N('009')	Incremental number, zero padding	MR- 001, MR- 002, MR- 003,...
PKG%N('009MI')	Incremental number, zero padding, trailing minus	PKG001, PKG002, PKG003,...

Note: By default, the DBMS format returns positive numbers with a leading space that is reserved to hold a plus sign (+). You can suppress this behavior by including the 'MI' (trailing minus) as the last characters in the format string as shown in preceding examples.

More information:

[Define a Create Package Process](#) (see page 74)

Define a Cross Project Merge Process

The cross project merge process merges changes made to items in one project with the changes made to the same items in another project. The cross project merge creates versions in the target project that are the latest for each item on the trunk.

The cross project merge process is especially useful in parallel development, when the same items are changed in two different projects.

You can link notify and UDP processes to a cross project merge process using the Pre-Linked or the Post-Linked process. Linked processes execute before and after the merge is complete.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Cross Project Merge Process from the shortcut menu.

The Cross Project Merge Properties dialog appears.

3. Name the process and specify the options that you want to be available to users on the process execution dialog.

Merge Conservatively

Creates a merge-tagged version, regardless of the contents of the versions. The process fails if the target package has an unmerged branched version of an item that is also in the source package.

Merge Aggressively

Creates a merge-tagged version only when conflicts are found. If no conflicts are found, the branch and trunk versions are merged to create a normal version. Normal tags can only be created when the versions being compared are in the original baseline of both projects. If the versions are checked in after baselining, merge tags are created regardless of whether conflicts exist.

Note: A conflict occurs when a line or block of data is modified in both the source and the destination; insertions and deletions are not conflicts.

Take Trunk (Target) Version

Automatically selects the destination (trunk) to create the final version, without comparing the contents of the versions. This option creates a normal version on the destination trunk and closes the source branch version.

Take Branch (Source) Version

Automatically selects the source branch version to create the final version, without comparing the contents of the versions. This option creates a normal version on the destination trunk and closes the source branch version.

4. Select one or more rules to specify the target location of the merge version automatically:

Branch Only

Creates a version on the target branch. This option allows changes to be copied from the source project to the target project even if one or more target items are reserved for update in the main trunk. A branch is created to store the changes. The target package cannot be the same package that contains the items reserved for update on the main trunk.

Trunk Only

Creates a version on the target trunk.

Trunk or Branch

Creates a version on the target trunk or branch. This option allows changes to be copied from the source project to the target project even when one or more target items are reserved for update in the main trunk. If items are reserved for update on the trunk, a branch is created to store the changes only if the target package is not the same package that contains the items reserved for update. If items are not reserved for update on the trunk, the items are simply copied to the trunk.

Options are defined for the process.

5. Click Apply.
Saves the process definition, but does not close the dialog.
6. Click the Access tab and set access for the process. Click OK.

The cross project merge process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

More information:

[Parallel Development](#) (see page 275)

Define a Delete Package Process

The delete package process lets CA Harvest SCM users delete packages and lets administrators differentiate users who can edit a package from those users who are allowed to delete a package. Users belonging to user groups who are allowed to execute a delete package process associated with a package state can delete packages and the forms associated with the packages.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Delete Package Process from the shortcut menu.

The Delete Package Properties dialog appears.

3. Name the process and click Apply.

Saves the process definition, but does not close the dialog.

4. Click the Access tab and set access for the process. Click OK.

The delete package process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Delete Version Process

The delete version process allows users to remove selected versions of items from the CA Harvest SCM repository.

You can link notify and UDP processes to a delete version process by using the Pre-Linked or the Post-Linked process. These processes execute before and after the delete version process completes successfully. In this case, successfully means that at least one version is deleted.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

2. The Processes folder is listed in the workspace.

3. Right-click the Processes folder, and select New, Delete Version Process.

The Delete Version Properties dialog appears.

4. Name the process.

5. (Optional) Select the Delete by Creator/Modifier only option.

Note: If you select this option, *only* the creator or modifier or administrator would be able to delete the version.

6. Click Apply.

Saves the process definition, but does not close the dialog.

7. Click the Access tab and set access for the process. Click OK.

The delete version process is now added to the state. The process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a List Version Process

The List Version Properties dialog lets you specify defaults for various execution options. When a user invokes the process execution dialog, the values specified with the properties are displayed as initial defaults and the user can override them.

The list version process lets users generate reports about the changes made to items in the current project. The listing produced by this report is in the same format as that produced by the UNIX diff command.

Linked processes execute before and after the list version process completes successfully. In this case, “successfully” means that at least one version is listed.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, List Version Process from the shortcut menu.

The List Version Properties dialog appears.

3. Name the process and click Apply.

Saves the process definition, but does not close the dialog.

4. Click the Access tab and set access for the process. Click OK.

The list version process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Move Item Process

The move item process lets users logically move an item from the current path to another path. The movement is included as a change associated with a package and can progress through the lifecycle as the package is promoted from one view to another. When an item has been moved, it no longer displays under the original path in the item view. The move item process creates a new version located on the new parent path. This version has properties like other versions and can be seen in the List view by double-clicking the item or through the Find Version dialog.

Linked processes execute before and after the move item process completes successfully. In this case, "successfully" means that at least one item is moved.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Move Item Process from the shortcut menu.

The Move Item Properties dialog appears.

3. Name the process and specify options:

On Trunk

Creates a trunk version for refactoring changes. (This option is identical to the check-out for update process.)

On Branch

Creates a branch version for refactoring changes. (This option is identical to the check-out for concurrent update process.)

Update and Release

Creates a normal version for the refactoring change. If a reserved version already exists in the same package, it is updated as a normal version.

Update and Keep

Creates a normal version for refactoring changes. Another new reserved version is created after the normal version.

The options are selected for the process.

4. Click Apply.

Saves the process definition, but does not close the dialog.

5. Click the Access tab and set access for the process. Click OK.

The move item process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Move Package Process

The move package process moves a package from a state in one project to a state in another project. The move package process is similar to the promote process. Promote moves a package forward from one state to another in a project. As with the promote process, if the state has approval processes, they are verified before the package is moved.

Unlike promote, the move package process moves only the package definition and history, not any versions associated with the package in the current project. If versions are associated with a package, it cannot be moved.

To execute this process, users must have access to the create package process in the destination project and state.

Important! If you delete a state that was the target of a move package process, the process must be deleted or redefined with a new target state.

Linked processes execute before and after all the selected packages are moved to the target project. If any package fails to be moved, the post-linked processes do not execute.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Move Package Process from the shortcut menu.

The Move Package Properties dialog appears.

3. Name the process and specify a destination project and state:

To Project

Lists all the projects in the current CA Harvest SCM installation. Select the default destination project for the package. If you want to allow CA Harvest SCM users to select a project, select <any>. If <any> is selected, the To State field is also reset to <any>.

To State

Lists all the states in the current project. Select the default state in the destination project for the package. If you want to allow CA Harvest SCM users to select a state, select <any>.

The name and destinations are defined for the process.

4. (Optional) Include package history to move all package history records to the new package. If you do not select this option, the package history begins with the new project. The initial record indicates that the package was moved.

The process is defined.

5. Click Apply.

Saves the process definition, but does not close the dialog.

6. Click the Access tab and set access for the process. Click OK.

The move package process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Move Path Process

The move path process lets users logically move a path from the current location to another path. The movement is included with the changes associated with a package and can progress through the lifecycle as the package is promoted from one view to another. When a path has been moved, it no longer displays under the original path in the item view. The move path process creates a version located on the new parent path. This version has properties like other versions and can be seen under the package's Versions folder.

When you move a path, the move path process also creates one new version for each item and path under this path. All those versions are combined as one change; individual versions cannot be deleted. Deleting the original moved path version automatically deletes all the sub-item/path versions created by that move path process.

Linked processes execute before and after the move path process completes successfully. In this case, “successfully” means that at least one path is moved.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Move Path Process from the shortcut menu.

The Move Path Properties dialog appears.

3. Name the process and specify options:

On Trunk

Creates a trunk version for refactoring changes. (This option is identical to the check-out for update process.)

On Branch

Creates a branch version for refactoring changes. (This option is identical to the check-out for concurrent update process.)

The following mode options are applicable *only* for subitems:

Update and Release

Creates a normal version for the refactoring change. If a reserved version already exists in the same package, it is updated as a normal version.

Update and Keep

Creates a normal version for refactoring changes. Another new reserved version is created after the normal version.

The options are selected for the process.

4. Click Apply.

Saves the process definition, but does not close the dialog.

5. Click the Access tab and set access for the process. Click OK.

The move path process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Notify Process

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

The Notify Properties dialog lets you define the process and specify defaults for various execution options. When the user invokes the process execution dialog, the user can override the default values.

The notify process uses standard MAPI or SMTP mail services to let you send a message to any combination of users and user groups. You can add this process to a state or link it to other processes so designated users are notified automatically during the execution of the parent process.

Linked processes execute before and after the initial notify process completes successfully.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Notify Process from the shortcut menu.

The Notify Properties dialog appears.

3. Name the process and complete the dialog fields:

Mail Utility

Specifies a mail program and any arguments you want to supply when the program is invoked. On Windows clients, the server computer executes the mail program and must be in the path of the user who started the server.

Default: mail

Subject

Specifies a default subject. This subject appears on the subject line of the email message when it is sent. Users can modify the content of this field when it is not set up as a linked process. For AIX and Linux operating systems, the Subject field cannot contain double quotes.

Note: If you are using hsmtp.exe and define a subject separately in the hsmtp.arg file, do not include a value in this field.

Mail Message

Specifies a default mail message for the process. You can enter up to 2,000 characters. For example, if this process is going to be used to notify a manager that a package has been promoted, you can specify the appropriate message in this field. If the notify process is being linked to another process, you can use system variables in the messages to represent various parameters.

Output

Specifies a default action to be taken with any general or error outputs generated by the notify process. The choices are Display or Discard. All output is automatically appended to the output when the Display choice is selected.

Users to Notify

Specifies the users to receive the notification. You can add a user to the list by clicking Add, selecting the user, and clicking OK. You can delete a user from the list by selecting the user, and clicking Delete.

User Groups to Notify

Specifies the user groups to receive the notification. To be notified, users in the group must have use access to the project unless they are listed in the Users to Notify list. You can add a user group to the list by clicking Add, selecting the user group, and clicking OK. You can delete a user group from the list by selecting the user group, and clicking Delete.

The process is defined.

4. Click Apply.

Saves the process definition, but does not close the dialog.

5. Click the Access tab and set access for the process. Click OK.

The notify process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

More information:

[How to Set Up hsmtp](#) (see page 122)

Define a Remove Item Process

The remove item process lets users logically delete selected items from a view. The removal is made part of the changes associated with a package and can then progress through the lifecycle as the package is promoted from one view to another.

When an item has been removed from a view, it is no longer displayed in item view. However, the item is not deleted; the remove item process creates a new version tagged as removed (D). This version has properties like other versions and can be seen in the list view by double-clicking the item or through the Find Version dialog.

Linked processes execute before and after the remove item process completes successfully. In this case, “successfully” means that at least one item is removed.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Remove Item Process from the shortcut menu.

The Remove Item Properties dialog appears.

3. Name the process and specify options:

On Trunk

Creates a trunk version for refactoring changes. (This option is identical to the check-out for update process.)

On Branch

Creates a branch version for refactoring changes. (This option is identical to the check-out for concurrent update process.)

The options are selected for the process.

4. Click Apply.

Saves the process definition, but does not close the dialog.

5. Click the Access tab and set access for the process. Click OK.

The Remove Item process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Remove Path Process

The remove path process lets users logically remove a path. The removal is included with the changes associated with a package and can progress through the lifecycle as the package is promoted from one view to another. When a path is removed from the trunk, it no longer displays in the item view. The remove path process creates a version tagged as removed (D). This version has properties like other versions and can be seen under the package's Versions folder.

When you remove a path, the remove path process also creates one new version (D) for each item and path under this path. All those versions are combined as one change; individual versions cannot be deleted. Deleting the original removed path version automatically deletes all the sub-item/path versions created by that remove path process.

Linked processes execute before and after the remove path process completes successfully.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Remove Path Process from the shortcut menu.

The Remove Path Properties dialog appears.

3. Name the process and specify options:

On Trunk

Creates a trunk version for refactoring changes. (This option is identical to the check-out for update process.)

On Branch

Creates a branch version for refactoring changes. (This option is identical to the check-out for concurrent update process.)

The options are selected for the process.

4. Click Apply.

Saves the process definition, but does not close the dialog.

5. Click the Access tab and set access for the process. Click OK.

The remove path process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Rename Item Process

The rename item process lets you rename an item. After you execute the process, the rename item process creates a version with a new name. The previous versions of the item exist with their former names and you can see it in the version view of the item.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Rename Item Process from the shortcut menu.

The Rename Item Properties dialog appears.

3. Name the process and specify options:

On Trunk

Creates a trunk version for refactoring changes. (This option is identical to the check-out for update process.)

On Branch

Creates a branch version for refactoring changes. (This option is identical to the check-out for concurrent update process.)

Update and Release

Creates a normal version for the refactoring change. If a reserved version exists in the same package, it is updated as a normal version.

Update and Keep

Creates a normal version for refactoring changes. Another new reserved version is created after the normal version.

The options are selected for the process.

4. Click Apply.

Saves the process definition, but does not close the dialog.

5. Click the Access tab and set access for the process. Click OK.

The rename item process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Rename Path Process

The rename path process lets users logically rename a path. The renamed path is included with the changes associated with a package and can progress through the lifecycle as the package is promoted from one view to another. When a path has been renamed, it no longer displays the old name in the item view. The rename path process creates a new version with the new name. This version has properties like other versions and can be seen under the package's Versions folder.

When you rename a path, the rename path process also creates one new version for each item and path under this path. All those versions are combined as one change; individual versions cannot be deleted. Deleting the original renamed path version automatically deletes all the sub-item/path versions created by that rename path process.

Linked processes execute before and after the rename path process completes successfully.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Rename Path Process from the shortcut menu.

The Rename Path Properties dialog appears.

3. Name the process and specify options:

On Trunk

Creates a trunk version for refactoring changes. (This option is identical to the check-out for update process.)

On Branch

Creates a branch version for refactoring changes. (This option is identical to the check-out for concurrent update process.)

The following mode options are applicable *only* for subitems:

Update and Release

Creates a normal version for the refactoring change. If a reserved version already exists in the same package, it is updated as a normal version.

Update and Keep

Creates a normal version for refactoring changes. Another new reserved version is created after the normal version.

The options are selected for the process.

4. Click Apply.

Saves the process definition, but does not close the dialog.

5. Click the Access tab and set access for the process. Click OK.

The rename path process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Switch Package Process

The switch package process lets you move versions from a source package to a target package.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Switch Package Process from the shortcut menu.

The Switch Package Properties dialog appears.

3. Name the process and click Apply.

Saves the process definition, but does not close the dialog.

4. Click the Access tab and set access for the process. Click OK.

The switch package process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a Take Snapshot Process

The take snapshot process lets users create snapshot views of their current working view.

Snapshot views are read-only images of working views that were modified before or on a specified date and time. Snapshots let you capture a software inventory at significant points in its development. After you create a snapshot, you can use it to support other application management functions, such as baselining or recreating an application at a modified version date.

Linked processes execute before and after the snapshot is created.

Follow these steps:

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, Take Snapshot Process from the shortcut menu.

The Take Snapshot Properties dialog appears.

3. Name the process and click Apply.

Saves the process definition, but does not close the dialog.

4. Click the Access tab and set access for the process. Click OK.

The take snapshot process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

Define a User-Defined Process

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

The User-defined Properties dialog lets you specify an external program to run as a process in your lifecycle. You can add this process to a state or linked to any other process. Linking UDPs to other processes lets you execute any set of user-defined actions as a result of the previous action.

Note: If this process is linked to another process, all required input must be defined in the properties dialog.

When your site uses numerous UDPs where you must change each UDP definition manually, the maintenance becomes cumbersome. You can minimize your maintenance effort by avoiding the use of:

- Hard-coded names for system variables that CA Harvest SCM can evaluate.
- Absolute paths for executables.

Instead, as permanent alternatives, use the following options:

- System variables (for example, use [broker] for current broker name and [user] for current user name).
- Executables defined in your PATH environment variable instead of absolute paths (for example, add \$CA_SCM_HOME/bin to your PATH environment variable, instead of using an absolute path of /opt/ca/CA SCM/bin/command).

To define a user-defined process

1. Click the Lifecycles tab of the Administrator application, and expand the state in which you want to define the process.

The Processes folder is listed in the workspace.

2. Right-click the Processes folder, and select New, UDP Process.

The UDP Properties dialog appears.

3. Name the process and complete the dialog fields:

Program

Names the executable file that you want to run. Specify a full path to the program. CA Harvest SCM searches for a path in the PATH variable of the user invoking the process, or in the case of a server UDP, in the PATH variable of the server system. To locate and select a program, click the browser button to invoke the Windows file browser.

When the CA Harvest SCM hsql relational database query command line is used in the Program field:

- Specifying hsql (not hsql.exe), executes SQL input directly from your CA Harvest SCM session. You do *not* need to specify the broker, username, and password, and the program does *not* accept the input file.
- Specifying either hsql.exe on Windows or \$CA_SCM_HOME/bin/hsql on Linux or UNIX, executes the hsql command line. *Specify* the broker, username, and password options, and the input file.

Note: For more information about the CA Harvest SCM hsql command line, see the *Command Line Reference Guide*.

Type

Specifies the location of the program to be executed.

Default: Client.

Input

Specifies input to the program you have specified. Many programs read input from the default input device. This option only applies to programs that read from the standard input device.

Limits: Up to 2,000 characters of input can be specified.

Secure Input

Defines the information in the Input field as modifiable in the process execution dialog. By default, this field is not modifiable. To allow users to modify the default input, clear the Secure Input check box.

No Prompt

Specifies that a stand-alone client UDP (not linked to any process) does not prompt with a process execution dialog.

Allow Additional Parameters

Specifies that no additional command line parameters are provided for this UDP process.

Output and Errors

Specifies a default action to be taken with any general or error output generated by the associated process. You can choose one of the choices for both fields: Display or Discard. Selecting Display automatically appends the entire output.

Description

Specifies up to 255 characters of descriptive text for the user-defined process. This text shows on the process execution dialog and lets you inform users about what the process does. You are not required to enter anything in this field. If the process is linked to another, the description is invisible during execution.

The user-defined process is complete.

4. Click Apply.

Saves the process definition, but does not close the dialog.

5. Click the Access tab and set access for the process. Click OK.

The user-defined process is added to the state and the process name appears in the Processes menu and in the title bar of the process execution dialog.

More information:

[How to Set Up hsmtp](#) (see page 122)

[hrefresh UDP Definitions](#) (see page 222)

[Peer Review Verification and Comment Purging – hpeerreviews UDP](#) (see page 229)

Chapter 6: Administering Repositories

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

This section contains the following topics:

[CA Harvest SCM Repositories](#) (see page 95)

[Create a Repository](#) (see page 98)

[Load Repositories](#) (see page 99)

[How to Move, Remove, or Rename an Item or Item Path](#) (see page 101)

[Create an Item Path](#) (see page 101)

[Delete an Item Path](#) (see page 102)

[Delete an Item](#) (see page 102)

[Duplicate a Repository](#) (see page 103)

[Delete a Repository](#) (see page 104)

CA Harvest SCM Repositories

A repository is a collection of items and item paths. When you install CA Harvest SCM, no repositories exist in the installation. Administrators create repositories and control access to them.

Administrators can decide how they want to divide an application's code, using different repositories to build different products. For example, a site might have a repository for shared code (SHARED), a repository for application one (APP1), and another for application two (APP2). Projects that support a release of application one would be based on the SHARED and APP1 repositories, whereas projects associated with application two would be based on APP2 and SHARED.

The Repositories tab on the Administrator application shows all the repositories that you have access to. Expanding a repository shows the items and item paths that make up the repository. For every item and item path, the name that is displayed reflects the name given to the latest trunk version of the item and item path, respectively. When you navigate to an item path, you see the items whose latest trunk versions are below that path.

Repository Properties lets you create and define a CA Harvest SCM repository. You can also create an empty repository with all attributes copied from a source repository. The Duplicate Repository dialog lets you duplicate a CA Harvest SCM repository.

More information:

[Load Repositories](#) (see page 99)

File Extensions

Significant differences exist between UNIX, Windows, and z/OS operating systems that can affect updating files on these operating systems. Decide how you want to handle file conversion issues when you intend to use mixed clients. You must specify that CA Harvest SCM convert the end-of-line and end-of-file markers. The File Extensions tab of the Repository Properties dialog lets you specify how CA Harvest SCM performs these conversions. Because different repositories may contain different types of code, you can specify different conversion rules for each repository. In a repository, conversions are based on file type (extension). CA Harvest SCM does not automatically execute any file conversions on Windows clients because such a conversion can have negative effects on binary files. When you are loading a repository with files that are in Windows format, and that UNIX clients do not need to access, no conversion is needed.

Note: On the Workbench, you can display file type attributes (binary or text). For information about displaying file type attributes, see the *Workbench User Guide*.

Case Issues

CA Harvest SCM handles file character case as follows:

- Windows client files checked in to a Windows repository retain their original case, and checking in files of the same name regardless of case updates them. For example, File.c is stored in the repository exactly as loaded, and Windows client files FILE.C and file.c are interpreted as being the same file.
- Windows client files checked in to a UNIX repository are stored “case-aware.” For example, FILE.C and file.c are interpreted as being the same file.
- UNIX client files checked in to a UNIX repository are stored “case sensitive.” For example, FILE.C and file.c are interpreted as different files.

To help ensure that files have acceptable case on both UNIX and Windows operating environments, users must verify the case on new files being introduced into the repository (using the Load Repository function or by checking in new items), regardless of the client (Windows or UNIX) they are using.

File Conversion

Before you load files from a Windows client into a UNIX-based repository or from a UNIX client to a Windows-based repository, decide how you want to handle file conversion issues.

- UNIX files use the line feed character for end-of-line markers.

Because UNIX and Windows clients can share files stored in a UNIX-based repository, you can specify that files be converted to UNIX format when they are loaded or checked in. Then when a user checks out a file to a UNIX directory, the file appears as expected. When the user checks out the file again to Windows, the file appears in a Windows format.

- Windows files use a carriage return and line feed combination for end-of-line markers.

When a user checks out a file to a UNIX directory, a Windows file that is not converted to UNIX format has a ^M character at the end of every line. Such a file can still be edited and checked back in without any problems by ignoring the ^M at the end of each line.

If you edit files exclusively on Windows operating systems, these file conversions are not necessary, even when using a UNIX-based repository.

- If you use *both* UNIX and Windows clients, use the File Extensions tab of the Repository Properties dialog to select file options for the end-of-line conversion during check-in, check-out or load repository.

More information:

[Create a Repository](#) (see page 98)

Repository Access

When items are initially loaded, the user group Public, which includes all users, is granted view access to the items. You can use the Access tab of the Repository Properties dialog to change the user group access.

More information:

[Controlling Object Access](#) (see page 105)

Create a Repository

When you install CA Harvest SCM, repositories do not exist in the installation. You create repositories and control access to them.

Follow these steps:

1. Click the Repositories tab of the Administrator application.
2. Right-click the server in which you want to create a repository, and select New Repository from the shortcut menu.

The Repository Properties dialog appears.

3. Define the properties of the repository:

Name

Names the repository and is the beginning point (root) of all paths to items in the repository.

You can change the repository name. When you change a repository name, the repository base view name is also changed. If the repository is part of a snapshot view, a warning message appears and you can continue or cancel the rename operation.

Compress All Files Except

Manages binary files such as executables; this option is enabled by default to compress these files on check-in. You can exclude files from being compressed by typing their file extension in this field, or disable compression by clearing the check box.

This Repository is for MVS files ONLY

Stores the repository in PDS format, if selected, not a tree hierarchy structure. File extensions are not supported, and they cause errors during check-out to partitioned data sets.

The properties are defined.

4. Click the File Extensions tab, and select file options for the end-of-line conversion during check-in, check-out or load repository when you use *both* UNIX and Windows clients.

Do Not use any extension

Treats all files as binary files with no extensions.

Use Global text file extensions

Shows a list of file extensions that you can include in the repository.

Use Repository text file extensions

Adds an extension. You can add an extension by typing an extension in the File Extension field and clicking Add. You can remove an extension by selecting the extension in the list box and clicking Remove.

Note: If you want to treat files with no extensions as text files, use the File Extension field to add <NONE>.

The file extension behavior is specified.

5. Click Apply.
Saves the repository definition, but does not close the dialog.
6. Click the Access tab and set access for the repository. Click OK.
The repository is created and appears in the Repositories tab.

More information:

[File Conversion](#) (see page 97)

Load Repositories

The load repository function is similar to check-in. The load operation brings files located in operating system directories into the CA Harvest SCM repository. You can load only new items into a repository. If the items exist in the repository, an error is generated. Files that are loaded are not associated with any package.

Load a Repository

After a repository is created with the Repository Properties dialog, the Load Repository dialog lets you bring an application into CA Harvest SCM.

Follow these steps:

1. Click the Repositories tab of the Administrator application.
2. Right-click the repository you want to load, and select Load Repository from the shortcut menu.

The Load Repository dialog appears.

3. Complete the dialog fields:

Directory

Specifies the directory in the external file system where the files you want to load are located. This directory should contain the source files for the application that you want CA Harvest SCM to control. You can click the button next to the Directory field to open the Select Directory Path dialog.

Note: The case of file and directory names are preserved in the repository exactly as they are loaded from the client.

Files

Specifies the files to load. You can use a wildcard pattern (*) to include all matching files in the load operation. Typically, a single wildcard is used, but you can use any number of wildcards in any position. You can also use the question mark (?) for a single character match.

Recursive

Loads an entire directory structure into a CA Harvest SCM repository. Each directory that contains files is created and the files matching the file name pattern are loaded. If corresponding paths do not exist in the repository, they are created if the paths are not empty.

Create Empty Repository Paths

Creates matching empty directories in the repository if any exist, and those directories containing files. This option is only available when Recursive is selected.

Repository Path

Specifies the repository name in which you want to load the files and create paths. When you initially load an empty repository, no paths exist. Load all files that you want to include in this repository beneath this top-level name. Clicking the button next to the Repository Path field shows the Select a Repository Item Path dialog that lets you select the location in the repository for the loaded files.

The Repository Path field is useful later when you want to load files into a specific directory in the repository.

Comment

Provides comments about the items being loaded.

Click Load.

The repository is loaded.

How to Move, Remove, or Rename an Item or Item Path

You can move, rename, or remove a repository item or item path only if there is no project that contains a version of the item or item path.

To perform any of those functions on an item or item path that has a version in a project, do the following procedure:

1. In the Workbench, go to the project and state that has the item or item path you want to move, rename, or remove.
2. Execute the process that corresponds to the action you want to perform on the item or item path:
 - Move Item
 - Remove Item
 - Rename Item
 - Move Path
 - Remove Path
 - Rename Path

The Repositories tab of the Administrator interface shows the latest trunk version, regardless of the version's project location.

Create an Item Path

The New Item Path dialog lets administrators create item paths in existing repositories.

Follow these steps:

1. Click the Repositories tab in the Administrator application.
2. Right-click a repository or folder, and select New Item Path from the shortcut menu.

The New Item Path dialog appears.
3. Complete the fields in the dialog, and click OK.

The item path is created.

Delete an Item Path

You can delete an item path in existing data views when the item path meets the following requirements:

- The path and all its subitems are not associated to any project.
- The path and all its subitems are associated to only one project, even if that project already includes changes for them.
- The path and all its subitems are associated to more than one project, but the path and none of the items were changed in any project.
- The path does not have an item that was moved to the path from another location. This requirement is because this item might be an orphan item if the parent path is deleted; an error message appears in this case.

Note: If an item was initially under the deleted path, even it was moved to another location in another project using the move item process; this item is deleted. This behavior is because in the Administrator repository editor, the item is still under this path.

Follow these steps:

1. Click the Repositories tab of the Administrator application.
2. Right-click the item path you want to delete, and select Delete from the shortcut menu.

A confirmation dialog appears.

3. Click Yes.

The item path is deleted.

Delete an Item

You can delete an item in existing data views when the item meets any of the following requirements:

- The item is not associated to any project.
- The item is associated to only one project, even if that project already includes changes for it.
- The item is associated to more than one project, but none of the items were ever changed in any project.

Follow these steps:

1. Click the Repositories tab of the Administrator application.
2. Right-click the item you want to delete, and select Delete from the shortcut menu.
A confirmation dialog appears.
3. Click Yes.
The item is deleted.

Duplicate a Repository

You can create an empty repository with all attributes copied from the source repository.

The following properties are duplicated when you duplicate a repository:

- Access (if selected on the Duplicate Repository dialog)
- Compress option
- MVS Option
- File Extension options

Follow these steps:

1. Click the Lifecycles tab of the Administrator application.
2. Right-click the repository you want to duplicate, and select Duplicate Repository from the shortcut menu.
The Duplicate Repository dialog appears.
3. Name the new repository.
4. (Optional) Select the Duplicate Access Control check box if you want access control duplicated.
The repository is duplicated.

Delete a Repository

You can delete a repository.

Follow these steps:

1. Click the Repositories tab of the Administrator application.
2. Right-click the repository you want to delete, and select Delete from the shortcut menu.

A confirmation dialog appears.

3. Click Yes.

The Repository is deleted.

Chapter 7: Controlling Object Access

This section contains the following topics:

- [Object Access](#) (see page 105)
- [Object Access Relationships](#) (see page 106)
- [SCM-Level Access](#) (see page 107)
- [Object-Level Access](#) (see page 109)
- [Generate Access Reports](#) (see page 118)
- [Access Summary](#) (see page 118)
- [Access Hierarchy Summary](#) (see page 119)
- [Access Hierarchy List](#) (see page 120)
- [Administrator Access Summary](#) (see page 120)

Object Access

You can set varying access permissions for every CA Harvest SCM object. Access information is comprised of two parts:

- Method—Action that can be taken
- User group—Who can perform the action

You can secure some objects through access control: SCM, projects, states, processes, repositories, item paths, items, and form types. Other objects are secured through an object to which they belong. For example, views, packages, and package groups are all secured through the project or state to which they belong.

You use the object properties dialog to grant access to one or more user groups. The access allows members of the groups to perform a particular type of action (method) on the object, for example:

- At the highest level, access granted to the group Repository Manager allows members to administer repositories.
- At the process level, access granted to the Developer group Developer allows members to check in and check out files.

Object Access Relationships

The following table shows the access relationships among the various CA Harvest SCM objects.

Access Category	Object	Based On User Group	Methods
None	None	Administrator	All
Administrative	CA Harvest SCM	Any	Secure SCM Admin Project View Project Admin Repository View Repository Admin Form Type View Form Type Admin User/User Group View User/User Group
Object	Project	Any	View Update Secure Use
Object	State		Update Update Package
Object	Process		Execute
Object	Repository		View Update Secure
Object	Item Path/Item		View
Object	Form Type		View Update Secure

SCM-Level Access

The highest level of access controls administrative access to the SCM-level objects. These objects are projects, repositories, users and user groups, and form types.

The access methods for these objects are as follows:

Secure SCM

Defines who can grant access control for the following SCM-level objects: Admin Project, View Project, Admin Repository, View Repository, Admin User/Group, View User/Group, Admin Form Type, View Form Type. Having Secure SCM access does not imply being able to delete or edit SCM-level objects. Users with access to the secure method must explicitly give themselves access to other methods; they are not automatically available.

Admin Project

Defines who can create, delete, view, and set access to all projects and their aggregate objects (states, processes). For example, when a user creates a project, that user can give the ownership of the project to a user group at the object-level by assigning Secure Project access. This access also means that users can give themselves access to do anything in the projects they have created.

View Project

Defines who can view the properties of all projects inCA Harvest SCM. User groups with View Project access can also view the properties of all objects (states, processes, packages, package groups, and views) in the project.

View Project access does not prevent a user from being able to modify objects in the project. For example, a user with View Project access can have Update access to states in that project. In this case, the user can modify the state properties, but not the project properties.

In addition, lacking Admin Project access does not prevent a user from being able to modify a particular project. For example, a user with View Project access can have object-level Update access to a particular project. In this case, the user can modify the project properties and add, delete, and update the states and processes in that project.

Admin Repository

Defines who can create, delete, view, and set access to all repositories and their aggregate objects (items and item paths). This access method implicitly grants view access to all items and item paths in all repositories.

View Repository

Defines who can view all repositories inCA Harvest SCM. Users who have view access to a repository can see the properties of the repository and the items and item paths in it, provided that item/path view access is granted at the item/path level.

Admin User/User Group

Defines who can create, delete, edit, and view users and user groups.

View User/User Group

Defines who can see the properties of user and user group records. View access allows the user to see the information that shows in the object's properties dialog, but in a read-only mode.

Admin Form Type

Defines who can create, update, delete, and set access to form types. Users with Admin Form Type access can give themselves or others access to edit the form types they created.

View Form Type

Defines who can view the properties of all form types in CA Harvest SCM.

Lacking Admin Form Type access to an object does not prevent a user from being able to update forms. For example, a user with View Form Type access can update forms of that form type. In this case, the user can modify the form properties but not the form type properties. Although forms are not securable objects, their access is controlled by the packages they are associated with and package access is controlled by Update Package access at the state level.

Set SCM-Level Access

You set and modify the SCM-level of access by using the Access Control dialog. This dialog is available on all tabs of the Administrator application.

You can modify access whenever necessary; however, consider the following caution:

Important! CA Harvest SCM reads object-access information when the object is accessed. When you are using an object and change its access, the change does not take effect until you exit the dialog and reload the object information.

Follow these steps:

1. Click any tab of the Administrator application.
2. Click SCM, Access Control.

The Access Control dialog appears.

3. Select the access type you want to grant to a user group from the Access Type drop-down list.

The user groups that are granted the access type selected in the Access Type are listed in the User Groups with Access list.

4. Click Add.

A dialog appears that lists all available user groups.

5. Select one or more user groups to grant access to and click OK.

The selected user groups are listed in the User Groups with Access list.

6. (Optional) To remove a user group from the list, select the user group, and click Delete.

The user group is removed from the list.

7. Click Close.

The user groups are granted access and are included in the list of user groups that have been granted access to the object.

Object-Level Access

Object-level access defines the access to a single object. These objects are securable objects and they are projects, states, processes, repositories, item paths and items, and form types.

Project Access

The access methods for a project are as follows:

View Project

Allows users to view a project and its associated lifecycle, view all views in the project, view the packages, package groups, and review requests that are in the project.

View access to an object does not prevent a user from being able to modify objects in the original object. For example, a user with View access to a project can have Update access to states in that project. In this case, the user can modify the state properties, but not the project properties.

Update Project

Defines who can view and modify the properties of a project and all of the objects (states, processes, packages, package groups, and views) in a project. With Update project access, the user can also set access to states and processes.

This method of access also defines who can create and delete package groups, states, processes, views, and who can delete packages. State and process Update access can also be set at the state level by using the Update Access method. This is useful when you want to allow only specific groups to have access to particular states and processes but not the project.

Secure Project

Defines who can set access for the project. The secure project method does not imply being able to delete or update an object. Users with access to the secure project method must explicitly give themselves access to other methods; they are not automatically available.

Use Project

Allows users to view a project and its associated lifecycle, view all views in the project, view packages and create, delete, update package groups and review requests that are in the project. To run any process in a project, the user must have Use access at the project level with Execute process access at the process level.

State Access

The access methods available at the state level are as follows:

Update

Defines who can view and modify the properties of a state and its processes. This method of access is useful when you want to allow only specific groups to have update access to a particular state and its processes. Even if users do not have Update state access, if they have Update project access, they can update states and their processes.

Update Package

Defines who can view, delete, and modify the properties of a package. This method of access is useful when you want to allow only specific groups to have update access to packages located in a particular state. This means package ownership changes throughout the lifecycle (states).

Process Access

Processes let users update data in CA Harvest SCM and perform tasks on the Workbench. At the process level, the only access method is Execute access. Execute access defines who can perform a process. For example, Execute access for a check-out process defines who can perform the check-out. Processes that are linked to another process are not secured separately like processes added to a state. When the user has access to execute the initial process, access is assumed for any other processes that are linked to the initial one.

Execute access of a process is only effective when you use it with Use project access. Admin Project access does not automatically grant Use project access or Execute process access. These two methods are effective only on the Workbench.

Item Path and Item Access

The access method for item paths and items is View access. View access defines who can view an item or item path and, in the case of an item path, all of its contents recursively including sub-paths and items.

New access is implicitly granted to Public when loading a repository that creates an item or path, or when checking in items. Users set access of an item only when they want to modify access to that item or path. This behavior avoids manual access setup for item or path objects. From the Workbench, the user does not need View Repository access to see an item or item path; the user simply needs View access to the item or item path. Users who do not have View access do not see the item in any CA Harvest SCM window or dialog.

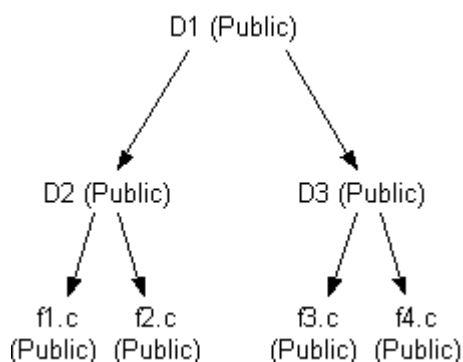
The list of User Groups with Access on the Access tab of an object's Properties dialog includes the user groups that have been granted View access to the item. When a user group is deleted from the list, it is not granted View access to the item. If a user is a member of at least one user group that has been granted View access to an item, the user is granted View access to the item.

To check out an item for Browse or Synchronize, a user needs at least View access. To check out an item for Update or Concurrent Update, or to reserve an item, the repository to which this item belongs should be assigned to the baseline with the Read/Write option instead of the Read/Only option. This type of access is different from other access methods and it is set in the Configure Baseline dialog of the Administrator application project setup.

GRANT methodology on item access is as follows:

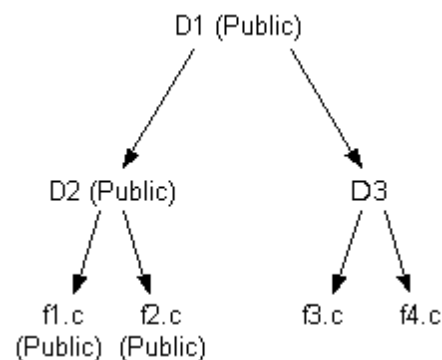
- Adding View access to an item propagates downwards to *all* its descendants and upwards to *all* its ancestors. (A *descendant* is an item or item path that is reached by navigating down from a given item in the Administrator interface to repository items. An *ancestor* is an item path that is reached by navigating up from a given item in the Administrator interface to repository items.)
- Removing View access from an item propagates *only* downwards to *all* its descendants.
- Only the item access changes are propagated.
- New items created using load repository or check-in items inherit View access settings from the parent directory.
- Items created at the root of a repository inherit Public as View access.

For example, the following illustration shows D1 at the root of the repository:



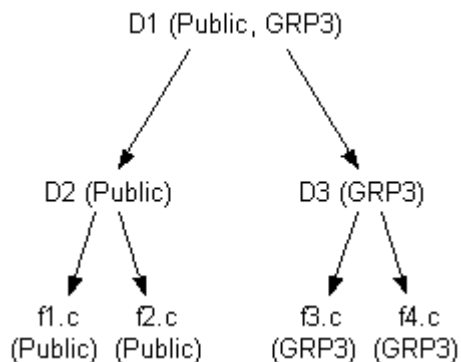
Note: (Public) is inherited from the root of the repository by D1 and *all* its descendants.

For example, the following illustration shows REMOVE (Public) View access from D3:



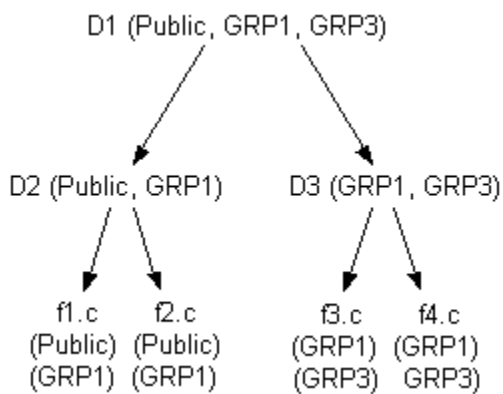
Note: The removal of (Public) has *only* propagated downwards to *all* descendants of D3.

For example, the following illustration shows GRANT (GRP3) View access to D3:



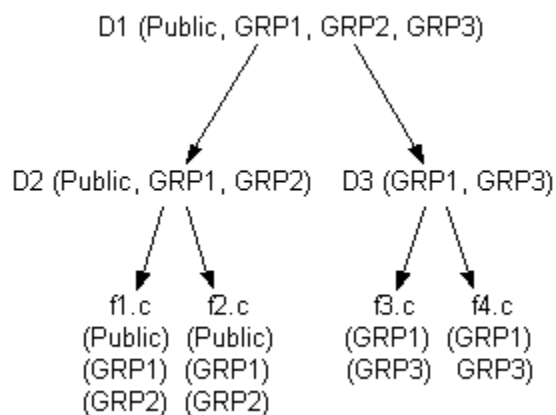
Note: (GRP3) has propagated downwards to *all descendants* of D3 and upwards to *all ancestors* of D3.

For example, the following illustration shows GRANT (GRP1) View access to D1:



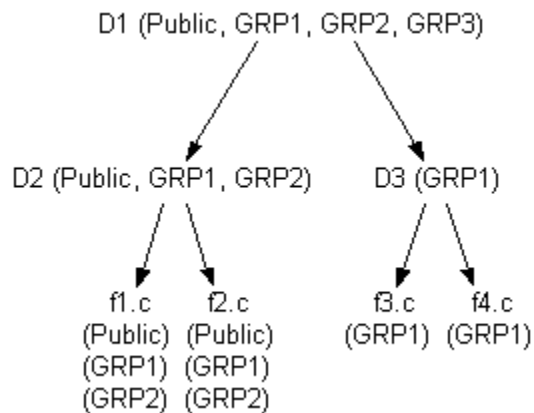
Note: (GRP1) has propagated downwards to *all descendants* of D1. *Only* the change to D1 (addition of GRP1) has been propagated.

For example, the following illustration shows GRANT (GRP2) View access to D2:



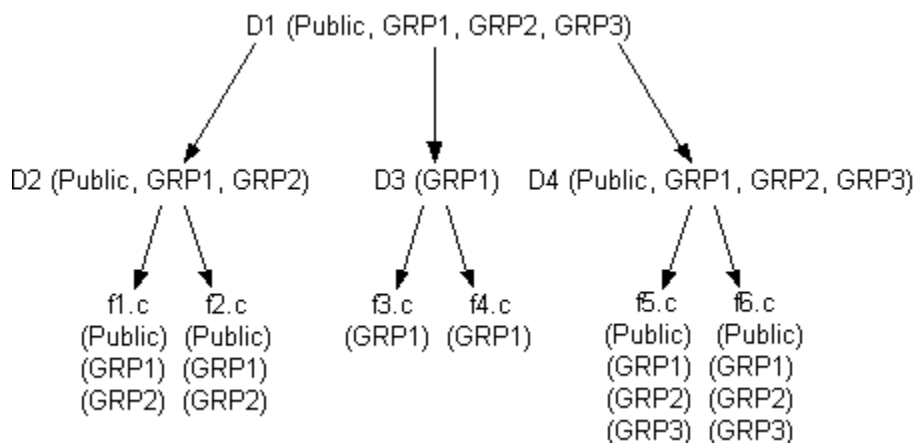
Note: (GRP2) has propagated downwards to *all descendants* of D2 and upwards to *all ancestors* of D2.

For example, the following illustration shows REMOVE (GRP3) View access from D3:



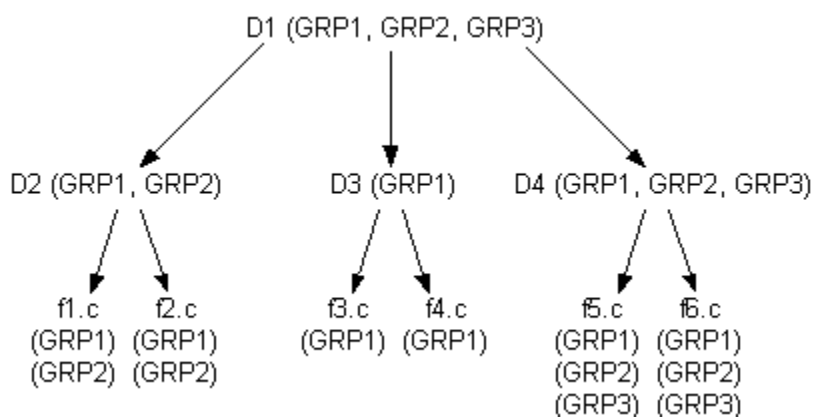
Note: The removal of (GRP3) has *only* propagated downwards to *all descendants* of D3.

For example, the following illustration shows LOAD D4 below D1:



Note: (Public, GRP1, GRP2, GRP3) are inherited from D1 by D4 and *all* its *descendants*.

For example, the following illustration shows REMOVE (Public) View access from D1:



Note: The removal of (Public) has propagated downwards to *all descendants* of D1.

Repository Access

The repository access methods are View, Update, and Secure. When you add a user group to the access list, that group is granted access to that object.

View

Defines who can view a repository and its properties. Users who have view access to a repository can see the properties of the repository and the properties of items and item paths in it when the item/path view access is granted at the item/path level.

Update

Defines who can view and modify the contents of a repository. The update method includes executing the load function, and when that item/path view access is granted at the item/path level, setting access to items and item paths in the repository, renaming items, deleting items and paths globally and permanently from the Administrator application. This access implicitly grants view repository access.

Secure

Defines who can add and delete access control records for a repository. The secure method does not imply being able to delete or update a repository. Users with access to the secure method must explicitly give themselves access to other methods; they are not automatically available.

Form Type Access

Form types have the following access methods:

View

Defines who can view a specific form type (for example, the Modification Request form type). View access does not automatically grant access to the specific forms of that type.

View access to a form type does not prevent a user from modifying forms. For example, a user with view access to a form type can have edit access to forms of that form type. In this case, the user can modify the form properties, but not the form type properties. Package access controls form access because forms are considered part of package properties. You cannot create or delete forms as single objects but only with packages.

Update

Controls which users can view and modify the properties of a form type.

Note: Form type modifications are written to files on the local file system. The installation-wide update of a form type is external to CA Harvest SCM. Therefore, CA Harvest SCM does not currently require the user to have Update access to modify form type properties.

Secure

Defines who can grant view or update access control records for a form type. The secure method does not imply being able to delete or update a form type. Users with access to the secure method must explicitly give themselves access to other methods; they are not automatically available.

Set Object-Level Access

You set access for an object through the object's Properties dialog. You can modify access whenever necessary; however, consider the following information:

Important! CA Harvest SCM reads an object's access information when it is first accessed. When you are using an object property dialog and change access, this change does not take effect until you exit the dialog and reload the object information.

Follow these steps:

1. Click any tab of the Administrator application.
2. Right-click the object you want to grant access, and select Properties.
The object's Properties dialog appears.
3. Click the Access tab.
The Access Type drop-down list displays a list of methods that varies with different objects.
4. Select the access type you want to grant to a user group from the Access Type drop-down list.
The user groups that are granted the access type selected in the Access Type are listed.
5. Click Add.
A dialog appears that lists all available user groups.
6. Select one or more user groups to grant access to and click OK.
The selected user groups are displayed in the User Groups with Access list.
Note: To remove a user group from the list, select the user group, and click Delete.
7. Click OK.
The user groups are granted access and are listed with the user groups that have been granted access to the object.

More information:

[Process Access](#) (see page 63)

Generate Access Reports

You can generate access reports that show the user and user group access in a project (Project Access), and in a repository (Repository Access). The reports use the respective vbscript files ProjAccessRpt.vbs and RepAccessRpt.vbs that are located in CA_SCM_HOME\VB. In these files, initialize the username and password variables with valid admin credentials to generate the reports. (With the installation, the username and password variables are initialized with “harvest”.)

Use the haccess command to generate reports from the command line, or click the Project Access Report or the Repository Access Report icons on the taskbar.

Reports are displayed in table format in the list view.

Note: For more information about the command line (haccess) utility, see the *Command Line Reference Guide*.

Access Summary

The following table is a quick reference to the access methods available for each CA Harvest SCM object and the actions those methods allow.

Object	Methods	Description
CA Harvest SCM	Secure SCM	Grant access rights to CA Harvest SCM objects.
CA Harvest SCM	Admin Project*	Create, delete, view, and set access to projects.
CA Harvest SCM	View Project*	View projects.
CA Harvest SCM	Admin Repository*	Create, delete, view, and set access to repositories.
CA Harvest SCM	View Repository*	View repositories and item/item path properties.
CA Harvest SCM	Admin Form Type	Create, delete, view, and set access to form types.
CA Harvest SCM	View Form Type	View form types.
CA Harvest SCM	Admin User/User Group	Create, delete, edit, and view users and user groups.
CA Harvest SCM	View User/User Group	View users and user groups.
Project	View*	View project and lifecycle, view views, and view packages, package groups, and review requests.
Project	Update*	Edit project, edit, and view lifecycle (states and processes), maintain baseline.
Project	Secure	Grant access rights to project.

Object	Methods	Description
Project	Use	View project and lifecycle (states and processes), view views and view packages, view package groups, view review requests, create package groups, create, delete, and update review requests, execute processes (not a sufficient but a necessary requirement for process execution).
State	Update*	Create, edit, delete, and view states and processes.
State	Update Package	Edit packages and forms associated with packages.
Process	Execute	Execute processes.
Repository	View*	View repository, its attributes, its items and item paths.
Repository	Update*	Edit and view repositories, and edit, delete, set view access to items and item paths globally and permanently (Administrator application).
Repository	Secure	Grant access rights to repository.
Item Path/Item	View*	View item or item path.
Form Type	View	View form type.
Form Type	Update	View and modify the properties of a form type. Note: Form type modifications are written to files on the local file system. The installation-wide update of a form type is done outside CA Harvest SCM. Therefore, CA Harvest SCM does not currently require the user to have Update Form Type access to modify form type properties.
Form Type	Secure	Grant access rights to a form type.

*Option is provided to cascade to the aggregate object access control.

Access Hierarchy Summary

The following table shows the hierarchical nature of the CA Harvest SCM access control system. For example, the row ADMIN PROJECT shows that the SCM-level Admin Project access is equivalent to project-level Update Access for all projects.

SCM-Level	Object-Level
ADMIN PROJECT	is equivalent to UPDATE ACCESS all Projects
ADMIN REPOSITORY	is equivalent to UPDATE ACCESS all Repositories

VIEW PROJECT	is equivalent to VIEW ACCESS all Projects
VIEW REPOSITORY	is equivalent to VIEW ACCESS all Repositories
VIEW FORM TYPE	is equivalent to VIEW ACCESS all Form Types

Access Hierarchy List

The following list shows the hierarchical nature of the CA Harvest SCM access control system. For example, the line Admin Project shows that Admin Project is higher in the hierarchy than View Project access. Therefore, a user group with Admin Project access implicitly has View Project access.

Admin Project is higher than View Project at the highest level.

Admin Repository is higher than View Repository at the highest level.

Admin User/ Group is higher than View User Group.

Admin Form Type is higher than View Form Type at the highest level.

Update Project is higher than View Project at the project level.

Update Repository is higher than View Repository at the repository level.

Update Project is higher than Update Access at the state level.

Update Project is higher than Update Package at the state level.

Administrator Access Summary

CA Harvest SCM Administrator access is summarized as follows:

ADMINISTRATOR=SECURE SCM+ADMIN PROJECT+ADMIN REPOSITORY+ADMIN USER/GROUP+ADMIN FORM TYPE

Chapter 8: Setting Up the Mail Utilities

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

This section contains the following topics:

[Mail Utility Programs](#) (see page 121)

[How to Set Up hmail](#) (see page 121)

[How to Set Up hsmtp](#) (see page 122)

Mail Utility Programs

CA Harvest SCM includes the following mail utility programs that work with the notify process:

- **hmail**—The hmail utility uses Windows mail systems that support the Messaging Application Programming Interface (MAPI), such as Microsoft Mail and Microsoft Exchange. You enable the simple MAPI client through your mail facility options.
- **hsmtp**—The hsmtp cross-platform utility uses the Simple Mail Transfer Protocol (SMTP) interface.

Although mail is executed on the server, a line is added to the message header identifying the name of the client user who initiated the notification process.

How to Set Up hmail

To set up hmail, create a notify process as follows for your operating system:

- (Windows) On the server, set the Mail Utility field on the Notify Properties dialog to hmail and specify the mail profile name using the -usr flag to interface with the system mail program. An example Mail Utility field follows:

```
hmail -usr <mail profile name>
```

- (UNIX) Set the Mail Utility field on the Notify Properties dialog to the appropriate mail agent for your system. An example Mail Utility field follows:

```
/usr/bin/mail
```

Note: On Hewlett-Packard computers, set the Mail Utility field to mailx.

How to Set Up hsmtp

To set up the hsmtp mail utility, do the following:

- Create an argument file named hsmtp.arg in the %CA_SCM_HOME% directory. For a list of hsmtp arguments and their descriptions, see the following list.

Note: You can also execute hsmtp directly from the command prompt. Arguments specified on the command line override the arguments specified in the argument file. If no options are specified in the argument file or on the command line, the defaults are used.

- Create a notify process or notify UDP process.
- In the Mail Utility field of the notify process, enter hsmtp.exe and its arguments. The Mail Utility field should look similar to the following:

```
hsmtp.exe -m smtpmail.company.com -p 25 -f sender@company.com  
recipient(s)@company.com
```

This command has the following arguments:

-m server_name | IP_address

(Required, if not specified in the hsmtp.arg file.) Specifies the name or IP address of the SMTP mail server that sends the email.

-p port_number

(Optional) Specifies the port number. By default, SMTP listens on port 25. You can use this option to change the default port.

-f address

(Required, if not specified in the hsmtp.arg file.) Specifies the sender's email address.

-s subject_line

(Optional) Specifies an individual subject line in the email. If not specified, hsmtp uses the default subject: "Message from CA Harvest SCM Change Manager."

-d

(Optional) Prints debugging information to the standard output device.

-cc address

(Optional) Specifies email addresses of persons who should receive a carbon copy of the email.

-bcc address

(Optional) Specifies email addresses of persons who should receive a blind carbon copy of the email.

-delay number

(Optional) Specifies the number of seconds to wait between retries and before trying the next server in the list when the retries for the current server have been exhausted. For Common Gateway Interface (CGI) scripts, specify few retries and short delays to return with a results page before the HTTP connection times out. For unattended scripts, you may want to use many retries and long delays to increase the probability of your mail being sent even with temporary failures on your network.

Default: 1 second

Note: This option is not available for the command line; you can specify this option in the hsmtp.arg file.

-retries number

(Optional) Specifies how many times the connection to the same SMTP server retries when a failure occurs.

Note: This option is not available for the command line; you can specify this option in the hsmtp.arg file.

-contenttype text or html

(Optional) Specifies the content type of the email. The default type is text/plain; the other option is text/html.

Note: This option is not available for the command line; you can specify this option in the hsmtp.arg file.

-charset character set

(Optional) Specifies the character set that the email body uses.

Default: The default is iso-8859-1. You can specify other character sets, for example, koi8-r (Korean), utf-8.

Note: This option is not available for the command line; you can specify this option in the hsmtp.arg file.

A sample hsmtp.arg file follows:

```
-mailserver=mail3.mycompany.com
-port=25
-subject=This is a message from CA SCM Change Manager
-from=abc@mycompany.com
-cc=def@mycompany.com
-bcc=xyu@mycompany.com
-delay=3
-retries=5
-charset=iso-8859-1
-contenttype=text/html
```

More information:

[Define a Notify Process](#) (see page 84)

[Define a User-Defined Process](#) (see page 92)

Set Up Variables for the Email Subject with hsmtp

You can set up hsmtp to use variables for the email subject as follows:

1. Verify that no subject is defined in the hsmtp.arg file.
2. Open the Notify Properties dialog that you want to use for email.
3. Enter a space character in the Subject field in the Notify Properties.

The subject uses the first line in the email body as the subject when the line starts with the word Subject followed by a space and a colon (:).

4. After the colon, specify a line with variables that the email will use as the subject.

Sample:

```
Subject : Notification from project [project]. User [user], promoted the  
package(s) [package] to the state [nextstate]
```

5. Save and close the dialog.

Hsmtp is set up to use variables for the email subject.

Chapter 9: Creating and Modifying Form Types

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

This section contains the following topics:

- [Default Form Types](#) (see page 125)
- [How to Create Form Types](#) (see page 127)
- [How to Design Form Types](#) (see page 128)
- [Create a Form Type](#) (see page 133)
- [Convert Form Definition Files to XML Files](#) (see page 139)
- [Store the XML Templates From the Form Reference Folder](#) (see page 139)
- [How to Modify a Form Type by Editing the XML File](#) (see page 140)
- [How Forms Use JavaScript Scripts](#) (see page 149)
- [Print a Form Type](#) (see page 153)
- [Add the Form Table to Your CA Harvest SCM Database](#) (see page 153)
- [Form Type Deletion](#) (see page 155)

Default Form Types

CA Harvest SCM includes the following default form types to assist users in tracking change information. Administrators can also create customized form types to fit their needs.

Application Change Request

Records information about changes that are associated with SAP/R3 components. This form is associated with the create package process in the Packaged Application Change Management template.

Comment

Records any type of information and associates it with a package if needed.

Defect Tracking

Records information that Help Desk applications use. This form lets CA Harvest SCM users, who are updating their Defect Tracking forms, populate the Help Desk form fields by using the command-line utilities.

ESD Change Request

Records information that pertains to an electronic software distribution project. After the form is created, the ESD coordinator evaluates, prioritizes, and documents the request. The ESD coordinator assigns an urgency level to the package and documents the change in the form with a business justification, risk analysis, and so on.

Modification Request

Records all data regarding a software change request. Modification requests are typically associated with the package used to resolve the problem.

Problem Report

Records the problems that initiate changes to code that CA Harvest SCM controls. Problem Reports are typically associated with the package used to resolve the problem.

Q and A

Creates a database of information in question and answer format. Such a database can be an effective support tool for a help desk or customer support group. Keyword searches can be performed to retrieve answers.

Testing Info

Assists in the package turnover process and is typically associated with a package. Developers can record information of interest to the users who perform tests or QA. In turn, testing and QA personnel can record their responses for management review or to assist developers if the package is rejected.

USD Package Information form and USD Platform Information

Specifies the parameters for deploying software from CA Harvest SCM through CA Software Delivery (previously named Unicenter Software Delivery):

- **USD Package Information** lets you record basic software delivery parameters such as the USD package name prefix and the deploy time.
- **USD Platform Information** lets you record specific details for a software deployment for a particular platform, including the operating system, the computers or computer groups on which to deploy the software, and the actual software components (files) used to install and deploy the software.

The USD forms are associated with the create software deployment package process in the Deploy Release template.

To use the Deploy Release template to deploy software, you must have both CA Harvest SCM and CA Software Delivery installed and configured.

User Contact

Creates a database of customers or other product users. User forms can then be associated with problem forms that the customer submits. The information is organized so that a mailing list can be generated.

How to Create Form Types

Form types use the following files:

CA Harvest SCM Form Definition (HFD) file

Names the form type and defines the layout of form fields.

XML form template

Stores the definition in the CA Harvest SCM database table on the CA Harvest SCM server.

Displays as a form in the Form Editor in the Workbench or the Web Interface.

Use one or more of the following utilities to create form types:

Form Wizard

Creates an HFD file that defines the form type.

CA Harvest SCM Form Converter

Converts an HFD template file to an XML file.

CA Harvest SCM relational database table

Stores the data from each form in a relational database table.

hformsync (Update Form Template) Command Line Utility

Stores the XML templates from the form reference folder into the CA Harvest SCM database.

Generate Form page

Adds the XML template to the CA Harvest SCM database for use the Web Interface.

To create a form type, do the following basic steps:

1. (Optional but highly recommended) Design the form type.
2. Use the Form Wizard to create the form type and database table definition (HFD file).

3. Use the CA Harvest SCM Form Converter to convert the HFD file to an XML file.

Note: One HFD file and one XML template exist for each form type.

4. (Optional) Use a text editor to edit the XML template to include form controls, such as image and hyperlink control types and additional features, such as field pattern validation.
5. (Optional) Use the FormEditor application programming interface (API) to edit form templates with additional customizations.

Note: To learn more about this interface and the customizations available to you, see the FormEditorAPI Java Doc which is located in your Windows client installation at `CA_SCM_HOME\Docs\FormEditorAPIDoc`.

6. Add the form table to your CA Harvest SCM database by using SQL scripts to define the relational database table for a form type.

More information:

[How to Design Form Types](#) (see page 128)

[Create a Form Type](#) (see page 133)

[Convert Form Definition Files to XML Files](#) (see page 139)

[How to Modify a Form Type by Editing the XML File](#) (see page 140)

[Add the Form Table to Your CA Harvest SCM Database](#) (see page 153)

How to Design Form Types

Before you use the Form Wizard, it is helpful, although not required, to plan a form type design. CA Harvest SCM form types can contain any number of fields on single or multiple pages. After you compile a full inventory of fields, you create the form; you can use the Form Wizard to create the form.

To design a form type, do the following:

1. Consider the use of the form and how the user will interact with it.

2. Compile a list of the fields (form control types) that you want to include in the form. (You refer to the list when you use the Form Wizard.) Include the following information for each field in the list:

- Field label name
- Database column name
- Field Type
- Default field values
- Fields that must be validated
- For combo box and radio button set fields, the item or list values
- For text fields, the length of the field (maximum number of characters)
- For tab controls, the page labels

Note: The Form Wizard automatically adds the Form Name field to each form type as the first field on the form. You do not need to include this field in your inventory list or define it when using the Form Wizard.

3. Order your inventory list in the sequence that you want the form to list the fields. If you are creating a multiple page form, list fields for each page in a separate sequence.
4. Include other control types, such as labels and lines, in this list.

Important! The GUI Workbench supports the following features and the Web Interface does *not* support them:

- Multi-column form page layout
- Image field
- Hyperlink field
- Rich label text
- JavaScript form scripts (VBScript scripts are not supported.)

Control Types

Form controls let users interact with forms. You can define control attributes by using its corresponding dialog in the Form Wizard. Following are descriptions of each control type and its corresponding dialog fields that are not self-explanatory:

- **Check Box**—A check box lets users select or clear an option. The field type is either enabled (checkmark) or disabled (no checkmark).

- **Combo Box**—A combo box control (also known as a drop-down list) lets users select a value for a field from a list of values. Combo boxes are useful for requiring users to select only the values that are valid for a field. When a user clicks a combo box, a list of all possible values appears. After the user selects a value, the value shows in the field.

To add a combo box list item, click the Add button located next to the Contents toolbar on the Add a new form field dialog. This action opens the Add New Combo Item dialog in which you can name an item to show in the combo box list.

- **Database Combo Box**—A database combo box control (also known as drop-down lists) lets users select a value for the field from a list of values. The values for this selection list are retrieved from the relational database using Structured Query Language (SQL) statements.

Database combo boxes are useful for requiring users to select only the values that are valid for a field. When a user clicks a database combo box, a list of all possible values appears. After the user clicks a value, the selected value shows in the field. For example, an SQL statement could retrieve all CA Harvest SCM users so the database combo field on the form would list all the CA Harvest SCM users. The user could then select a user name to populate the form field.

The SQL Commands field lets you enter SQL command statements to retrieve database information that the database combo box field uses. The SQL statement must include the object ID. The object ID does not show in the combo control list.

For example, to show a list of all CA Harvest SCM user names stored in the haruser table, a query on the CA Harvest SCM username follows:

```
select usrobjid, username from haruser
```

- **Date**—A date, or calendar, control lets users select a date from a calendar. Clicking a scroll arrow next to the date field, opens the calendar, which shows the current month. The user can move to previous and subsequent months by using the arrows located at the top of the calendar. Clicking a date, selects it, returns it to the form field, and closes the calendar. By default, a date control shows the current date.
- **Label**—A label control identifies and helps organize fields according to a type of information. Labels are nonmodifiable text blocks.
- **Line**—A line control provides you with additional formatting for a form type. You can use line controls in a similar way as in a page layout.
- **Radio Button Set**—A radio button set control lets you group values from which the user can only select one. Unlike a combo box, a radio button set shows all possible values for the field. Radio button sets are useful for requiring users to select only the values that are valid for a field.

- **Tab**—A tab control creates pages, identified by tabs, on a form. When the user clicks a tab or uses the arrow keys on the keyboard, that page moves to the foreground and the user can browse or modify the field contents. Tab pages are useful for forms that contain many fields. Tabs also help organize fields so the user can easily enter or locate information.

You can only add one tab control per form, but you can specify as many pages (labels) as you want. An efficient method of creating your form is to add a tab control and name each page (label), and then add fields to each page. To add a tab label and a tab page, click Add (+) located next to the Tab Labels label. This action opens the Add New Tab Label dialog in which you can create and name a tab that identifies a page in your form. You can also remove a table label or change the tab label order.

- **Text Edit**—A text edit control lets users edit text fields that can contain any alphanumeric data. This control is the most flexible field type and you can use it for any form field.

You cannot use the following form controls:

- Group boxes that contain form controls. You can use line and label fields to replace group boxes.
- Buttons

Note: You can include control types that the Form Wizard does not support by manually editing the XML file.

More information:

[How to Modify a Form Type by Editing the XML File](#) (see page 140)

Multiple Page Forms

Multiple pages are useful for form types that contain many fields. Multiple pages organize fields so that the user can easily enter or locate information. A tab control creates pages that tabs identify on a form. When a user clicks a tab, that page moves to the foreground and the user can browse or modify the field contents. When you want to create a tab control, the Form Wizard leads you through the field definitions for each tab page. You can use the edit functions, available on the shortcut menu, to relocate fields and modify your form at anytime.

To create a multiple page form, use the following method:

1. Add a tab control.
2. In Tab Control Properties, add a tab label for each tab in the form.

The tabs are identified.

3. Click Next, navigate the first tab, and add fields.
4. Click Next, continue adding fields and navigating through the tabs until you reach the last tab.

Note: The title bar on the Form Wizard does not indicate the tab location until you add a field to the tab.

When you want a form to display general information fields at all times, you add these fields *before* adding a tab control. For example, if a form has an information area that shows customer name and address fields, and you want this area to be visible no matter what tab the user selects, create the tab control after the customer name and address fields, as shown in the following table:

Type	Label Name
Text Edit	Customer Name
Text Edit	Customer Address
Tab Control	Application Problem
<i>first field</i>	<i>on Application tab page</i>
<i>second field</i>	<i>on Application tab page</i>
<i>third field</i>	<i>on Application tab page</i>
<i>first field</i>	<i>on Problem tab page</i>

A Sample Compilation of Form Fields

The following table is a sample compilation of all the fields that a problem report form requires. A more detailed table could also list the height and width of the tab and text fields.

Form Name: Brickyard Problem Report

Table Name: brproblemreport

Tab Page	Type	Label Name	Database Column	Length	Item/Menu Value
	Tab Control	Customer Application		2 labels	
One	Date	Date Reported	datereported		

Tab Page	Type	Label Name	Database Column	Length	Item/Menu Value
One	Text Edit	Reported By	reportedby	30	
One	Text Edit	Category	category	30	
One	Line	Line			
One	Text Edit	Customer ID	customeridentification	42	
One	Check Box	Account is Current	accountstatus		
One	Text Edit	Company	companyname	60	
One	Text Edit	Phone	companyphone	32	
One	Line	Line			
Two	Combo Box	Application	applicationname	four menu values	Brickyard Plus Brickyard Tools DeskMaker DeskBuilder
Two	Text Edit	Release #	applicationrelease	10	
Two	Combo Box	Operating environment	platform	four menu values	HP IBM PC OS/2 Windows Windows
Two	Text Edit	OS#	operatingsystem	10	
Two	Text Edit	Problem Description	problemdescription	100	
Two	Radio Button Set	Severity	problemseverity	six values	1-6

Create a Form Type

The Form Wizard lets you create a form type and define fields for it.

Follow these steps:

1. Click the Forms tab on the Administrator application.

2. Right-click the broker icon, and select New Form Type from the shortcut menu.

The Form Wizard starts and the Create New Formtype dialog appears.

3. Click Next to continue.

The Form Name dialog appears.

4. Complete the dialog fields and then click Next.

Form Name

Defines a name for the new form type. Spaces are allowed.

Important! Record the form name if you anticipate additional editing sessions on the form type.

Table Name

Defines a name for the table that stores the form data. Spaces are not allowed.

Label Layout

(Optional) Specifies the label position for all the fields on your form. Select *one* of the following:

Top of Field-Positions labels at the top of fields.

Left of Field-Positions labels at the left of fields. This is the default.

Wrap Label-Wraps label text to align with fields. This option is enabled only if Left of Field is selected.

Database Format

Specifies a database format for your form.

The Main Page dialog appears. You can add and delete fields, and rearrange their order on the form by selecting one or more fields and using the arrow buttons on the toolbar to move the fields up or down.

5. Click Add (+).

The Add a new form field dialog appears in which you can define the form fields and database columns. It is useful to have your field inventory list handy for reference when defining the fields and columns.

6. Enter the names for fields, database columns, and so on. You can change the default field types by adjusting the settings. For example, to change the default width of a Text Edit field, use the arrows next to the Width field to increase or decrease the width.

Field Type

Specifies the type of field you want to define and add to the form.

Properties

Depending on the field type you select, the Properties box lists appropriate attributes.

Note: Spaces are not allowed in the Database Column field.

7. (Optional for the Radio button set, Combo box, and Tab control field types) Use the buttons in the upper right corner of the Properties box to open a dialog that lets you add and name labels, delete the labels, or resort the order in which labels appear for these field types.

8. Click Finish to save the field definition.

The form type definition is saved.

If you are including tab pages, the Form Wizard requests a list of fields to include on the tab page.

9. (Optional) Click Back and Next to move from one tab page to another and click OK on each dialog to save your changes.

After defining and saving the fields, you can change the field properties directly from the definition window.

The new form type does not exist in the database. The HFD and XML files for your new form type are located in the Software Change Manager/Forms directory. You must decide whether the form is ready to be added to the database or whether the form type needs further editing before continuing.

More information:

[How to Create Form Types](#) (see page 127)

How to Add or Modify Form Fields

You can use various form controls to modify your form type fields or to add fields to new form types. The Add a new form field dialog in the Form Wizard lets you define form fields and database columns. Use your field inventory list for reference when you define the fields and columns.

Follow these steps:

1. Start the Form Wizard and click the plus sign (+) on the main page.

The Add a new form field dialog appears.

2. Select a type from the FieldType drop-down list.

The type of field you want to define and add to the form is specified and the control type dialog for the field type you selected appears.

3. Enter the names for fields, database columns, and so on, in the control type dialog. You can change the default control types by adjusting the settings. For example, to change the default width of a Text Edit field, use the arrows next to the Width field to increase or decrease the width.

Important! A comma in a form field or tab name truncates the name before the comma. For example, Application, Environment shows as Application.

Most control type dialogs use the following fields:

Label

Names a field. The name shows on the form next to the text edit field.

Database Column

Names the database column. This name becomes the name for the attribute stored in the relational database. The database column name is equivalent to the field in the form and locates information in the database to populate the form fields. Spaces are not allowed in the Database Column field.

4. Click OK.

The form fields are added or modified.

Preview and Test a New Form Type

The Preview Form dialog lets you preview and test your new form type. You can update the field definitions on the Form Wizard definition window while the Preview Form dialog remains open.

Important! You can re-enter a Form Wizard editing session only when the database has not created the corresponding form type record. You must know the form type name.

Follow these steps:

1. Click the Forms tab on the Administrator application.
2. Right-click the broker icon, and select New Form Type from the shortcut menu.

The Form Wizard starts and the Create New Formtype dialog appears.

3. Click Next to continue.

The Form Name dialog appears.

4. Enter the form name in the Form Name field.

The Table Name field is automatically populated.

5. Click Next.

The Main Page dialog appears.

6. Click Preview On.
Your new form type appears in the Preview Form dialog.
7. Test the fields by clicking Validation or Default Values.
An error message appears when your test fails.
8. (Optional) Click Print.
The form prints in a list format.
9. Click Close.
The Main Page dialog appears.
10. Click Finish.
The preview and test are complete.

View or Modify Form Type Properties

The Form Type Properties dialog lets you view or redefine the form type properties. Using the Properties dialog, you can do the following:

- Change label names
- Rearrange field order
- Change field height, width, maximum characters, and content
- Change the label format-top or left aligned
- Change the database table format
- Change or create initial values for fields
- Change or create field validations
- Change the Read/Read-Only status

Follow these steps:

1. Click the Forms tab of the Administrator application.
2. Right-click the form type you want to modify, and select Properties from the shortcut menu.

The form Properties dialog appears and you can view the form name and table name.

3. Modify the properties you want to change.

Label Layout

Specifies the label position for all the fields on your form. Select *one* of the following:

- Top of Field—Positions labels at the top of fields.
- Left of Field—Positions labels at the left of fields. This option is the default.
- Wrap Label—Wraps label text to align with fields. Enabled only if Left of Field is selected.

Database Format

Specifies a database format for your form.

Main Page and the subsequent tabs

Lists the fields on each page and let you modify and reposition fields. You can assign default field values, enforce field validation before a user can save a form, and specify the label positions using the Layout Format buttons.

4. (Optional) Select a field name and click Scripts.

The Forms Scripts dialog appears and you can edit and apply modifications to VBScript scripts for default values, initialization, validation, and interaction of form fields.

5. (Optional) Click Preview On.

Your modified form type appears in a separate window. You can update the field definitions on the Form Type Properties window while keeping the Preview Form window open.

6. Click OK.

The form type is modified.

Edit a Form Type Using the Form Wizard

You can edit a custom form type by using the Form Wizard if the following conditions exist:

- The corresponding form type record has not been created in the database
- You know the form type name

Follow these steps:

1. Click the Forms tab of the Administrator application.
2. Right-click the broker icon, and select New Form Type from the shortcut menu.

The Create New CA Harvest SCM Formtype welcome window appears.

3. Click Next.

The Form Name field and Table Name field appear in a window.

4. Enter the name of the form type you want to edit in the Form Name field and tab to the Table Name field.

The Table Name field is automatically populated with the table name.

5. Click Next.

A previous Form Wizard editing session appears.

6. Edit the form type. Click OK.

The form type is edited.

Convert Form Definition Files to XML Files

To use your existing forms in the Workbench and Web Interface, convert the form definition files (HFD) to XML files.

Follow these steps:

1. Navigate to CA_SCM_HOME and execute FormGen.exe.

The CA Harvest SCM Form Converter appears.

2. Complete the fields in the dialog; select XML for the Target Format.
3. Click Convert.

XML files are created and located in the location you specified. Each XML file name is the same as its corresponding form definition file.

More information:

[How to Create Form Types](#) (see page 127)

Store the XML Templates From the Form Reference Folder

Important! Workbench forms *only* require this procedure.

The hformsync command-line utility lets you store the XML templates from the form reference folder into the CA Harvest SCM database.

Other hformsync options let you do the following:

- Compare the last update timestamps of the database and reference versions before updating the database version of the template file.
- Select the form reference directory.
- Process all HFD and XML files in the form reference folder.

Follow these steps:

1. At the command prompt, navigate to the %AllUsersProfile%\Application Data\CA\SCM\Forms directory.

Note: In Windows Vista, Application Data is not visible from Explorer; use the entire path %AllUsersProfile%\Application Data\CA\SCM\Forms to navigate to the Forms folder.

2. Run the hformsync command, specifying either the -all or -hfd option, and any additional option.

For example:

```
hformsync {-b name} {-usr username -pw password} [-d folder] [-hfd] [-all]
```

The XML files specified in the form reference directory are added to the database.

Note: For more information about the hformsync command, including a description of all command options, see the *Command Line Reference Guide*.

How to Modify a Form Type by Editing the XML File

You can modify a form type by editing the form XML file. For example, you can use Workbench form template features such as Multiple Columns, Field Validation Patterns, and Required Fields. You can also introduce your own JavaScript form scripts for advanced customizations.

To change a form type, do the following:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and open the XML file for the form you want to edit in an XML editor program.
2. Edit the XML file and save it.
3. Use the hformsync command-line utility to place the form definitions in the CA Harvest SCM database.

Note: For information about using the hformsync command-line utility, see the *Command Line Reference Guide*.

More information:

[How to Create Form Types](#) (see page 127)

Define a Multi-Column Form Page Layout

Multi-column form pages help you organize your form fields. You can use multiple columns and you can specify that fields span more than one column in the form.

Follow these steps:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and use a text editor to open the XML file for the form you want to edit.

The form definition appears in the text editor.

2. Edit a <tabpanel> or <harvest_form> element to introduce a columns attribute.

Note: Pages which do not define columns default to a single column.

3. (Optional) Define the span attributes to specify the number of columns that a field spans.

Note: Fields that do not define a span default to spanning a single column.


4. Save the file and resynchronize the form changes with your CA Harvest SCM server.

Example: Define Column and Span Attributes

This example shows an XML form template that uses columns and span attributes:

```
<tabpanel id="1" label="Origination" columns="2">
<image ... span="2"/>
<text value ... span="2">
<date-field ... label="Date reported"/>
<dbcombobox ... label="Reported By"/>
<combobox ... label="Category" maxsize="11" rows="1">
...</combobox>
<text-field ... label="SS# (nnn-nn-nnnn)" .../>
<text-field ... label="Component" .../>
<text-field ... label="Department# (nnn)" .../>
<text value= ...>
<image ...>
<hyperlink label="CA Home" ... span="2">
</tabpanel>
```

The fields appear in the Form Editor as shown in the following figure (invisible columns and span are also shown here for illustration):

<div>Unify and Simplify</div> <div>Optimize your IT with Enterprise IT Management (EITM)</div> <div>> Learn more</div> 		span=2
Rich Form Label		span=2
Date reported: <input type="text" value="Mar 30, 2007"/>	Reported By <input type="text"/>	
Category <input type="text" value="Bug Fix"/>	SS# (nnn-nn-nnnn): <input type="text"/>	
Component: * <input type="text" value="abc"/>	Department# (nnn): * <input type="text"/>	
<div>New Features</div> <ul style="list-style-type: none">• Java CMSDK• Workspace• Server-based Form Templates		<div>CA Security Advisor</div> <div>Current Global Condition</div> <div><div></div><div>WARNING</div><div></div></div> <div>> View Latest Threats</div>
CA Home		span=2

Define Required Text Fields

You can require that users complete specific form text fields. If a required field is not complete when a user saves a form, a dialog appears that alerts the user to the missing required fields. Required fields are marked with a trailing asterisk (*) in the Form Editor. Use a required boolean attribute to implement required fields without using scripts. If the required attribute is undefined, the form editor will not require user input for the form fields.

Follow these steps:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and use a text editor to open the XML file for the form in which you want to define required fields.
The form definition appears in the text editor.
2. Edit a <text-field> element and introduce a required attribute set to true, and save the file.
3. Re-synchronize the form changes with your CA Harvest SCM server.

Example: Define Required Text Fields

This example shows a <text-field> definition for the setting of the required attribute:

```
<text-field cols="18" dbfield="mrcomponentname" id="mrcomponentname"
label="Component" maxsize="30" required="true"/>
```

The definition requires the user to complete the Component field of a form. The field is marked with a trailing asterisk (*) in the Form Editor.

Use Case: A User Attempts To Save a Form With Missing Required Fields

This use case illustrates what occurs when a user attempts to save a form without completing required fields:

1. A user attempts to save a form with missing required fields.
A dialog appears and lists the fields that are missing required data.
2. The user clicks OK on the dialog.

The user's focus is returned to the Form Editor and cannot save the form until the required fields are completed.

Define Text Field Pattern Validation

You can use regular expressions in forms to help ensure that data is entered correctly before it is saved to the database. A regular expression is a pattern of characters that describes a set of strings. The pattern attribute represents an industry standard regular expression for the format of the text field.

You can also specify an optional patternmessage attribute. This attribute represents user-friendly text explaining the pattern requirement. This text is displayed to users if a pattern is not satisfied when a form is saved.

The following table shows examples of text field validation patterns:

Pattern	Example Matching Text	Usage Example
\d{3}-\d{2}-\d{4}	123-45-6789	Social security number
\d{4}-\d{4}-\d{4}-\d{4} \d{16}	1234-1234-1234-1234 or 1234123412341234	Credit card number
D\d{3},\d{3} D\d{6} D\d{6}	D339,456 D321987 D000152	US Design Patent Number

Note: For a complete list of regular expression constructs, see <http://java.sun.com> and go to `java.util.regex.Pattern` JavaDoc page.

Follow these steps:

1. Navigate to `%AllUsersProfile%\Application Data\CA\SCM\Forms` and use a text editor to open the XML file for the form in which you want to edit for text field pattern validation.

The form definition appears in the text editor.

2. Edit a `<text-field>` element to set the pattern attribute. For example:

```
<text-field cols="18" dbfield="mrdepartment" id="mrdepartment"
label="Department# (nnn)" maxsize="30" pattern="\d\d\d" />
```

3. (Optional) Specify a `patternmessage` attribute. For example:

```
patternmessage="Three Digits Required"
```

The complete example line is:

```
<text-field cols="18" dbfield="mrdepartment" id="mrdepartment"
label="Department# (nnn)" maxsize="30" pattern="\d\d\d" patternmessage="Three
Digits Required"/>
```

4. Save the file and resynchronize the form changes with your CA Harvest SCM server.

In this example, the specified pattern restricts the edit field to a series of three digits. When a user saves the form if the edit field does not match the pattern, the save is disallowed and the user is notified "Three Digits Required."

Use Case: Save a Form with Invalid Field Data

1. A user attempts to save a form that has one or more invalid fields (the field text does not match the specified validation pattern).
2. A dialog appears that lists the invalid fields.
 - If the `PatternMessage` attribute was specified in the field definition, this text is displayed for each invalid field.
 - If no `PatternMessage` text is specified, the actual `Pattern` attribute value is displayed within this dialog for each invalid field.
3. The user clicks OK on the dialog, is returned to the Form Editor, and the Save action is canceled. If the Save action occurred as a result of closing the editor or closing the Workbench, the close operation is cancelled.

Define an Image Field

Image fields let you specify pictures to display in forms. Supported image formats are JPEG, GIF, PNG, BMP (Windows bitmap), and ICO (Windows icon).

Follow these steps:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and use a text editor to open the XML file for the form you want to edit.

The form definition appears in the text editor.

2. Define the image attributes in the form definition:

url

(Required) Specifies the image file location. You can specify image locations using http:// or file:// protocols.

href

(Optional) Specifies a hyperlink URL. When a user clicks the image, an external browser opens the specified URL.

span

(Optional) Specifies the number of grid columns that the image spans. If not supplied, the span defaults to a single column.

3. Save the file and resynchronize the form changes with your CA Harvest SCM server.

Example: Define an Image Field

This example shows a sample image <element> field definition:

```
<image url="http://www.ca.com/images/front/chips/unify_walkie1.jpg"
href="http://www.ca.com" span="2"/>
```

Define a Hyperlink Field

Hyperlink fields launch an external web browser at the specified hyperlink URL.

Follow these steps:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and use a text editor to open the XML file for the form you want to edit.

The XML definition appears in the text editor.

2. Define the hyperlink attributes in the form definition:

href

(Required) Specifies a hyperlink URL. When a user clicks the hyperlink, an external browser opens the specified URL.

title

(Optional) Specifies text that appears on the hyperlink. If not supplied, the hyperlink text displays the actual URL supplied by the Href attribute.

Save the file and resynchronize the form changes with your CA Harvest SCM server.

Example: Define a Hyperlink Field

This example shows a sample hyperlink field definition:

```
<hyperlink label="CA Home" href="http://www.ca.com"/>
```

Define a Rich Label

You can define existing <text> label fields with HTML-like tags in their value attribute. The Workbench depicts labels by interpreting these tags and depicting them as richly formatted text. The Form Editor uses an SWT FormText widget to render rich labels. This widget supports a limited subset of HTML-like tags. The supported tags are documented at <http://help.eclipse.org> in the Eclipse JavaDoc for the class `org.eclipse.ui.forms.widgets.FormText`.

Follow these steps:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and use a text editor to open the XML file for the form you want to edit.
The form definition appears in the text editor.
2. Define the rich label attributes in the form definition.
3. Save the file and resynchronize the form changes with your CA Harvest SCM server.

Example: Define a Rich Label

This example shows a <text> label definition example that contains HTML-like tags within its value attribute:

```
<text>  
  
<![CDATA[<p></p><p>New CA SCM Features</p><li>Java  
CMSDK</li><li>Workspace</li><li>Server-based Form Templates</li><p></p>]]>
```

This label appears in the Form Editor as follows:

New CA Harvest SCM Features

- Java CMSDK
- Workspace
- Server-based Form Templates

This example shows special characters such as &, <,>, and so on. Define special characters as value attributes of the text tags.

```
<text value="Features &amp; Properties" />
```

This label appears in the Form Editor as follows:

Features & Properties

Enable the Harweb-Based Form Editor

By default, CA Harvest SCM Workbench displays forms using its Eclipse-based Form Editor. You can enable an alternative Harweb-based Form Editor that uses an embedded web browser. You enable the Harweb Form Editor in individual XML form template files by adding a `harweburl` attribute to the `<harvest_form>` element. The `harweburl` attribute refers to the web URL of your Harweb server.

Follow these steps:

1. Navigate to `%AllUsersProfile%\Application Data\CA\SCM\Forms` and use a text editor to open the XML file for the form in which you want to enable the Harweb-based Form Editor.

The form definition appears in the text editor.

2. Define the web URL of your Harweb server within a `<harvest_form>` element.

For example, the following form template identifies a Harweb URL:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE harvest_form SYSTEM "harwebForms.dtd">
<harvest_form dbtable="harchangerequest" id="harchangerequest" name="Change
Request" numtabs="6" harweburl="http://myserver:8080/harweb">
...
</harvest_form>
```

3. Save the file and resynchronize the form changes with your CA Harvest SCM server.

The form is defined to enable the Harweb-based Form Editor.

When a user selects Edit Form on the Workbench, the form displays in an embedded web browser.

Redirect a Form

You can modify a form definition to redirect a form to an external web application or to Harweb.

Follow these steps:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and use a text editor to open the XML file for the form you want to edit.

The form definition appears in the text editor.

2. Add the harweburl attribute.

Note: At runtime, additional parameters such as formobjid, broker name, and username are appended to the URL.

3. Add the isharweb attribute to define the behavior that occurs when a user opens the Workbench form as follows:

`isharweb="false"`

Redirects the request to an external URL. The HTML response from the web application appears in the embedded web browser instead of in the default form editor view.

`isharweb="true"`

Redirects the request to Harweb. If you do not specify the isharweb attribute, by default the request is redirected to Harweb.

4. Save and close the file.

Example: Redirect a Form

The following example shows code that redirects a form to an external URL:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE harvest_form SYSTEM "harwebForms.dtd">
<harvest_form dbtable="sampletaskform" id=" sampletaskform " name="Task Form"
numtabs="3" harweburl=" http://myhost:8080/TaskForm.do" isharweb="false">
</harvest_form>
```

How Forms Use JavaScript Scripts

The form type XML file uses JavaScript. The following scripting elements that you can define as child elements of <harvest_form> represent form type scripts:

1. <initialization> Element—Use to initialize any field.
2. <validation> Element—Use to validate any critical field data.
3. <events> Element—Use to handle user input and respond to these actions. User actions (type a character, select from a combo, and so on) are named events.

XML Form Script Examples

Examples of XML form scripts follow:

```
<harvest_form dbtable="harmr" id="harmr" name="Modification Request" numtabs="6">
```

```
  <initialization language="javascript" client="eclipse">
    editor.setTextFieldValue("mrcomponentname", "JavaScript value");
  </initialization>
```

```
  <validation language="javascript" client="eclipse">
    if (editor.getTextFieldValue("mrcategory").length() == 0)
    {
      editor.alert("You must select a category.");
      editor.setValid(false);
    }
  </validation>
```

```
  <events language="javascript" client="eclipse">
    function formOpened()
    {
      editor.alert("JavaScript formOpened Event");
    }
    function mrapplicationChanged()
    {
```

```
        editor.setTextFieldValue("mrdocument",
editor.getTextFieldValue("mrapplication"));
    }
</events>
...
</harvest_form>
```

Note: For performing form validations in Harweb, specify the client attribute value as “harweb”. An example script follows:

```
<validation language="javascript" client="harweb">
```

```
function validate()
{
    if (trimAll(document.getElementById('usdpackagenameprefix').value) == "")
    {
        alert("Please enter USD Package Name Prefix");
        return false;
    }
}
```

Use Initialization Scripts

Initialization scripts let you perform custom initialization of form field values. Thus, when a user opens a form in the Form Editor, the field value is set to its initial value. A field can have both an initial value and a current value, both of which are character strings. A field's initial value does not change unless the user edits it. Edited form values are not stored in the database until a user saves the form.

Follow these steps:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and use a text editor to open the XML file for the form in which you want to set initial values.

The form definition appears in the text editor.

2. Define the initialization logic in the form definition within an initialization element.

```
<initialization language="javascript" client="eclipse">
    editor.setTextFieldValue("mrcomponentname", "JavaScript value");
</initialization>
```

3. Save the file and resynchronize the form changes with your CA Harvest SCM server.

The initial values for the form field are set.

Example: Define Initialization Logic

An example of an initialization logic definition follows:

```
<initialization language="javascript" client="eclipse">
    editor.setTextFieldValue("pacinitiator", "Hello");
editor.selectTab("Change Evaluation");
editor.setFocus("pactestplan");
</initialization>
```

When a user opens a form in the Form Editor, this block of JavaScript code is invoked and the form shows:

- The pacinitiator field populated with Hello.
- The Change Evaluation form tab is selected.
- The focus at the pactestplan field.

More information:

[How Forms Use JavaScript Scripts](#) (see page 149)

Use Field Validation Scripts

The XML template can include form scripts that validate form data to help ensure that the user completes required fields and to alert the user when they do not. Scripted data validation occurs when a user saves a form in the Form Editor.

Follow these steps:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and use a text editor to open the XML file for the form in which you want to edit for validation.

The form definition appears in the text editor.

2. Define the field validation logic in the form definition within a validation block.

Note: Your JavaScript must report validity by calling the editor.setValid method. Calling this method with false indicates that the form is invalid and thus saving should be disallowed.

```
<validation language="javascript" client="eclipse">
    if (editor.getTextFieldValue("mrcategory").length() == 0)
    {
        editor.alert("You must select a category.");
        editor.setValid(false);
    }
</validation>
```

3. (Optional) Call an alert method to display an alert message in a pop-up dialog on the Workbench.

For example:

```
editor.alert("Form validation failed")
```

4. Save the file and re-synchronize the form changes with your CA Harvest SCM server.
Field validation is set.

More information:

[How Forms Use JavaScript Scripts](#) (see page 149)

Use Field Event Scripts

The XML file script can contain functions to handle user actions and to respond to these actions. JavaScript listener functions for field changes support user actions (type a character, select from a combo, and so on). The Form Editor invokes listener functions whenever a change is detected in the corresponding form field, and these functions respond to changes in field values by updating the values of other fields.

Use the following naming convention to define listener functions for field changed events:

```
function [FIELDNAME]Changed
```

Follow these steps:

1. Navigate to %AllUsersProfile%\Application Data\CA\SCM\Forms and use a text editor to open the XML file for the form in which you want to define functions.

The form definition appears in the text editor.

2. Define the function logic in the form definition within an <events> element.

For example, if a form includes an mrapplication field, you can define a listener function that responds to its changes as follows:

```
<events language="javascript">
  function mrapplicationChanged()
  {
    editor.setTextFieldValue("mrdocument",
editor.getTextFieldValue("mrapplication"));
  }
</events>
```


3. Define the logic for a `formOpened()` event function. This function is invoked whenever a user opens a form. For example:

```
function formOpened()  
{  
    editor.alert("JavaScript formOpened Event");  
}
```

4. Save the file and resynchronize the form changes with your CA Harvest SCM server.
The form field is defined to handle user actions.

More information:

[How Forms Use JavaScript Scripts](#) (see page 149)

Print a Form Type

Form types are listed in the Forms tab of the Administrator application and you can print them.

Follow these steps:

1. Click the Forms tab of the Administrator application.
2. Do *one* of the following:
 - Right-click the form type you want to print, and select Print from the shortcut menu.
 - Select the form type you want to print, and click File, Print from the main menu.
 - Select the form type you want to print, and click the print toolbar icon.

The form type prints in a list format.

Add the Form Table to Your CA Harvest SCM Database

A relational database table stores and defines the data from each form. The table is automatically created when you define the form type in the Form Wizard.

The management of form database tables is transparent to you; the CA Harvest SCM server process handles it. After you define the table, the CA Harvest SCM server process automatically updates and maintains the database tables; no code is needed to create or modify the database tables.

When you click Apply or OK on a form, the relational database table stores the data associated with that form for that form type. The Find Form dialog lets you query this database table and return a list of the forms that match the selected filtering criteria.

Important! You can no longer update a custom form type after you add it to the database.

Follow these steps:

ORACLE

1. From the command prompt, move to the Forms directory in the following locations:

Windows: %AllUsersProfile%\Application Data\CA\SCM\Forms

UNIX: \$CA_SCM_HOME/forms

2. Add the form to the CA Harvest SCM database using SQLPlus:

```
sqlplus <user>/<password> @'sqlfile' <logfile>
```

For example, the database user is "ca" and the password is "cascm":

```
sqlplus ca/cascm @'Change Request.sql' addform.log
```

Only run this command once; if you try to add the form table to the database again after a successful addition, you receive an error.

3. When you open the Administrator application and switch to the Forms tab, the new form type is listed in the form type list.

If the DBMS is not on the same host where the form was created (for example, UNIX server), add the form to the database using an Oracle service:

```
sqlplus <user>/<password>@<service name> @'sqlfile' <logfile>
```

SQL Server

1. From the command prompt, move to the forms directory:
2. Add the form to the CA Harvest SCM database using the osql utility and passing the form file as a parameter. Whenever possible, execute osql with the `-e` option (trusted connection) for the authentication information. If that is not available, you can use the following SQL Server SQL:

```
osql -d <database name> -i sqlfile -U <user name> -P <user password> -e -b -o output.log
```

For example, the database is “scm”, the user is “cascm”, and the password is “cascm”.

```
osql -d scm -i sqlfile -U cascm -P cascm -e -b -o output.log
```

Only run this command once; if you try to add the form table to the database again after a successful addition, you receive an error.

3. When you open the Administrator application and switch to the Forms tab, the new form type is listed in the form type list. If the DBMS is not on the same host where the form was created (for example, UNIX server), copy the SQL file to the DBMS host and add it as in Step 2.

More information:

[How to Create Form Types](#) (see page 127)

Form Type Deletion

You can delete a form type from the database and a custom form type from the Administrator application using the following procedures:

- Delete from the database
- Delete from the Administrator application

Delete a Form Type From the Database

You can delete a form type by deleting the form type record from the form type table and dropping the form type table.

Several tables in the database refer to form types:

- The form table—harComment, harESD, and so on
- harformtemplates—Stores the form templates for the form type
- harformtype—Lists all form types
- harformtypedefs—Lists all fields for all form types (linked by form type objid)
- harformtypeaccess—User access records for each form type
- harcrpkgproc—Stores the definition of a create package process including the option to create an associated form

Important! Use caution when you delete a form type because *all* instances of that form type will be deleted.

Follow these steps:

1. Start your database SQL query utility.
2. Remove the form type from a create package process, if the create package process is defined to create a form automatically. For example:

```
update harcrpkgproc set formtypeobjid =NULL, createassocform ='N' where  
formtypeobjid = (select formtypeobjid from harformtype where formtypename =  
"harComment")
```

3. Delete the form type record from the form type table. For example:

```
delete from harformtype where formtablename='harComment';
```

4. Drop the form type table. For example:

```
drop table harComment;
```

5. Commit the change:

```
commit;
```

The form type and the form type table are deleted from the database.

Delete a Custom Form Type From the Administrator Application

Users with Admin Form Type access can delete custom form types.

Follow these steps:

1. Click the Forms tab of the Administrator application.

2. Right-click the form type you want to delete, and select Delete Form from the shortcut menu.

A warning dialog appears when any instances of the custom form type exist.

3. Click Yes in the confirmation dialog.

The form type is deleted.

Note: The form type files (XML and HFD) are not deleted and remain in the %AllUsersProfile%\Application Data\CA\SCM\Forms directory, allowing you to add the custom form type again if you need it.

Chapter 10: Administering Data

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

CA Harvest SCM includes utilities that assist you in administering your CA Harvest SCM installation.

This section contains the following topics:

[Oracle Database Maintenance](#) (see page 159)

[SQL Server Database Maintenance](#) (see page 161)

[Relational Database Query Utility](#) (see page 162)

[Audit Log](#) (see page 168)

[Database Tables](#) (see page 180)

[How to Copy or Move a Harvest SCM Project from One Database to Other](#) (see page 189)

Oracle Database Maintenance

The following recommendations can help you maintain your Oracle database for best performance.

- Use Oracle's extensive Backup and Restore capabilities to maintain the CA Harvest SCM database. CA Harvest SCM stores repositories in the database (previous releases of CA Harvest SCM stored them on disk); therefore, a database backup includes repositories. Version data is stored as Binary Large Objects (BLOBs) in the Oracle database. For performance reasons, logging is turned off for the BLOB columns. Shut down CA Harvest SCM before performing an online physical backup. Terminating the broker shuts down the server processes.
- If CA Harvest SCM must be available continuously for configuration item updates, consider logging BLOB column changes.

The following SQL commands turn on logging for the BLOB columns:

```
ALTER TABLE HARVERSIONDATA  
MODIFY LOB (VERSIONDATA )  
( NOCACHE LOGGING );  
ALTER TABLE HARVERSIONDELTA  
MODIFY LOB (VERSIONDELTA )  
( NOCACHE LOGGING );
```

Logging VERSIONDATA can downgrade the performance for check-in and load repository items.

Logging VERSIONDELTA can downgrade the performance of the list version process.

If you do not want to log BLOB column changes, use the Oracle Export utility to make a logical backup of the CA Harvest SCM data.

Note: Use this information as a guide. For complete instructions about maintaining your Oracle database, see your Oracle documentation.

BLOB Data Storage

The Oracle default BLOB storage type is 'storage in row'. This type means that the BLOB data is stored in the row if its total size is less than or equal to 4KB. If the size is greater than 4KB, the entire BLOB is stored out of row and control information with a pointer is stored in its place.

HARFORMATTACHMENT, HARVERSIONDATA, and HARVERSIONDELTA each contain large BLOB data types configured for 'storage in row' and, therefore, are created in the HARVESTBLOB tablespace. If you determine that 'storage in row' is not the best option for your unique CA Harvest SCM configuration, you can change the setting to 'disable storage in row', recreate the table in HARVESTMETA tablespace, and configure the BLOB storage to HARVESTBLOB tablespace.

Note: For detailed BLOB storage information, see your Oracle documentation.

Oracle Export

You can use the Oracle Export utility to make a logical backup of the CA Harvest SCM data. This type of backup has the advantage of being recoverable on a different computer. To restore CA Harvest SCM to the time of the export, use the Oracle Import utility.

You do not need to shut down CA Harvest SCM during export. However, execute export at a time of low usage so performance is not affected. CA Harvest SCM must be shut down to restore this type of backup.

The following export parameters let you create an export dump file that contains all CA Harvest SCM data:

USERID

Specifies the name/password of the Oracle user that owns the CA Harvest SCM schema.

FILE

Specifies the name of the file to contain export data.

LOG

Specifies the name of the file for storing export messages.

OWNER

Specifies the name of the Oracle user that owns the CA Harvest SCM schema.

You can check the export log file for errors. If no errors exist, save the exported dump file to tape or some other backup media.

Oracle Import

You can use the Oracle Import utility to restore CA Harvest SCM to the time of the export. If you want to restore the CA Harvest SCM data to the same database using the same Oracle user name, first drop and then recreate the Oracle user. SQL scripts for this purpose are provided in SCM\Database.

Important! Before you drop the user, verify that you have a good export dump file. Shut down CA Harvest SCM before you proceed.

DropUser.sql drops the user. Use it as follows:

```
sqlplus DBA name/DBA password @DropUser.sql name_of_user_to_be_dropped
output_log_file_name
```

creatusr.sql creates the Oracle user for CA Harvest SCM, grants it the necessary rights, and assigns it quota in tablespaces. Execute creatusr.sql as follows:

```
sqlplus DBA name/DBA password @creatusr.sql name_of_user_to_be_created
password_for_the_user HARVESTMETA HARVESTBLOB HARVESTINDEX
temporary_data_tablespace_name rollback_data_tablespace_name output_log_file_name
```

To restore data to the user, import using the following parameters:

```
USERID = name/password of the user
FILE = exported dump file name
LOG = name of file for storing import messages
FULL = Y
```

SQL Server Database Maintenance

The following recommendations can help you maintain your SQL Server database for best performance:

- Verify that SQL Server is taking advantage of available memory. The amount of memory allocated to SQL Server should be the total system memory minus memory required by the operating system and other applications.

- Create a database maintenance plan for your CA Harvest SCM database. For example, you can run integrity checks, update statistics, perform database backups, and rebuild the indexes at prescheduled times. Verify that the backup schedule is set to run regularly, especially after a large amount of data has been added, changed, or deleted.

Note: For instructions to create a database maintenance plan, see your SQL Server documentation.

- Eventually, the MSDB database may fill up to a point where it can slow down the SQL Server Agent. You can remove backup and restore records from the MSDB database by running the `sp_delete_backuphistory` stored procedure. For example:

```
USE msdb
EXEC sp_delete_backuphistory '08/28/05'
```

This example removes records dated earlier than 08/28/05.

- Remember to perform regular disk maintenance so that the physical files on your disks are defragmented. Rebuilding indexes is not the same as defragmenting the drive. Rebuilding indexes occurs in the SQL Server data files only.
- If your sqlserver log gets very large and takes a long time to load, run the command `DBCC ERRORLOG` to truncate the server log.

Note: Use the information in this section as a guide. For complete instructions to maintain your SQL Server database, see your SQL Server documentation.

Relational Database Query Utility

The Relational Database Query Utility (hsql) utility lets you generate reports from a client computer. Output is generated in a form that you can import to spreadsheet or word-processing programs. To generate this type of SQL, you must understand the relationships among the various CA Harvest SCM tables.

When you execute hsql from the command line (not as a user-defined process), it requires a CA Harvest SCM server process to perform the required transaction with the relational database.

If no input and output options are specified, hsql reads input from the standard input and writes output to the standard output. On Windows, input and output files are required. The hsql utility is limited to executing SQL `SELECT` queries, and cannot alter the contents of the CA Harvest SCM database.

To simplify the documenting of SQL input files, hsql supports a comment feature. Any line that begins with a pound sign (#) in column one is treated as a comment and ignored at execution time.

Note: For detailed descriptions of the hsql command options, see the *Command Line Reference Guide*.

Access Issues

The standard reports generated from the CA Harvest SCM interface require read access to an object before you can generate a report for it. The hsql utility, alternatively, has access to all information stored in the CA Harvest SCM relational database tables. Therefore, access for hsql is not controlled at the object level. Set permission to execute this command using operating system access control methods.

If a user group named HSQL (case-independent) does not exist, access operates as described in the previous paragraph. If such a user group exists, only the users in that user group can execute hsql with one exception-the Admin user group always has permission to access and run hsql.

If a user group named HSQL exists and a user who does not belong to that group tries to run hsql, the user receives the following error message:

No HSQL execution access. Only user in "HSQL" user group can execute HSQL.

hsql Standard Output Example

This example shows how to generate reports based on information entered in the standard Problem Report form.

You create a file named probrep.sql in your working directory containing the following SQL statements:

```
SELECT
    formname,datereported,reportedby,category,
    application,hardware,operatingsystem,document,
    revisionnum,problemdescr
FROM
    harform,harproblemreport
WHERE
    harproblemreport.formobjid = harform.formobjid
ORDER BY
    formname
```

Issue the following command:

```
hsql -f probrep.sql -o probrep.out
```

The following text is an example of probrep.out:

```
FORMNAME = STR-9245
DATEREPORTED = 17-SEP-96
REPORTEDBY = john
CATEGORY = software
APPLICATION = Application A
HARDWARE = Sun 4
OPERATINGSYSTEM = SunOS
DOCUMENT =
REVISIONNUM =
PROBLEMDSCR =
```

File update routine generates errors when there is concurrent access to the file. To create the problem, follow these steps:

1. Open the file for edit with vi
2. Run Application A with this file name specified
3. Executable core dumps when it tries to open the file.

Information for each selected form shows in the same format, separated by two blank lines.

Suppose you add the -nh option to the previous command as follows:

```
hsqL -nh -f probrep.sql -o probrep.out
```

The content of probrep.out now is as follows:

```
STR-9245
17-SEP-96
john
software
Application A
Sun 4
SunOS
```

File update routine generates errors when there is concurrent access to the file. To create the problem, follow these steps:

1. Open the file for edit with vi.
2. Run Application A with this file name specified.
3. Executable core dumps when it tries to open the file.

hsql Tab-Delimited Output Example

This hsql example shows how the -t option creates output that is easily transformed into tables or imported into a spreadsheet.

The following command adds the -t option:

```
hsql -t -f probrep.sql -o probrep.out
```

The output of this command looks like the following:

```
NAME          DATEREPORTEDREPORTEDBY  CATEGORY
APPLICATION   HARDWAREOPERATINGSYSTEM
DOCUMENTREVISIONNUM  PROBLEMDDESCR
-----
-----
-----
STR-9245Mon    Aug 22, 1996   john    software
Application    ASun 4   SunOS   File update
routine generates errors when there is concurrent access to the file. To create the
problem follow these steps:<NL>1.<TAB>Open the file for edit with vi<NL>2.<TAB>Run
Application A with this file name specified<NL>3.<TAB>Executable core dumps when it
tries to open the file.
```

Notice how the problem description has been reformatted. Each occurrence of a tab in the problem description has been replaced with the string <TAB>. New lines have been replaced with the string <NL>.

Suppose that you are only interested in three columns-the form name, the date it was reported, and the description. You would modify the input file to look like the following:

```
SELECT
    formname,datereported,problemdescr
FROM
    harform,harproblemreport
WHERE
    harproblemreport.formobjid = harform.formobjid
ORDER BY
    formname
```

Now the previous hsql command would generate this output:

```
FORMNAME      DATEREPORTED  PROBLEMDDESCR
-----
STR-9245Mon Aug 22, 1996File update routine generates errors when there is concurrent
access to the file. To create the problem follow these steps:<NL>1.<TAB>Open the file
for edit with vi<NL>2.<TAB>Run Application A with this file name
specified<NL>3.<TAB>Executable core dumps when it tries to open the file.
```

It is easy to create a table using this output by using the Text to Table command of Microsoft Word for Windows. To reformat the problem description, you can use the replace command to replace every occurrence of <NL> with a paragraph mark and every occurrence of <TAB> with a tab.

hsql Specific Report Example

This hsql example shows how to generate reports on specific objects. For example, you might want to generate a report on all forms associated with packages in each state of the development project.

To generate a report that provides information about Problem Report forms in relation to packages in the various states of a project, you would need to search a number of tables. The following SQL generates such a report for a project named Release 1.1 Project. Notice how comment markers (#) indicate where you might want to make a change.

```
SELECT
    formname, statename, problemdescr
FROM
    harform, harassociatedpackage, harproblemreport, harenvironment,
    harpackage, harstate,
WHERE
    harform.formobjid = harassociatedpackage.formobjid AND
    harproblemreport.formobjid = harform.formobjid AND
    harassociatedpackage.assocpkgid = harpackage.packageobjid AND
    harenvironment.envobjid = harpackage.envobjid AND
    harpackage.stateobjid = harstate.stateobjid AND
# To use this report for another project, you must change
# the environmentname value in the next line
    harenvironment.environmentname = 'Release 1.1 Project'
ORDER BY
    statename, formname
```

The following report shows the formatted output generated from hsql:

Release 1.1

STR #	State	Problem Description
-------	-------	---------------------

STR-9245	Assign STRs	File update routine generates errors when there is concurrent access to the file. To create the problem follow these steps: 1. Open the file for edit with vi. 2. Run Application A with this file name specified. 3. Executable core dumps when it tries to open the file.
STR-9233	Coding	Messages must be enhanced for internationalization.
STR-8765	QA	Internal table limits are too small.

From this report, you can see the report generating capabilities of hsql. With this type of report, you know the status of each problem related to a development effort.

hsql Calculations Example

This example addresses the question that managers often ask, “What is development status?” A simple metric that addresses this question is the number of packages in each state in a development project.

To generate this report, write and save the following SQL in a file named pkgdist.sql. This SQL queries for the number of packages in all states of a project named Rel 1 Project.

```
SELECT
    statename, COUNT(*) AS number_of_packages
FROM
    harpackage, harstate, harenvironment
WHERE
    harstate.stateobjid = harpackage.stateobjid AND
    harpackage.envobjid = harenvironment.envobjid AND
    harenvironment.environmentname = 'Rel 1 Project'
GROUP BY
    statename
```

The following hsql command is executed:

```
hsql -t -f pkgdist.sql -o pkgdist.out
```

The command produces the following tab-delimited output:

STATENAME	NUMBER_OF_PACKAGES
Assign STRs	20
Implementation Design	3
Design Completed	1
Deferred	0
Invalid	5
Development	22
Development Completed	4
Test/QA	10
Release	77

Because the output is tab-delimited, you can easily format it in a table using a word processor.

hsql UDP Creation Example

This example shows how you can add a report to the lifecycle as a UDP. Consider how you use a UDP to add the Package Distribution Report. The following syntax shows the command that generates the output:

```
hsql -t -f pkgdist.sql -o pkgdist.out
```

When a UNIX or Linux client executes a UDP, you can omit the `-f` and `-o` options when you define the UDP. When you do not specify an output file, the output goes to the log.

Copy the SQL statements into the Input edit field of the UDP Process Properties dialog, or specify a full path to the `pkgdist.sql` file so that the file is found regardless of who executes the process.

When `hsql` is executed as a UDP, it does not require the existence of another server process. The CA Harvest SCM client issues the transaction like any other transaction.

Audit Log

The audit log records events to any resources (objects) that are defined in CA Harvest SCM and helps your site meet the Sarbanes–Oxley Act (SOX) compliance standard. The audit log records the following events:

- Alter file type
- Lifecycle change
- Delete package
- Delete version

- Delete snapshot
- User group membership
- User login failure

The following examples show how using the audit log can help you administer your project:

- You can look at user login failures, discover any security problems, and determine which user accounts or workstations must be secured.
- You can learn who is making lifecycle changes and perhaps why.

Note: For information about implementing the audit log, see the *Implementation Guide*.

More information:

[Audit Resource Types](#) (see page 169)

[Audit Events](#) (see page 170)

Audit Resource Types

The following resources (objects) are available for auditing:

- Broker
- Project
- User
- User Group
- User Membership
- State
- Package
- Deleted Version
- Alter File Type
- Baseline View
- Working View
- Snapshot View Deletion
- State Process
- Linked Process

Audit Events

The events (actions) that are available for auditing, with columns that are useful when you query the database, are listed in the following table.

ActionObjId

Identifies the audit action associated with this log entry.

ResourceTypeObjId

Identifies the resource type of the primary resource of this auditable action.

ActionName

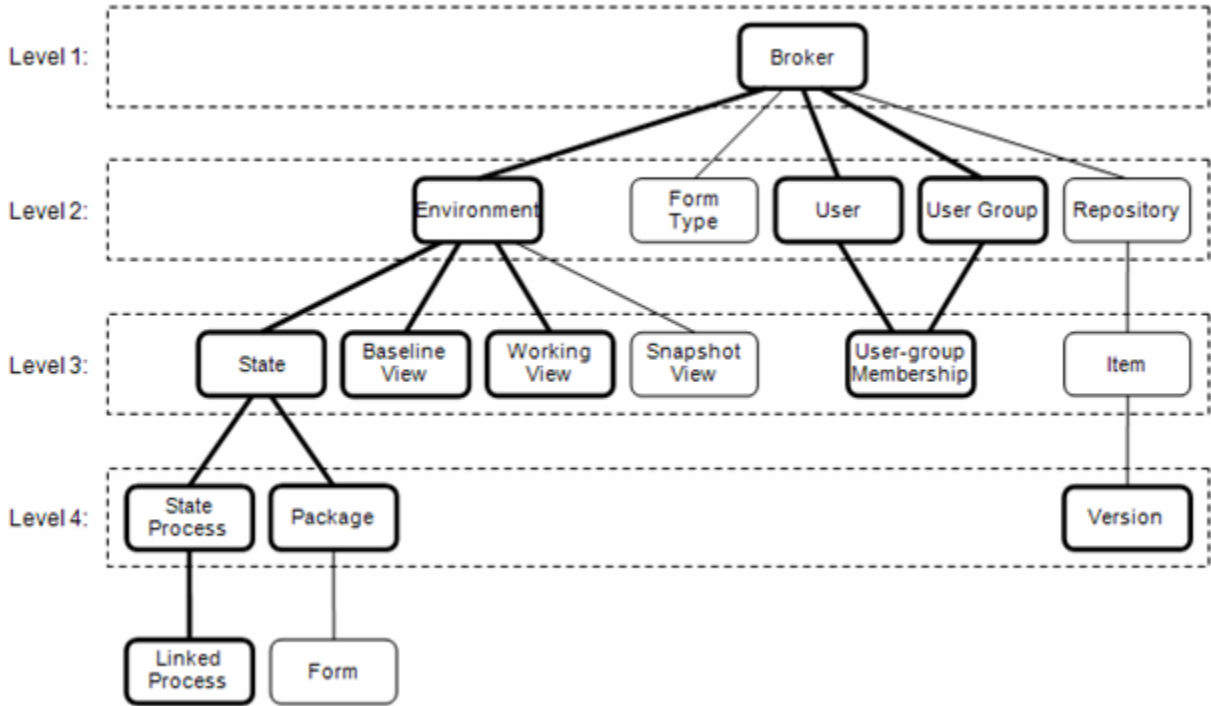
Name of the event (audit action).

ActionObjId	ResourceTypeObjId	ActionName
200	2	Create project
201	2	Update project
204	2	Delete project
205	2	Secure project
400	4	Create user
401	4	Update user
404	4	Delete user
411	4	Login user
500	5	Create user group
501	5	Update user group
504	5	Delete user group
700	7	Create state
701	7	Update state
704	7	Delete state
705	7	Secure state
801	8	Configure baseline view
806	8	Secure baseline view
900	9	Create working view
901	9	Update working view
905	9	Delete working view

ActionObjId	ResourceTypeObjId	ActionName
906	9	Secure working view
1100	11	Create user group membership
1104	11	Delete user group membership
1300	13	Create state process
1301	13	Update state process
1305	13	Delete state process
1306	13	Secure state process
1400	14	Create package
1401	14	Update package
1404	14	Delete package
1504	15	Delete version
1700	17	Create linked process
1701	17	Update linked process
1704	17	Delete linked process

Resource Relationships and Levels

The audit log shows the levels (child, parent, grandparent, and so forth) and the relationship of the resource involved in the audit event. The following diagram shows the resource type relationships and hierarchy. Text boxes in bold outline indicate resource types that can be audited.



HARAUDITLOGVIEW

HARAUDITLOGVIEW is a database view that is provided as the recommended querying and reporting of the audit event log. The following table lists the HARAUDITLOGVIEW columns, and provides the data type, null option, and a brief description for each column.

Column Name	Data Type	Null Option	Information
AUDITEVENTOBJID	NUMBER	NOT NULL	Object ID identifying the audit log entry. Identifies a record in the HARAUDITEVENT tables.
ACTIONOBJID	NUMBER	NOT NULL	Object ID of the audit action associated with this log entry. This column is an index to the HARACTION table.

Column Name	Data Type	Null Option	Information
SUCCESSFAILURE	CHAR(1)	NOT NULL	Indicator of the success or failure of the action that is logged. S=action was successful, F=action failed.
ACTIONNAME	VARCHAR (128)	NOT NULL	Name of the audit action (for example, Update state).
EVENTTIME	DATE	NOT NULL	Date and time of when the auditable action was attempted.
USROBJID	NUMBER	NULL	Object Identifier of the user attempting to perform the auditable action.
USERNAME	VARCHAR (128)	NULL	Name of the user attempting to perform the auditable action.
CLIENTHOSTNAME	VARCHAR (128)	NULL	Name of the CA Harvest SCM client host machine that initiated the transaction that requested the auditable action.
CLIENTPROCESSID	NUMBER	NULL	Process identifier of the process on the client host machine that initiated the transaction that requested the auditable action.
SERVERHOSTNAME	VARCHAR (128)	NULL	Name of the CA Harvest SCM server host machine that processed the transaction requesting the auditable action.
SERVERPROCESSID	NUMBER	NULL	Process identifier associated with the CA Harvest SCM server process that processed the transaction requesting the auditable action.
RESOURCEOBJECTID	NUMBER	NOT NULL	Object identifier of the resource type of the primary resource of this auditable action. This column is a reference to the HARRESOURCETYPE table.
RESOURCETYPENAME	VARCHAR (128)	NOT NULL	Name of the resource type associated with RESOURCEOBJECTID.

Column Name	Data Type	Null Option	Information
RESOURCEOBJID	NUMBER	NULL	Object Identifier of the primary resource affected by the auditable action. Depending on the action, this identifier is a reference to tables: HARENVIRONMENT, HARSTATE, HARSTATEPROCESS, HARLINKEDPROCESS, HARVIEW, HARUSERS. Note: The broker object identifier always has identifier 1.
RESOURCENAME	VARCHAR (1024)	NULL	Name of the primary resource affected by the auditable action.
EVENTDESCRIPTION	VARCHAR (4000)	NULL	This textual description of the audit log event is populated for the create and update event for system objects and for login failures.
ER1RESOURCETYPEOBJID	NUMBER	NOT NULL	Object identifier of the resource type associated with the first level ancestor of the primary resource affected by this auditable action. This column is a reference to the HARRESOURCETYPE table.
ER1RESOURCETYPENAME	VARCHAR (128)	NOT NULL	Name of the resource type associated with ER1RESOURCETYPEOBJID
ER1RESOURCEOBJID	NUMBER	NULL	Object Identifier of the first level ancestor of the primary resource affected by the auditable action. Depending on the action, this identifier is a reference to tables: HARENVIRONMENT, HARSTATE, HARSTATEPROCESS, HARLINKEDPROCESS, HARVIEW, HARUSERS. Note: the broker object identifier always has identifier 1.
ER1RESOURCENAME	VARCHAR (1024)	NULL	Name of the CA Harvest SCM resource of ER1RESOURCEOBJID

Column Name	Data Type	Null Option	Information
ER2RESCOURCETYPEOBJID	NUMBER	NOT NULL	Object identifier of the resource type associated with second level ancestor of the primary resource affected by this auditable action. This column is a reference to the HARRESCOURCETYPE table.
ER2RESCOURCETYPENAME	VARCHAR (128)	NOT NULL	Name of the resource type associated with ER2RESCOURCETYPEOBJID
ER2RESOURCEOBJID	NUMBER	NULL	Object Identifier of the second level ancestor of the primary resource affected by the auditable action. Depending on the action, this identifier is a reference to tables: HARENVIRONMENT, HARSTATE, HARSTATEPROCESS, HARLINKEDPROCESS, HARVIEW, HARUSERS. Note: The broker object identifier always has identifier 1.
ER2RESCOURCENAME	VARCHAR (1024)	NULL	Name of the CA Harvest SCM resource of ER2RESOURCEOBJID.
ER3RESCOURCETYPEOBJID	NUMBER	NOT NULL	Object identifier of the resource type associated with third level ancestor of the primary resource affected by this auditable action. This column is a reference to the HARRESCOURCETYPE table.
ER3RESCOURCETYPENAME	VARCHAR (128)	NOT NULL	Name of the resource type associated with ER3RESCOURCETYPEOBJID.
ER3RESOURCEOBJID	NUMBER	NULL	Object Identifier of the third level ancestor of the primary resource affected by the auditable action. Depending on the action, this identifier is a reference to tables: HARENVIRONMENT, HARSTATE, HARSTATEPROCESS, HARLINKEDPROCESS, HARVIEW, HARUSERS. Note: The broker object identifier always has identifier 1.
ER3RESCOURCENAME	VARCHAR (1024)	NULL	Name of the CA Harvest SCM resource of ER3RESOURCEOBJID.

Column Name	Data Type	Null Option	Information
ER4RESOURCETYPEOBJID	NUMBER	NOT NULL	Object identifier of the resource type associated with fourth level ancestor of the primary resource affected by this auditable action. This column is a reference to the HARRESOURCETYPE table.
ER4RESOURCETYPENAME	VARCHAR (128)	NOT NULL	Name of the resource type associated with ER4RESOURCETYPEOBJID.
ER4RESOURCEOBJID	NUMBER	NULL	Object identifier of the fourth level ancestor of the primary resource affected by the auditable action. Depending on the action, this identifier is a reference to tables: HARENVIROMENT, HARSTATE, HARSTATEPROCESS, HARLINKEDPROCESS, HARVIEW, HARUSERS. Note: The broker object identifier always has identifier 1.
ER4RESOURCENAME	VARCHAR (1024)	NULL	Name of the CA Harvest SCM resource of ER4RESOURCEOBJID.

Audit Log

You can use Oracle or SQL database tools to open the HARAUDITLOGVIEW view and examine the audit log. For example, open the contents of HARAUDITLOGVIEW in Oracle Enterprise Manager to view all the records that are currently in the audit log.

Each audit log record in HARAUDITLOGVIEW contains the following:

- The time and sequence identifier for the audit event
- The identifier and name of the action associated with the audit event
- The success or failure of the action attempted
- The identifier and name of the user associated with the action
- The process identifier and node name associated with the client and server process associated with the action
- The name and identifier associated with the resource of the audit action, and all ancestor resources of this resource
- The resource type name and resource type identifier associated with the resource, and all ancestor resources of this resource

Alternatively, you can use the Structured Query Language (SQL) standard query format to query the HARAUDITLOGVIEW database view and report audit events which have been logged. Queries retrieve information from a database and consist of questions presented to the database in a predefined format.

Important! The use of object IDs instead of object names results in faster execution of queries.

More information:

[HARAUDITLOGVIEW](#) (see page 172)

Audit Example

A typical scenario of a user performing CA Harvest SCM project actions and then of an administrator auditing those actions follows.

Example: Perform User Actions

This example shows the user JohnD performing the following CA Harvest SCM actions, and the records of the actions in the audit log.

1. Log in as user JohnD to Broker1 with an invalid password.
Action 1: The login fails.
2. Log in as user JohnD to Broker1 with a valid password.
The login succeeds.
3. JohnD performs the following actions:
Action 2: Creates project E1
Action 3: Creates state S1 in project E1
Action 4: Creates state process P1 in state S1
Action 5: Creates linked process L1 in state process P1
4. Events are recorded in the audit log as follows:

	Action 1	Action 2	Action 3	Action 4	Action 5
AuditEventObjId	1	2	3	4	5
ActionObjId	411	200	700	1300	1700
SuccessFailure	F	S	S	S	S
ActionName	Login user	Create project	Create state	Create state process	Create linked process

	Action 1	Action 2	Action 3	Action 4	Action 5
EventTime	12/4/2006 11:10am	12/4/2006 11:15am	12/4/2006 11:20am	12/4/2006 11:25am	12/4/2006 11:28am
UsrObjId	10	10	10	10	10
UserName	JohnD	JohnD	JohnD	JohnD	JohnD
ClientHostname	HClient1	HClient1	HClient1	HClient1	HClient1
ClientProcessId	2358	2358	2358	2358	2358
ServerHostname	HBroker1	HBroker1	HBroker1	HBroker1	HBroker1
ServerProcessId	5867	5869	5868	5867	5868
ResourceTypeObjId	4	2	7	13	17
ResourceTypeName	User	Project	State	State Process	Linked Process
ResourceObjId	10	350	567	1200	1201
ResourceName	JohnD	E1	S1	P1	L1
ER1ResourceTypeObjId	1	1	1	1	1
ER1ResourceTypeName	Broker	Broker	Broker	Broker	Broker
ER1ResourceObjId	1	1	1	1	1
ER1ResourceName	HBroker1	HBroker1	HBroker1	HBroker1	HBroker1
ER2ResourceTypeObjId			2	2	2
ER2ResourceTypeName			Project	Project	Project
ER2ResourceObjId			350	350	350
ER2ResourceName			E1	E1	E1
ER3ResourceTypeObjId				7	7

	Action 1	Action 2	Action 3	Action 4	Action 5
ER3ResourceTypeNa me				State	State
ER3ResourceObjId				567	567
				S1	S1
ER3ResourceName					
ER4ResourceTypeObj Id					13
ER4ResourceTypeNa me					State Process
ER4ResourceObjId					1200
ER4ResourceName					P1

Example: Query the Log View

This example shows how the administrator queries the log view using queries.

- Get all failed login events query

```
SELECT *
FROM HARAUDITLOGVIEW
WHERE ActionName = 'Login user' AND SuccessFailure = 'F'
```

Results: Action 1.

- Get all actions performed by user JohnD query

```
SELECT *
FROM HARAUDITLOGVIEW
WHERE UserName = 'JohnD'
```

Results: Actions 1,2,3,4,5.

- Get all actions performed only against the project E1 (nonrecursive) query:

```
SELECT *
FROM HARAUDITLOGVIEW
WHERE ResourceTypeNa = 'Project' AND ResourceName = 'E1'
```

Results: Action 2.

- Get all actions performed against the project E1 and any of its components (recursive) query:

```
SELECT *
FROM HARAUDITLOGVIEW
WHERE
    (ResourceTypeName = 'Project' AND ResourceName = 'E1')
    OR
    (ER2ResourceTypeName = 'Project' AND ER2ResourceName = 'E1')
```

Results: Actions 2,3,4,5.

Note: Projects are a Level 2 resource.

- Get all actions performed against the stateS1 and any of its components (recursive) query:

```
SELECT *
FROM HARAUDITLOGVIEW
WHERE
    (ResourceTypeObjId = 7 AND ResourceObjId = 567)
    OR
    (ER3ResourceTypeObjId = 7 AND ER3ResourceObjId = 567)
```

Results: Actions 3,4,5.

Note: States are a Level 3 resource.

Database Tables

Following are brief descriptions for the CA Harvest SCM database tables. All supported database management systems in CA Harvest SCM use the same schema layout; however, the column types use database-specific mapping.

Note: Detailed descriptions of the database tables are available at Technical Support at <http://ca.com/support>.

HARACTION

Stores the definition of CA Harvest SCM actions that are auditable.

HARALLCHILDRENPATH

Stores all the paths under an item to facilitate recursive searches.

HARALLUSERS

Stores the object user information including active and deleted users.

HARAPPROVE

Stores the definition of an approve process.

HARAPPROVEHIST

Stores the history of an approve process.

HARAPPROVELIST

Stores the definition of an approve process requirement.

HARASSOCPKG

Stores the information for associated packages.

HARAUDITEVENT

Stores the actual auditable actions recorded by the audit system.

HARAUDITEVENTDESCRIPTION

Stores the description field for the auditable action recorded by the audit system.

HARAUDITEVENTRESOURCE1

Stores the resource information for the first level (or top-most) ancestor resource associated with an audit log event.

HARAUDITEVENTRESOURCE2

Stores the resource information for the second level (or second top-most) ancestor resource associated with an audit log event.

HARAUDITEVENTRESOURCE3

Stores the resource information for the third level (or third top-most) ancestor resource associated with an audit log event.

HARAUDITEVENTRESOURCE4

Stores the resource information for the fourth level (or fourth top-most) ancestor resource associated with an audit log event.

HARBRANCH

Stores the information for branches.

HARCHECKINPROC

Stores the definition of a check-in process.

HARCHECKOUTPROC

Stores the definition of a check-out process.

HARCOMMENT

Stores the information for a comment form.

HARCONMRGPROC

Stores the definition of a concurrent merge process.

HARCONVERSIONLOG

Stores conversion information.

HARCRPKGPROC

Stores the definition of a create package process.

HARCRSENVMRGPROC

Stores the definition of a cross-project merge process.

HARDEFECT

Stores an example of a custom form.

HARDELPKGPROC

Stores the definition of a delete package process.

HARDELVERSPROC

Stores the definition of a delete version process.

HARDEMOTEPROC

Stores the definition of a demote process.

HARENvironment

Stores the active project objects. If users delete a project, it will be removed from this table. However, the project object information will be stored in the harAllEnvirs table.

Note: In earlier releases of CA Harvest SCM, a project was referred to as an environment. In the database tables, the word *environment* may remain; it is equivalent to the current name *project*.

HARENvironmentACCESS

Controls CA Harvest SCM project security.

HARESD

Stores an example of a custom form.

HAREXECUTABLEACTION

Stores the identify of the resource type of an auditable action which is the resource or parent resource on which action is executed.

HARFILEEXTENSION

Stores the file extension information.

HARFORM

Stores the definition of an object information form.

HARFORMATTACHMENT

Stores the definition of an attachment data form.

HARFORMHISTORY

Stores the definition of a history form.

HARFORMTEMPLATES

Stores form type templates.

HARFORMTYPE

Stores the name of form types and corresponding table names.

HARFORMTYPEACCESS

Stores the definition of security for a CA Harvest SCM form.

HARFORMTYPEDEFS

Stores the definition of form types.

HARHARVEST

Stores the definition of CA Harvest SCM object attributes.

HARGLOBALAUDITPOLICY

Identifies which valid audit events are allowed to be logged at run-time, if auditing is enabled for the CA Harvest SCM server.

HARINTMRGPROC

Stores the definition of an interactive merge process.

HARITEMACCESS

Stores the definition of security for a CA Harvest SCM item.

HARITEMNAME

Stores the definition of an object of an item.

HARITEMS

Stores the definition of an object of an item.

HARLINKEDPROCESS

Stores the definition of a linked process.

HARLISTDIFFPROC

Stores the definition of a list difference process.

HARLISTVERSPROC

Stores the definition of a list version process.

HARMOVEITEMPROC

Stores the definition of a move item process.

HARMOVEPATHPROC

Stores the definition of a move path process.

HARMOVEPKGPROC

Stores the definition of a move package process.

HARMR

Stores an example of a custom form.

HARNOTIFY

Stores the definition of a notify process.

HARNOTIFYLIST

Stores the definition of notification requirements.

HAROBJECTSEQUENCEID

Contains a list of the tables that use sequences and the corresponding name of the sequence they use.

HARPAC

Stores an example of a custom form.

HARPACKAGE

Stores the definition of a package.

HARPACKAGEGROUP

Stores the definition of the related package group.

HARPACKAGENAMEGEN

Tracks the unique names of generated packages.

HARPACKAGESTATUS

Stores package status attributes used for resetting package status.
HARPACKAGESTATUS stores additional information about packages whose status is not Idle or Archived. This helps the CA Harvest SCM server decide when it is safe to reset the status of a package (to Idle).

HARPASSWORDHISTORY

Stores the previously used passwords for the given 'UserObjId'.

HARPASSWORDPOLICY

Stores the password policy settings for both global and user's overrides.

HARPATHFULLNAME

Stores the full repository path to every version of an item path.

HARPEERREVIEWHISTORY

Stores information used for reporting and for providing history information on a review request. This information is recorded for user consumption. It is not displayed by any CA Harvest SCM component, but can be used to produce reports or provide an audit trail of information concerning peer reviews.

HARPKGHistory

Stores information used for reporting and for providing history information on a package. HARPKGHistory records associated with a particular package are not deleted when the package is deleted from its associated state. This means that the size of this table will always increase and will not decrease when packages are deleted.

HARPKGSINCMW

Stores the definition of the packages in CMW.

HARPKGSINPKGGRP

Stores the relationship between packages and package groups.

HARPROBLEMREPORT

Stores the definition for problem report forms.

HARPROMOTEPROC

Stores the definition of a promote process.

HARQANDA

Stores the information for the Q and A (Questions and Answers) forms.

HARREMITEMPROC

Stores the definition of a remove item process.

HARREMOVEPATHPROC

Stores the definition of a remove path process.

HARRENAMEITEMPROC

Stores the definition of a rename item process.

HARRENAMEPATHPROC

Stores the definition of a rename path process.

HARREPINVIEW

Stores a list of repositories available in each view.

HARREPOSITORY

Stores repository objects.

HARREPOSITORYACCESS

Stores the definition of security for a CA Harvest SCM repository.

HARRESOURCETYPE

Stores the definition of the CA Harvest SCM audit resource types.

HARRESOURCETYPECHILD

Stores the parent-child relationships of CA Harvest SCM audit resource types to one another.

HARRESOURCETYPEDESCENDANT

Stores the ancestral relationships of CA Harvest SCM audit resource types to one another.

HARSEQTABLE

Created and used only in CA Harvest SCM with Microsoft SQL Server implementation. It contains a list of the tables that require sequence numbers, and the corresponding value of the current sequence.

HARSNAPVIEWPROC

Stores the definition of a take snapshot process.

HARSTATE

Stores state information.

HARSTATEACCESS

Stores the definition of security for a CA Harvest SCM state.

HARSTATEPROCESS

Stores the definition of each process state.

HARSTATEPROCESSACCESS

Stores the definition of security for a CA Harvest SCM process.

HARSWITCHPKGPROC

Stores the definition of a switch package process.

HARTABLEINFO

Stores the identification of release versions.

HARTESTINGINFO

Stores the definition of a testing form.

HARUDP

Stores the definition of a User-defined Process.

HARUSDCOMPUTERNAMES

Maintains USD target computer names. It is updated periodically by an HServer based on an HBroker.arg parameter that specifies at what interval the information is to be resynchronized.

HARUSDDEPLOYINFO

Maintains USD software deployment information on a platform by platform basis.

HARUSDGROUPNAMES

Maintains USD target group names. This information is also synchronized on an interval basis.

HARUSDHISTORY

Maintains common USD package maintenance and deployment history.

HARUSDPACKAGEINFO

Maintains common USD package information.

HARUSDPACKAGENAMES

Maintains USD package names. This information is synchronized on an interval basis.

HARUSDPLATFORMINFO

Maintains USD package information on an operating system basis.

HARUSER

Stores the definition of users.

HARUSERCONTACT

Stores the information for user contacts.

HARUSERDATA

Stores the user data for all active users.

HARUSERGROUP

Stores the object user group information.

HARUSERSINGROUP

Stores the relationship between users and user groups.

HARVERSIONDATA

Stores the BLOB data of each version.

HARVERSIONDELTA

Stores the result data of a list differences process.

HARVERSIONINVIEW

Stores all version activities that occur in a view, such as check-in, promote, and demote.

HARVERSIONS

Stores the version object.

HARVERSIONTRACKING

Stores historic values that exist for the version at the time of the switch package operation (that is, at EXECDTIME).

HARVIEW

Stores the view object.

HARRPTSQL

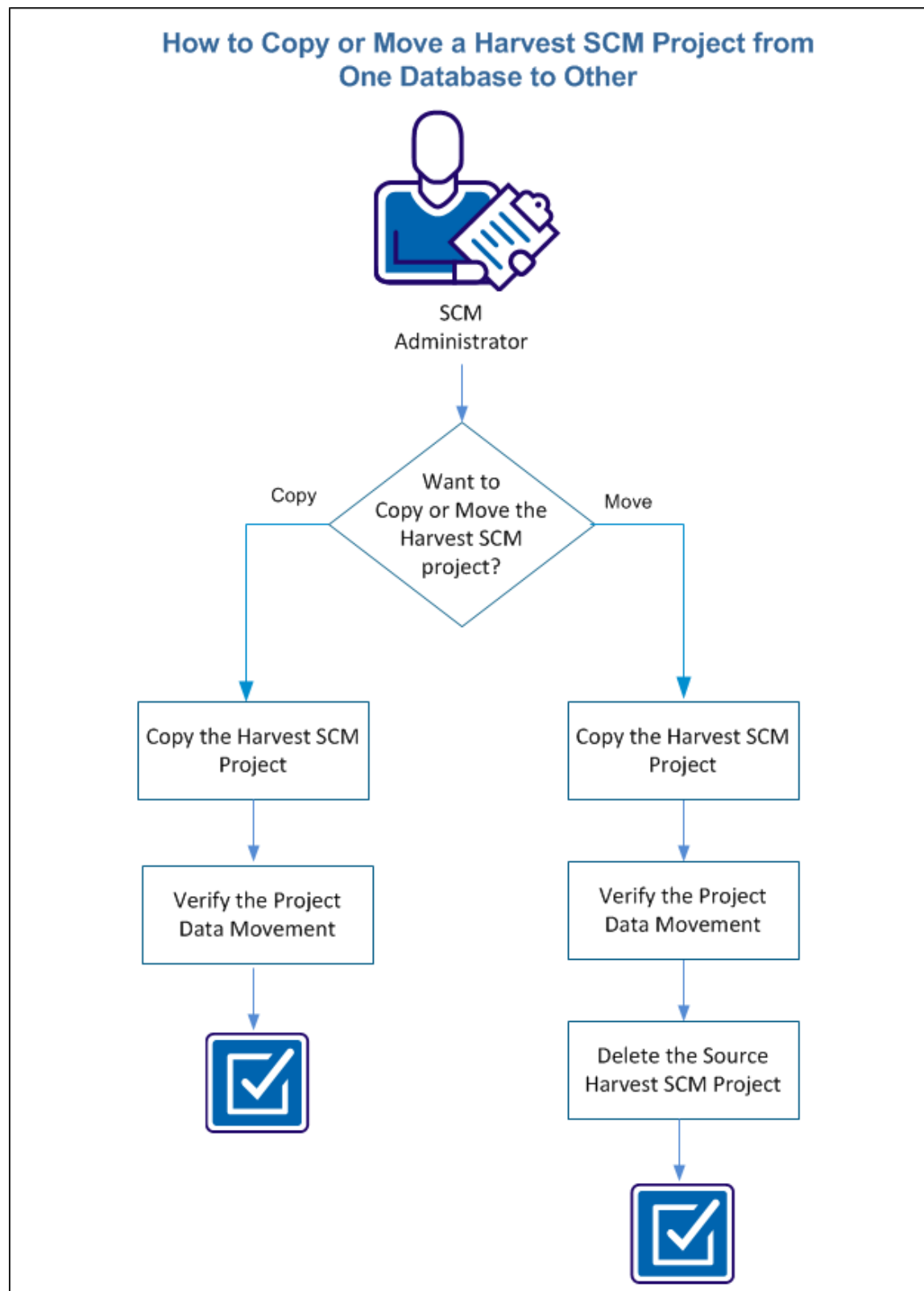
Stores the definition of the custom report queries that are used to generate the SCM Project Dashboard Reports.

How to Copy or Move a Harvest SCM Project from One Database to Other

As an SCM Administrator, your responsibilities include optimum utilization of databases by reducing the inactive database size and reorganize the databases appropriately.

The following diagram illustrates how you can perform the following tasks using the *hmvproj* command-line utility:

- Copy or move a project from one database to other
- Verify the successful data movement of the project
- Delete the source project in case of moving a project



More Information:

[Copy the Harvest SCM Project](#) (see page 191)

[Verify the Project Data Movement](#) (see page 195)

[Delete the Source Harvest SCM Project](#) (see page 197)

Prerequisites

The prerequisites to execute the command-line utility are as follows:

- Access to execute this command line in the operating system.
- Configure the ODBC data source setup for both source project database and target database.
Note: This ODBC configuration could be the same or similar to what is already being used by the SCM server processes.
- Credentials to access both databases with update access.
- Set the project that is involved to the *Inactive* status, to avoid any concurrent update.
- Ensure that the source project has not already been copied to the target database. If the project has been copied to the target database, delete it from the target before copying that project again. This scenario applies even if the project has been renamed in the target database.

Copy the Harvest SCM Project

The *hmvproj* is a command-line utility for the Harvest SCM Administrator usage only. This utility does not require the Harvest SCM broker connection or client components, to execute. With the specified ODBC connections to both source database and target database, the project is copied or moved between databases.

Note: This utility is packaged in the installation of Harvest SCM Server component and not available from client installation.

Important! Considering the possible impact to the existing usage, we recommend that you run this utility during off-peak time or your product maintenance period. Depending on the size of project, the copy or move project may take an extended time and database resources to complete the process of query, insert, and update in database. In addition, the verification has been bundled as a post copy project operation to assure the data integrity.

When move project is executed, the source project gets deleted after successful copy and verification of the project. Alternatively, the copy operation can be used and then specify DELETEONLY to remove it from source database.

Important! The target database has to be at same or newer Harvest SCM database schema release level than the source database. This utility returns an error, if target database's Harvest SCM release level is older than source database.

Follow these steps:

1. Install the command-line utility from the server install package.
2. Execute the *hmvproj.exe* from the command prompt by providing the values for the following options:

-S_ODBC

Specifies the ODBC name for the source project database.

-S_USR

Specifies the source database login username.

-S_PW

Specifies the source database login password.

-T_ODBC

Specifies the ODBC name of the target project database.

-T_USR

Specifies the target database login username.

-T_PW

Specifies the target database login password.

-S_EH

Specifies the source database encrypted login username/password.

-T_EH

Specifies the target database encrypted login username/password.

Note: The encrypted database credentials for both the source and target databases use the convention that the server install is supported.

-PJ

Specifies the SCM project name.

-USER_OPT

Specifies the user/user group handling option flag.

Default valid values: 0, 1, 2

0

Copy all users and user groups.

1

Copy only users and user groups, which has access to that project.

2

Ignore user and usergroup.

Note: If you select this option:

- All the relationship between user/usergroup and other objects in this project gets dropped.

For example, all access settings, the approve process user list, and the notify process user list are dropped.
- If the same name of a user or usergroup exists in the target database, their data does not get updated.

-FORM_OPT

Specifies the form handling option flag.

Default valid values: 0, 1, 2

0

Ignore form if exists in target database.

1

Overwrite form if exists in target database and source database one has newer updated date.

2

Overwrite form if exists in target database.

-DELETE

Specifies the option to delete the project in the source database after completing the copy operation.

-DELETEONLY

Specifies the option to delete the project in the source database directly without copying.

-VERIFYDATA

Specifies the option to verify the version data and the amount of data that must be compared for each version data record.

For example, specifying `-VERIFYDATA=4096` indicates that the version data must be verified and that for each version data record, the first 4096 bytes of data must be compared.

Specifying `-verifydata=0` indicates that the entire version data must be compared for each record.

Note: This option runs as a standalone operation.

-VERIFYROLLBACK

Specifies the option of rolling back the copied project, if verify fails.

-SQL

Specifies the option of SQL getting copied in to the log file for debugging.

3. Provide log file location and any other options that you require.
4. Run the command-line utility.

The project is copied / moved to the target database.

After you copy / move a project, all the objects including the project lifecycle states, processes, packages, and versions are copied in to the target database. Shared objects, such as a user, user group, form, repository are also copied and the relationship between the objects is retained in the target database. The target database project status remains the same as in the source database when you copy the project.

Verify the Project Data Movement

After you complete the data movement of an SCM project, you must verify the completeness of the data movement. When you execute the *hmvproj* utility, the data movement verification operation automatically runs after the successful data movement from one database to other.

The verification operation performs the following tasks:

1. Performs an extensive analysis of the SCM project table data that is copied to ensure proper data movement.
2. Verifies by validating the table record size and ensures the object ID mapping to assure the data integrity.
3. Copies the verification results to the log file that is created during the data copy.

From the log file, analyze the verification results such as the tables that are processed, detailed record counts, and fix any verification errors.

By default, the copied project does not get rolled back (backed out of the target database) if the verification fails. This option lets you investigate and determine if the issues in the verification results are of a major concern.

- Use the *-DELETEONLY* option to clean up the project in the target database if it is determined that the copied project is not acceptable.
- Use the *-VERIFYROLLBACK* option to roll back the copied project immediately if the verification fails.

Copy or Move Project Logs

The `hmvproj` command produces two output log files. One is a full log file containing all of the information that is associated with the copy or move. This log file is named `mvproj_<project name>.log` where `<project name>` is the name of the Harvest project that is being copied. The second log file is a subset of the first. This file contains higher-level information that gives an overview of the copy or move operation. This summary log file is named `mvproj_summary_<project name>.log`.

Both the log files list the following basic information that is associated with the copy or move operation:

- The source and target databases
- The user names
- The parameters that are specified to control the operation such as user handling and form handling

Both logs show table processing during the copy operation and all of the verification information. The difference between the two logs is the detailed information that is provided about the copy operation. The object name and record counts are included in the full log providing a detailed activity of that occurs during the copy.

The verification information is same for both the logs. Each table displays the success and failed counts for all the verified records. Any table with a non-zero failed count results in an overall verification failure. You can investigate this failure to determine the cause of the error and if the error is fatal. However, in any case, consider any verification failure as a fatal error unless, after further investigation, it is considered as acceptable.

In some rare cases, records of a table are intentionally skipped from the copy, because of some of the shared schema objects. The verification detects this and lists the records. However, it does not consider them as errors. In this case, an informational message displays indicating that the reported mismatched records are not to be considered errors.

Troubleshooting the Data Movement

Symptom:

The moved or copied project contain incomplete data in the target database due to execution failure.

Solution:

Use the `-DELETEONLY` option to clean up the incomplete project in the target database and rerun the utility to move / copy the project.

Symptom:

The execution of move or copy project fails right after processing the [harFormTypeDefs] (check the log file here) table with the following error :
“SQL Error: ORA-00955: name is already used by an existing object” or “[SQL Server] There is already an object named ...in the database”

Solution:

This error is caused by the case sensitivity difference of custom form type name between source database and target database. Check table HARFORMTYPE, column FORMTYPENAME, and FORMTABLENAME, and correct the name accordingly. Match the value of both columns (case sensitively) with each other to work properly.

Symptom:

The move or copy of Harvest data with form attachments fails when having same name, but with case insensitive.

Solution:

Before the migration, if there are any form attachments in source and target databases with same name but using different cases, ensure that either of the attachments are renamed to avoid mismatching of form attachments during copy.

Upgrade Support

The move/copy project enhancement supports the following releases to upgrade to the current release. This upgrade support enables you to use this feature as a project migration utility from one release to the latest release.

- Release 12.1
- Release 12.1.01
- Release 12.1.02
- Release 12.1.03

Delete the Source Harvest SCM Project

If you choose to move the Harvest SCM project to a target database, the Delete option deletes the project from the source, after successful copy and verification of the project in the target database.

Chapter 11: Synchronizing Reference Directories

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

This section contains the following topics:

[CA Harvest SCM Reference Directory Synchronization](#) (see page 199)

[Terminology](#) (see page 199)

[hsync Directory Synchronization](#) (see page 200)

[hrefresh Command](#) (see page 209)

[How to Set Up hrefresh](#) (see page 215)

CA Harvest SCM Reference Directory Synchronization

The CA Harvest SCM Reference Directory Synchronization Utility (RDSU) lets you keep directories up-to-date in a reliable and efficient manner. This utility uses the hrefresh and hsync command-line utilities to populate (synchronize) reference directories with appropriate versions of items from project views.

RDSU lets you do the following:

- Minimize disk space requirements for read-only source code directories with the active view approach.
- Manage reference directories on multiple operating systems.
- Allow development teams in different geographical locations to have a synchronized copy of the shared source code managed through the RDSU reference directories. This synchronization is useful for common libraries and reusable code.

Note: For more information about the command-line utility syntax for hsync and hrefresh, see the *Command Line Reference Guide*.

Terminology

The following terms will help you understand RDSU:

active item

An *active item* is a CA Harvest SCM item that has a version in a package located in a state that shares the working view with the specified state.

active reference directory

An *active reference directory* contains the latest version of active items in a particular CA Harvest SCM view.

active view approach

An *active view approach* manages the synchronization of the source versions based on package promotion and demotion. Your inventory is active items only.

browse mode

Browse mode checks out items to the destination directory but does not reserve the items and does not allow you to check the files back in. The read-only attribute is set on files checked out in browse mode.

full reference directory

A *full reference directory* contains the latest version of all items in a particular CA Harvest SCM view.

full view approach

A *full view approach* manages the synchronization of the source versions based on the latest versions of all items in a particular working view, or all versions in a snapshot view.

reference computer

A *reference computer* is usually a remote computer hosting one or more reference directories.

reference definition

A *reference definition* is a record (line) in an hrefresh configuration file that establishes a relationship between a CA Harvest SCM context (project and state) and a reference directory (host and client path).

reference directory

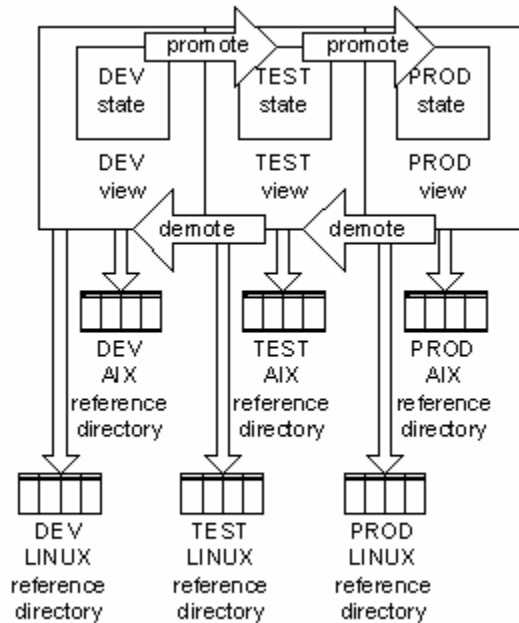
A *reference directory* is a file system directory (UNIX directory, Windows folder, or MVS library). It is populated with files from the CA Harvest SCM configuration management system.

hsync Directory Synchronization

The hsync command updates a file system directory (reference directory) so it reflects the contents of a CA Harvest SCM view path. The hsync command checks out the latest versions of items in the CA Harvest SCM view and deletes files in the reference directory that no longer belong.

CA Harvest SCM supports full view and active view approaches. The inventory of items that is checked out to each reference directory depends which approach you use. The approaches are described in the following sections.

The following scenario illustrates the basic process of refreshing reference directories. The diagram shows a CA Harvest SCM lifecycle with three states and their corresponding working views: DEV, TEST, and PROD. Promote and demote processes move packages from one state to the next state. Reference directories external to CA Harvest SCM keep copies of the files that CA Harvest SCM manages. You can have a single reference directory for each view, or multiple reference directories associated with each view. You can use multiple reference directories on different operating systems that support different build targets as shown in the diagram.



Full View Approach

In a full view approach, the latest trunk versions of all items in the associated view are checked out to the reference directory. The directory inventory can represent an entire repository tree, the recursive contents of a certain repository view path, or multiple repositories. When a process in CA Harvest SCM triggers the check-out of items, only one reference directory per host computer is processed. The hsync command selects the latest versions from the view and copies those versions to the target directory.

Active View Approach

In an active view approach, CA Harvest SCM manages the synchronization of source code versions based on package promotion and demotion. The hsync command locates files across multiple directories, and checks out the latest version of only those items that are associated with active packages based on their state location. Because files move from one reference directory to another, CA Harvest SCM must process multiple reference directories—the directories that correspond to the initial and destination states—when promotes and demotes take place.

hsync Item Selection Options

The hsync command creates an inventory of items and paths based on criteria specified by an item selection option. The item selection options are as follows:

All Items in View

Selects all items and paths under a given view path. You can use this option to initialize or refresh a reference directory at any time. Use this option in a stand-alone UDP or script.

Item List

Selects items and paths with names in the item list. Use this option after a CA Harvest SCM process uses a specific set of items.

Package List

Selects the items and paths in packages with names in the package list. The packages do not need to be located in the state where hsync is executed. Use this option after a CA Harvest SCM process uses specific set of packages.

All Packages in State

Selects items and paths in packages that are located in the state where hsync is executing. Use this option as an efficient alternative to the package list when the set of packages is large and all the packages are in the given state.

All Packages in View

Selects items and paths in packages that are located in any state that shares a view with the packages' current state. This option is an alternative to use when the packages are in states that share a view.

Item selection searches are always recursive and start at a given item path.

Renaming an item or item path creates a version of the same item or item path. The inventory includes the names and paths of all versions of the selected items. Therefore, if an item has been renamed in the current project, both the old and the new item name are in the inventory. If an item has been moved in the current project, both the old and the new path of the item are in the inventory.

To the item list will be added items that are not covered by the item selection option, but have a version with the same name and path as an item that is already in the inventory.

Check-Out Version Selection

From the selected check-out items, hsync uses the No-tag filter to select the latest versions in View and other version selection filters that depend on reference directory type and the branch or trunk options as follows:

- When the reference directory type is Full view, hsync selects the latest versions of all the items.
- When the reference directory type is Active view, hsync selects the latest versions of all the items that are active.
- When the Trunk option is enabled, hsync selects the latest versions on the trunk.
- When the Branch option is enabled, hsync selects the latest versions on the branches based on timestamp.
- When the Trunk and Branch option is enabled, hsync selects the latest version whether it is on a trunk or on a branch based on timestamp.

File Deletion

If the purge option is not enabled, hsync deletes files in the reference directory that meet the following conditions:

- The check-out version list does not have the version with the same name and relative path as the file.
- The file is read-only.

If an item is selected, but does not have a version in the check-out version list and a file exists with the same name in the reference directory and it has read-only permission, the file is deleted. Like check-out, hsync reports an error if the file has write access. For each item path in the inventory, but not in the check-out version list, if the directory has no files other than a signature file, the directory is deleted. After files are deleted from the reference directory, check-out is executed using the previously selected version list.

Files that do not correspond to items in CA Harvest SCM are not touched; therefore, the directory may contain files from some other source.

If a new item is demoted so that no version of the item remains in the current view, the hsync -iv option does not delete the corresponding file in the directory because the item is not in its inventory. For scenarios like this one, use the purge option.

When Subdirectories Are Deleted

After files are deleted, hsync determines the subdirectories to delete. The hsync command deletes subdirectories in the reference directory that meet the following conditions:

- If the purge option is not enabled, the inventory includes a path that has the same name and relative path as the subdirectory.
- The check-out version list does not have the version with the same name and relative path as the subdirectory.
- The subdirectory is empty except for a signature file.

Purge Option and Exclusion Lists

You can use the hsync -iv (All Items in View) option to perform a refresh using a stand-alone UDP. Use the purge option (-purge) with -iv when a possibility exists that all versions of an item have disappeared from view after the last execution of hsync. When you enable the purge option, item inventory is not used to decide if a file or subdirectory should be deleted.

With the purge option, you can use exclusion lists to specify files and directories that should not be deleted. The hsync command provides two ways to specify file names:

- -excl list of file or subdirectory names
- -excls file name pattern

A file is deleted if the following conditions are met:

- The file has not been excluded by either of the exclusion options.
- The check-out version list does not have a version with the same name and relative path as the file.
- The file is read-only.

A subdirectory is deleted if the following conditions are met:

- The subdirectory has not been excluded by the `-excl` option.
- The check-out version list does not have a version with the same name and relative path as the subdirectory.
- The subdirectory is empty except for a signature file.

Note: For details about the syntax for the exclusion list options, see the *Command Line Reference Guide*.

Examples: `-excl` option

To recursively synchronize all files with the `.c` file extension, use the following command:

```
hsync [options] -excl "*.c"
```

To synchronize `*.c` and `*.exe` and recursively synchronize the files `one.c`, `two.c`, and `three.c`, use the following command:

```
hsync file.txt *.c ... -excl one.c two.c three.c [option] *.exe
```

PurgeOpts and PurgeArg Tokens

Use the following format to specify hsync purge options (`-purge`, `-excl`, `excls`) using the `PurgeOpts` or `PurgeArg` tokens:

```
PurgeOpts: {command 1; command 2;...} PurgeArg: {command 1; command 2;...}
```

You specify the `PurgeOpts` and `PurgeArg` tokens in the same way that you specify the `PreCmd` and `PostCmd`, and the order of the `PurgeOpts`, `PurgeArg`, `PreCmd`, or `PostCmd` tokens is not significant; you can specify them in any order at the end of the line. Providing the purge options in a file allows for easier reuse of common exclusion lists, and keeps the hrefresh configuration file less cluttered.

Hsync does not allow the purge options to be used when package list or item list modes (`-pl`, `-ps`, `-pv`, and `-il`) are used. Hrefresh automatically excludes or applies the provided purge options based on the context. A certain record in the hrefresh configuration file can be used in an incremental refresh or a full view (`-iv` mode) refresh. Hrefresh supplies the purge options when they are legal or excludes them when they are not. The decision is logged in the job log file.

You can use the following methods to specify the `PurgeOpts` or `PurgeArg` tokens:

- In the hrefresh configuration file record

For example:

```
Project1,Test,,\,c:\ref\p1\test,FULL,user03s,7001,harvest.dfo,user03.dfo,  
PurgeOpts: {-purge none -excl aaa bbb ccc}, PreCmds: {echoargs}
```

- In an external argument file that an argument file name in the hrefresh configuration file record specifies

For example:

```
Project1,Release,,\,c:\ref\p1\rel ,FULL,user03s,7001,harvest.dfo,user03.dfo,  
PurgeArg: {purge.arg}
```

In this example, purge.arg exists in the hrefresh home directory and contains any of the purge options. This argument file can be multiline, making it easier to read and maintain than traditional harvest argument files. For example, the file can include the following:

```
-purge all  
-excl aaa bbb ccc  
-excls xxx yyy zzz
```

Synchronous/Asynchronous Execution Mode

The option, -execmode, changes the way hrefresh runs with respect to synchronous as opposed to asynchronous behavior.

- In asynchronous mode, hrefresh initiates the refresh jobs and does not wait for any jobs to finish. Hrefresh does not detect nor report the exit status of any refresh job. This behavior is the default.
- In synchronous mode, hrefresh initiates the refresh jobs and waits for all jobs to finish. Upon completion of the longest running job, a report displays the exit codes of all the refresh jobs. Additionally, if any refresh job exits with a failure (nonzero exit code), hrefresh also exits with a failure.

The -execmode option is optional and you can specify it in the following ways:

- In the hrefresh.arg argument file
To set the default execution mode for all hrefresh executions, use -execmode synchronous or -execmode asynchronous as an entry in hrefresh.arg. If hrefresh.arg does not contain this option, asynchronous mode is assumed.
- On the hrefresh command line
To set the default execution mode for a specific hrefresh execution, use -execmode=synchronous or -execmode=asynchronous as a command-line option. The command-line -execmode value has precedence over what is specified in hrefresh.arg.

Use Case-hsync Stand-Alone Execution

This setup example shows how you can use hsync to support incremental nightly builds.

The build setup is as follows:

- All source code is in one reference directory named /Build/Src. The directory contains files that were checked out from CA Harvest SCM.
- The CA Harvest SCM user has an encrypted username/password file (hsync.dfo) that executes hsync.
- The Build state in CA Harvest SCM has a check-out process that allows Synchronize mode and does not preserve the original file time stamp. The hsync user has execute access to the check-out process.

Check-out uses the following options to select versions: Latest in View, No Tag, Trunk Only, and Recursive. These versions are checked out in Synchronize mode.

The nightly build script executes the following command:

```
hsync -b Broker -en Project -st Build -vp \Repository\Src -cp "/Build/Src" -eh  
"/Build/Cred/hsync.dfo"
```

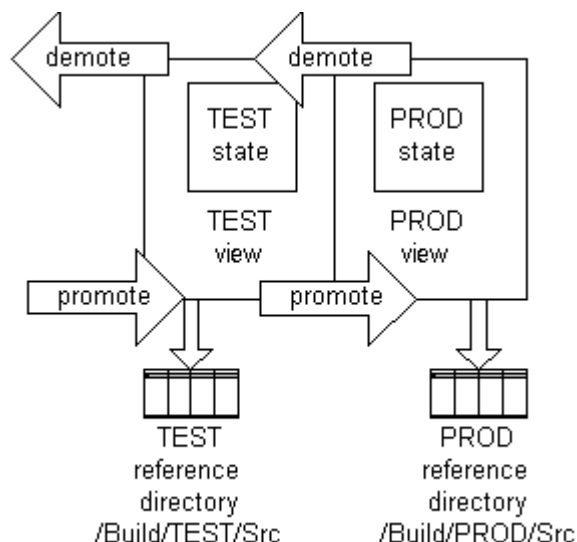
hsync refreshes the /Build/Src reference directory with the latest items in the view path \Repository\Src. If removed items entered the Build state's view that day, the files corresponding to those items are deleted. If a renamed item entered the view, the file with the old item name is deleted, and the latest version of the item with the new name is checked out.

hsync for Promotional File Management Example

This example demonstrates how you can use hsync for promotional file management (active view approach). This example uses the following setup:

- The reference directory /Build/TEST/Src holds the latest version of active items in the TEST state working view. QA evaluates the changes in the TEST state before the changes are promoted for a final build out of the PROD state.
- The PROD state has a working view and follows TEST in the project lifecycle. /Build/PROD/Src holds all the source files for release builds, and contains the latest version of all items in the PROD state working view.
- /Build/TEST/Src and /Build/PROD/Src are in the search path that the TEST build procedure uses. When the build tool does not find a source file in /Build/TEST/Src, it uses the file in /Build/PROD/Src. The reference directories are not located on the same computer as the CA Harvest SCM server installation. Therefore, a remote file agent accesses the reference directories.

The following diagram shows the lifecycle for this example:



Promotional file management occurs as follows:

1. A server UDP is post-linked to the promote process that promotes packages into TEST. This UDP uses the following command:

```

hsync -b [broker] -en "[project]" -st TEST -vp \Repository\Src -cp /Build/TEST/Src
-pl ["package"] -av -rm buildmachine -rport agentport -eh
/CAproduct/Cred/hsync.dfo -er /CAproduct/Cred/bldmach.dfo

```
2. After packages are successfully promoted to TEST, the hsync command executes. hsync examines all items in the list of promoted packages. If there are removed or renamed items in the packages, hsync deletes corresponding files in /Build/TEST/Src. Latest versions of inventory items are checked out in Synchronize mode to /Build/TEST/Src when they are active in the TEST view, on the trunk, and have no tag.
3. The Demote process in the TEST state has a post-linked server UDP with the same hsync command. The package list tells hsync what items must be processed. The demoted packages are not in TEST when hsync executes.
 - If an item is not in any package in TEST, it is no longer active; the corresponding file is deleted from the TEST reference directory.
 - If a demoted package contains a removed item, the item appears in TEST again and is checked out to the reference directory.
 - If a renamed item was demoted, the file with the new name is deleted from the reference directory and the previous version of the item with the old name is checked out.

4. The Promote process in TEST that promotes packages to PROD has two post-linked UDPs. One UDP uses hsync to update the TEST reference directory; the other UDP uses hsync to update the PROD reference directory. After the Promote process executes successfully, items that were active in TEST are now active in PROD. The corresponding files must move from the TEST reference directory to the PROD reference directory.

The hsync command to update the TEST reference directory is the same command as described previously:

```
hsync -b [broker] -en "[project]" -st TEST -vp \Repository\Src -cp /Build/TEST/Src  
-pl ["package"] -av -rm buildmachine -rport 7001 -eh /CAproduct/Cred/hsync.dfo  
-er /CAproduct/Cred/bldmach.dfo
```

5. After all packages are promoted from TEST to PROD, the items in the packages are still in the TEST working view, but they are no longer active. Therefore, files corresponding to those items are deleted from the TEST reference directory.

The following hsync command updates the PROD reference directory:

```
hsync -b [broker] -en "[project]" -st PROD -vp \Repository\Src -cp /Build/PROD/Src  
-pl ["package"] -rm buildmachine -rport 7001 -eh /CAproduct/Cred/hsync.dfo -er  
/CAproduct/Cred/bldmach.dfo
```

Note: You can use the default reference directory type, -fv, instead of -av because the PROD reference directory holds all source files in the PROD view.

hrefresh Command

The hrefresh command works with hsync to synchronize a file system directory with the contents of a path in a CA Harvest SCM view (working or snapshot).

Server UDPs, which may be post-linked to any CA Harvest SCM process that updates the contents of data views such as promote and demote processes, execute the hrefresh command. The post-link command provides hrefresh with the project, state, and package context information. The hrefresh command uses this information to determine which reference directories should be refreshed for that particular transaction. The hrefresh command then uses hsync to synchronize the files on servers that are defined as reference computers.

At execution time, hrefresh reads a configuration file and selects configuration records that apply to its argument list. For each reference computer record that matches the records in the list, hrefresh asynchronously executes a series of commands, including pre-commands and post commands listed in the configuration file. The hsync command executes after the pre-commands and the post commands follows it. The hrefresh command stores the results of all command executions in a log file located in the hrefresh directory.

Note: A stand-alone server UDP can also execute hrefresh, supporting on-demand directory synchronization.

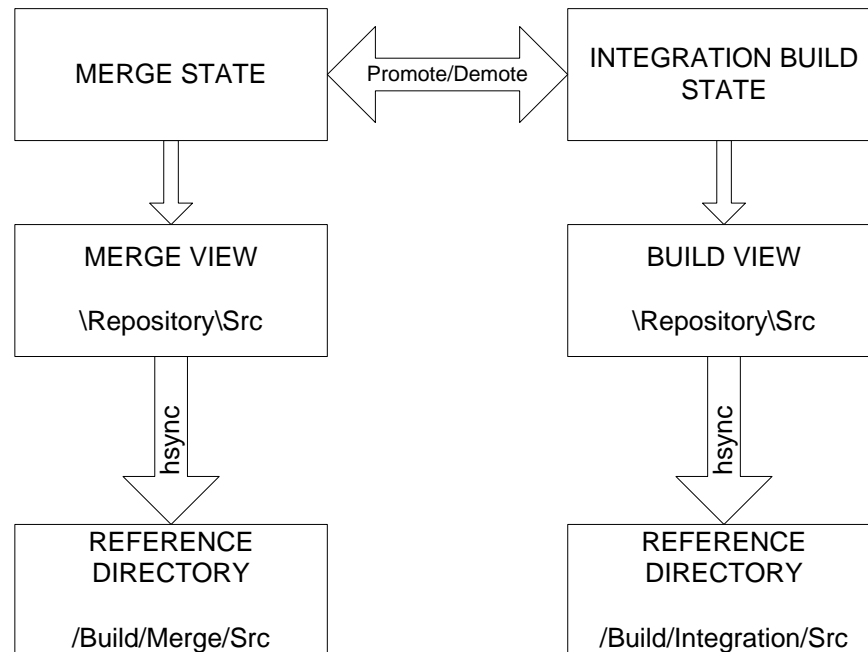
How hrefresh Selects Items

Based on the input that hrefresh receives from the CA Harvest SCM UDP, hrefresh determines which inventory option to use in the following sequence:

1. Item list—The hrefresh command checks the input list for the [version] variable. The inventory is all items in the view of the specified state and specified view path. The inventory is equivalent to using the -il option.
2. Package list—The hrefresh command checks the input list for the [package] variable. The inventory is all items that have versions in the list of packages and the specified view path. The inventory is equivalent to using the -pl option.
3. Full view—The hrefresh command checks the input list for the [version] and [package] variables; if neither is found, the inventory is all items in the view of the specified state and view path. The inventory is equivalent to using the -iv option.

hrefresh Post-Linked to Processes Use Cases

In the following use cases, hrefresh is a post-linked user-defined process (UDP) to processes in the MERGE and INTEGRATION BUILD states of a project. Both states use the default option, -to (trunk only). In the MERGE state, RDSU is configured for an Active reference directory type. In the Integration Build state, RDSU is configured for a Full reference directory type. The following diagram illustrates this scenario:



Use Case: Remove Path

This use case presents synchronization of the items impacted by a remove path process.

1. An hrefresh UDP that is post-linked to a remove path process in the MERGE state executes hsync.

The remove path process supports the package system variable, so the `-pl` option invokes hsync.
2. The package contains all the D-tag versions that the remove path process created. These versions include a version of the removed path, plus a version of every item and path below the removed path. The package can contain other items. Because hsync does not know which items in the package must be synchronized, it synchronizes all.
3. All those items disappear from the state item view, so no version of those items are checked-out.
4. For every removed item below the Src view path, if hsync finds a file under `/Build/Merge/Src` corresponding to a previously checked-out version of the item, hsync deletes the file.
5. If the directories corresponding to removed paths are empty except for signature files, the directories are also deleted.

Use Case: Rename Path

This use case presents synchronization of the items impacted by a rename path process.

1. An hrefresh UDP that is post-linked to a rename path process in the MERGE state executes hsync.

The rename path process supports the package system variable, so hsync is invoked with the `-pl` option.
2. The package contains all the versions that the rename path process created. These versions include a version of the renamed path, plus a version of every item and path below the renamed path.
3. Hsync deletes files under `/Build/Merge/Src` corresponding to previously checked-out older versions of the same items. If the directories corresponding to older path versions are empty except for signature files, the directories are also deleted.
4. If the path that was renamed is below the Src view path, hsync checks out all the new versions to a subdirectory with the new name below `/Build/Merge/Src`.

Note: Refactoring of the top-level view path is not supported. The name of the path is in the hrefresh configuration file. If the path is renamed in CA Harvest SCM, you must change the name manually in the configuration file. The view path name passed to hsync must be consistent with the latest version of a path in view.

Use Case: Move Item

This use case presents synchronization of the items impacted by a move item process.

1. An hrefresh UDP that is post-linked to a move item process in the MERGE state executes hsync.

The move item process supports the package system variable, so the `-pl` option invokes hsync.
2. The package contains the version that the move item process created.
3. If hsync finds a file under `/Build/Merge/Src` corresponding to a previously checked-out older version of the item in its previous path, hsync deletes the file.
4. If the new path of the item is below the Src view path, hsync checks out the new version of the item using the Preserve and Create Directory Structure option to the same relative path below `/Build/Merge/Src`.

Use Case: Move Path

This use case presents synchronization of the items impacted by a move path process.

1. An hrefresh UDP that is post-linked to a move path process in the MERGE state executes hsync.

The move path process supports the package system variable, so the `-pl` option invokes hsync.
2. The package contains all the versions created by the move path process. These versions include a version of the moved path, plus a version of every item and path below the moved path.
3. The hsync command deletes files under `/Build/Merge/Src` that correspond to previously checked-out older versions of the moved items. If the directories that correspond to moved paths are empty except for signature files, the directories are also deleted.
4. If the new paths of the versions are below the Src view path, hsync checks out all the new versions to the same relative paths below `/Build/Merge/Src`.

Use Case: Check In Empty Directory

This use case presents synchronization of the items impacted by a check-in process.

1. An hrefresh UDP that is post-linked to a check-in process in the MERGE state executes hsync.

Check-in supports both the version and package system variables and the pathversion variable.
2. A user checks in an empty directory.
3. If the new path is under Src, hsync creates a subdirectory with the same relative name under `/Build/Merge/Src`.

Use Case: Delete Version of Moved Item

This use case presents synchronization of the items impacted by a delete version process.

1. An hrefresh UDP that is post-linked to a delete version process in the MERGE state executes hsync.
2. A user deletes the version created by the move item process.
3. The delete version process supports the version system variable, but this support is not sufficient when the version represents a moved or renamed item. Hsync cannot find the name and path of the previous version when it is only given the name and path of a version that has already been deleted. Therefore, the delete version process uses the system variable for itemobjid. The post-linked hrefresh UDP uses both the version and itemobjid system variables and these values are passed to hsync.
4. If hsync finds a file under /Build/Merge/Src that corresponds to the deleted version, hsync deletes the file.
5. If the moved item was in a path under Src, hsync checks out the latest version of the item.

Use Case: Delete Version of Refactored Path

This use case presents synchronization of the items impacted by a delete version process.

1. A user deletes a path version corresponding to a removed, renamed, or moved path. All descendant versions are deleted as a consequence.
2. For each deleted version, a version and itemobjid system variable are passed to hsync.
3. If hsync finds files under /Build/Merge/Src corresponding to the deleted versions, hsync deletes the files.
4. If the impacted items are under Src, hsync checks out the latest versions of the items.

Use Case: Merge Refactored Items and Paths

This use case presents synchronization of the items impacted by a merge process.

1. The hsync command deletes files in /Build/Merge/Src corresponding to previous trunk versions of items that were removed, moved, or renamed in the merge. If a removed/moved/renamed path was merged and if the directories are now empty except for signature file, hsync deletes directories corresponding to the older trunk versions.
2. The hsync command checks out the latest normal-tagged trunk version of non-removed items in the inventory list to /Build/Merge/Src. If a merge-tagged version was created, the previous trunk version is checked out (if it was not already checked out). If a delete-tagged version was merged to the trunk, no version of the item is checked out.

Use Case: Promote Packages with Refactored Items and Paths

This use case presents synchronization of the refactored items impacted by a promote process.

1. An hrefresh UDP is post-linked to a promote process in the MERGE state. All packages in the MERGE state are promoted together.
2. The post-linked hrefresh UDP process executes hsync in both the MERGE state and in the INTEGRATION BUILD state using `-pl` with the package system variable.
3. In the Merge state, hsync determines that no check-out is required because the reference directory type is Active and no packages exist in the state.
4. In /Build/Merge/Src, hsync searches for files corresponding to checked-out versions of inventory items (items in promoted packages). Hsync deletes all such files. If the inventory contains paths, and directories corresponding to versions of those paths are now empty except for signature files, the directories are also deleted.
5. In /Build/Integration/Src, hsync deletes files corresponding to inventory items that were removed, moved, or renamed in the promoted packages. If there are directories corresponding to removed/moved/renamed paths and those directories are now empty except for signature files, hsync deletes those directories.
6. In the INTEGRATION BUILD state, hsync checks out inventory items in the BUILD view.

Use Case: Demote Packages with Refactored Items and Paths

This use case continues with the previous use case. Some of the packages that were promoted to INTEGRATION BUILD are now demoted back to MERGE. Those packages contain removed, renamed, and moved items and paths.

1. The demote process in the INTEGRATION BUILD state has a post-linked hrefresh process that executes hsync in both the MERGE state and in the INTEGRATION BUILD state using `-pl` with the package system variable.
2. In `/Build/Integration/Src`, hsync deletes files and directories corresponding to versions of inventory items that are no longer in the BUILD view. This action backs out demoted refactoring changes.
3. In the INTEGRATION BUILD state, hsync checks out the latest versions of inventory items remaining in the BUILD view after demote. This action restores the files and directories in `/Build/Integration/Src` that were deleted in the previous use case due to promoted refactoring changes.
4. Because `/Build/Merge/Src` was previously cleaned out by the promote process, it does not find any files to delete.
5. In the MERGE state, hsync checks out the latest, normal-tagged, trunk version of nonremoved items and paths in the demoted packages.

Server, Client, and Agent Configuration

The CA Harvest SCM server, client, or agent do not require special configurations to use RDSU. You install and configure CA Harvest SCM agents on the reference computers. The agents can be single-user or multi-user, with access to the intended reference directories on each computer.

Note: For more information about configuring CA Harvest SCM communication, see the *Implementation Guide*.

How to Set Up hrefresh

The hrefresh utility setup requires the following administrative tasks:

1. Configure the server, client, and agent.
2. Define the hrefresh Configuration File.
3. Define the hrefresh Argument File.
4. Create the Encrypted Password Files.
5. Define CA Harvest SCM UDPs for hrefresh.

How to Define the hrefresh Configuration File

Create and define a configuration file, by default named HRefresh.cfg. The hrefresh configuration file is an ANSI text file that contains reference definition records. Each reference definition is a comma-delimited set of items on a single line.

The order of items and permissible values for a reference definition are as follows:

1. Project
2. State
3. Check-out process name (The process must have Browse and Synchronize modes enabled.)
4. View path
5. Remote agent host path
6. Reference directory type ["Full | Active"] (case-independent)
7. Remote agent computer name or IP address
8. Remote agent port number
9. File name containing the username/password of CA Harvest SCM user (for -eh)
10. File name containing the username/password of Agent user (for -er)
11. (Optional) Purge options and purge exclusion lists.
12. (Optional) Pre-check-out command definitions
13. (Optional) Post-check-out command definitions

The pre- and post-check-out command definitions let you specify lists of commands to run before and after hsync executes. Hexecp executes the commands using the same authentication (-er file) as hsync. You can configure multiple pre and post commands to run using the syntax PreCmds: {command 1; command 2;...} and PostCmds: {command 1; command 2;...}. hrefresh executes each command asynchronously in the order listed (command 1 executes first, then command 2, and so on).

The pre and post commands have the following format:

PreCmds: {command 1; command 2;...}, PostCmds: {command 1; command 2;...}

The following rules apply to PreCmds and PostCmds:

- Enclose all pre and post commands with parentheses separated by a semicolon “;”.
- Separate pre and post commands with commas “,”.
- Each command can include a program to execute followed by a list of arguments.
- Specify the program to execute as a fully qualified path with no spaces.
- Separate each passed argument with a space.

Example: Format PreCmd and PostCmd

```
CA World 2007, Development , Check Out for Browse , \Web
Application\WebDev\B2CApp,c:\build\dev, Full, hasqlwin2k3, 7001,
myuser.dfo,agentadmin.dfo,PreCmds: {C:\Perl\bin\perl.exe C:\temp\rcommand\rcmd.pl
I am First "Pre Command"; C:\Perl\bin\perl.exe C:\temp\rcommand\rcmd.pl "I am Second"
"Pre Command"}, PostCmds: {C:\Perl\bin\perl.exe C:\temp\rcommand\rcmd.pl "I am
First" "Post Command"; C:\Perl\bin\perl.exe C:\temp\rcommand\rcmd.pl I am Second Post
Command}
```

To define your reference directories, do the following:

1. Create an ANSI text file and specify a name for the hrefresh configuration. The default name is HRefresh.cfg.
2. Enter items and permissible values in the order as described earlier in this section.
 - Blank lines are allowed but hrefresh ignore them.
 - Comment lines are allowed, and are designated by “#” or “//” at the beginning of the line.
3. Save and close the file.
4. Place the file in the hrefresh home directory that is defined in the HRefresh.arg file. If the home directory is not defined, place the file in CA_SCM_HOME.

Following is a sample configuration file:

```
projA,TEST, , \Rep\Src,c:\build\test,Active,winmach,7001,hardest.dfo,winmach.dfo
projA,TEST, , \Rep\Src,/build/test,Active,lnxmach,7001,hardest.dfo,lnxmach.dfo
projA,TEST, , \Rep\Src,/build/test,Active,aixmach,7001,hardest.dfo,aixmach.dfo
projA,TEST, , \Rep\Src,/build/test,Active,hpmach,7001,hardest.dfo,hpmach.dfo
projA,DEV, , \Rep\Src,c:\build\dev,Active,winmach,7001,hardev.dfo,winmach.dfo
projA,DEV, , \Rep\Src,/build/dev,Active,lnxmach,7001,hardev.dfo,lnxmach.dfo
projA,DEV, , \Rep\Src,/build/dev,Active,aixmach,7001,hardev.dfo,aixmach.dfo
projA,DEV, , \Rep\Src,/build/dev,Active,hpmach,7001,hardev.dfo,hpmach.dfo
projA,RELEASE, , \Rep\Src,c:\build\rel,Full,winmach,7001,harrel.dfo,winmach.dfo
projA,RELEASE, , \Rep\Src,/build/rel,Full,lnxmach,7001,harrel.dfo,lnxmach.dfo
projA,RELEASE, , \Rep\Src,/build/rel,Full,aixmach,7001,harrel.dfo,aixmach.dfo
projA,RELEASE, , \Rep\Src,/build/rel,Full,hpmach,7001,harrel.dfo,hpmach.dfo
```

hrefresh Argument File Definition

Note: This task is optional.

The hrefresh argument file defines the following:

- Maximum Retry Value
- Retry Delay Value in seconds
- -nolock option
- hrefresh home directories for each broker name

The home directory locates the hrefresh configuration file and the encrypted password files. In each hrefresh home directory, create a subdirectory named log for the hrefresh log files.

- (Optional) Configuration file name. Default is HRefresh.cfg.
- (Optional) Execute mode; synchronous or asynchronous (default).

Note: The HRefresh.arg file is optional but recommended. If you omit the HRefresh.arg file, the hrefresh home directory is *CA_SCM_HOME* and no retry capability is enabled.

HRefresh.arg supports the following arguments:

-maxretries=*n*

Specifies the maximum number of job retries. If hrefresh determines that an hrefresh job is running for a particular host, hrefresh waits and retries the job up to *n* times. If you do not specify this parameter, the default is 0 (do not retry).

-retrydelay=*n*

Sets the number of seconds to wait between retry attempts, if you configure hrefresh to retry a job when a previous job is still running (maxretries>0). If you do not specify this parameter, the default is 0 (no delay between retries).

-nolock

Overrides semaphore locks and allows multiple check-out operations for the same target directory to run concurrently. If you do not specify -nolock, the semaphore locks are active (the default), and the check-out operations run consecutively. Use -nolock with caution.

Note: When -nolock is in effect, the -maxretries and -retrydelay settings in the HRefresh.arg file are ignored.

-home=directory

Sets a default directory that hrefresh uses as the home directory. If you do not specify this parameter, hrefresh uses CA_SCM_HOME.

-home_name=directory

Sets a default directory that hrefresh uses as the home directory in cases where multiple brokers or hserver version groups exist on a single CA Harvest SCM server installation. If you do not specify this parameter for a broker name or version name, the value specified by *-home=directory* is used.

This option mimics the *-homeserver_version* argument in HBroker.arg, in which you specify where to find the HServer.arg file for each hserver version.

For example, if your HBroker.arg file specifies the following:

```
//Version 1 for Telon Group
-minserver_telon=10
-maxserver_telon=35
-queuesize=5
-homeserver_telon=H:\CA\Software Change Manager\Ver1
```

```
//Version 2 for Telco Group
-minserver_telco=2
-maxserver_telco=35
-queuesize=5
-homeserver_telco=H:\CA\Software Change Manager\Ver2
```

You create separate hrefresh home directories for each group in HRefresh.arg as follows:

```
//Version 1 for Telon Group
-home_telon=H:\CA\Software Change Manager\Ver1

//Version 2 for Telco Group
-home_telco=H:\CA\Software Change Manager\Ver2
```

The *name* portion of the argument is matched at runtime using the following logic:

1. If the broker value is in the form *brokername/version*, *version* locates a designated hrefresh home directory.
2. If the broker value does not have the version name, *brokername* locates a designated hrefresh home directory.
3. If no match exists for either one, the *-home* value is used when it is defined; otherwise, *CA_SCM_HOME* is used.

This approach supports cases where *brokername* is forced using *RT_FORCE_NODE_NAME*, and where multiple brokers may be running.

-cfg=filename

(Optional) Specifies the configuration file name.

Default: HRefresh.cfg

execmode= synchronous|asynchronous

(Optional) Sets the default execution mode for all hrefresh executions, unless overridden by using *-execmode* on the hrefresh command line.

Default: asynchronous

To define the argument file, do the following:

1. Create an ANSI text file and name the file HRefresh.arg.
2. Place the file in the *CA_SCM_HOME* directory.
3. Enter arguments and permissible values in the order as described earlier in this section.

Blank lines are allowed, but hrefresh ignores them.

Comment lines are allowed, and are designated by “#” or “//” at the beginning of the line.

4. Save the file and make sure it remains in the *CA_SCM_HOME* directory.

Semaphore Locks

CA Harvest SCM uses semaphore locks to help prevent failures that may occur when multiple check-out operations for the same target directory are run concurrently. For example, a configuration post-links hrefresh to the check-in process. Each time a user checks in versions, hrefresh automatically synchronizes one or more corresponding reference directories. When another check-in occurs in the same state before the first hrefresh job is finished, a semaphore file created by the first job blocks the second hrefresh job. The second job waits for a specified period (set by the *-retrydelay=n* parameter) for the first job to finish. The second job aborts if the semaphore file is not released within the specified period.

The semaphore-lock files that hrefresh creates are named *project_state_host.lock* and are stored in the directory that you specify as your hrefresh home directory. These lock files are not deleted when jobs complete. An active job obtains an exclusive lock on the semaphore file when it starts, and releases that lock when it finishes.

In some cases, you may want to change this locking behavior; for example, when you want to synchronize multiple separate directories on a single host. The hrefresh configuration file may contain the following entries:

```
#Project , State ,, Viewpath , Clientpath, Type, Host , ...
Project 1, State 1,, \Rep1\path1, /dir1 , Full, host1, ...
Project 1, State 1,, \Rep1\path2, /dir2 , Full, host1, ...
Project 1, State 1,, \Rep1\path3, /dir3 , Full, host1, ...
```

In this example, by default, hrefresh does *not* allow the check-outs for these three directories to execute concurrently. Instead, the processes run consecutively. The job for /dir1 starts, and the other jobs wait until that job is finished.

For some users, the semaphore locks that require processes to run consecutively are more time-consuming than necessary. To override semaphore locks and allow the jobs to run concurrently, specify the *-nolock* option, as follows:

- To set *-nolock* for all hrefresh executions, add a line specifying *-nolock* to your HRefresh.arg file.
- To set *-nolock* for certain instances of hrefresh, specify *-nolock* as a command-line option when hrefresh is invoked. You can specify the *-nolock* option on the program line in hrefresh UDP definitions or custom scripts.

Note: For details about hrefresh command-line options, see the *Command Line Reference Guide*.

Encrypted Password Files

Each reference definition in the configuration file requires two sets of user names and passwords.

- CA Harvest SCM login uses one set for running the check-out using hsync.
- hsync and hexecp use one set for the remote agent connection.

CA Harvest SCM encrypted password files, created by *svrenc -f*, provide the user names and passwords. As an administrator, you run *svrenc -f filename* for each unique username and password. You name the password files and use these names in the configuration file.

Reference definitions can share password files. All hsync operations in a single system can use a single CA Harvest SCM user account. Remote agent login user names and passwords can also be shared.

hrefresh UDP Definitions

The CA Harvest SCM administrator determines how hrefresh is used in each lifecycle. The flexibility of this command enables various levels of hrefresh automation.

In a simple case, you can define stand-alone UDPs for the on-demand execution of hrefresh. Such UDPs may also be used to initially populate reference directories when more automated mechanisms are being used. For stand-alone hrefresh UDPs, the function synchronizes to all related reference directories all items in the context of the view path (as configured in the reference definition).

hrefresh can be invoked automatically when CA Harvest SCM packages are promoted and demoted. This refreshes only those items contained in the list of packages being promoted or demoted.

Whether the UDP is stand-alone or linked, the basic definition is as follows:

Program

```
hrefresh -b "[broker]" -pr "[project]" -st "[state]" -nst "[nextstate]"
```

Input

[package]

[version]

A sample definition of an hrefresh UDP follows:

UDP Properties - hrefresh

User Defined Process | Description | Pre-Linked | Post-Linked | Access | Note

Name: hrefresh

Program: hrefresh -b "[broker]" -pr "[project]" -st "[state]" -nst "[nextst]"

Type: Server

Input: ☒ Secure Input

[package]
[version]

Display

Output: Display Errors: Display

Creation Date: 11/16/2005 2:16:35 PM Creator: harvest

Modified Date: 11/16/2005 2:17:01 PM Modifier: harvest

OK Cancel Apply

More information:

[Define a User-Defined Process](#) (see page 92)

Chapter 12: Repository Cache on Remote Site

This section contains the following topics:

[CA Harvest SCM Repository Cache on Remote Site Overview](#) (see page 225)

[Cache Terminology](#) (see page 226)

[How the Cache Works](#) (see page 226)

[Cache Management](#) (see page 226)

[Cache Maintenance](#) (see page 227)

[Repository Cache on Remote Site Configuration](#) (see page 227)

CA Harvest SCM Repository Cache on Remote Site Overview

Repository Cache on Remote Site copies version data that is stored in the CA Harvest SCM database by the CA Harvest SCM server to locations that check-out operations can access locally. At sites where development teams are located in various geographic areas that have WAN, VPN, or WAN and VPN access to the CA Harvest SCM server, the transmission of repository data can degrade. When large recursive check-outs of data occur, the network speed can have a performance problem because of the amount of data that transfers from the server to the user.

The CA Harvest SCM server sends version data to designated CA Harvest SCM agents, which store it in the local file system so that it is available for LAN-based check-outs. This local access to the data performs better than the WAN-based server check-outs. The cache may not contain all version data files that the check-out needs. The check-out operation assumes this condition; any version data files that are not found in the cache are obtained from the server, but even in this situation the overall performance of the check-out is improved.

Repository Cache on Remote Site is available for check-out for browse and check-out for synchronize only.

Cache Terminology

The following terms help you understand Repository Cache on Remote Site:

- *Cache usage* describes accessing (retrieving) the cache data files during check-out for browse or check-out for synchronize.
- *Cache management* describes the process of updating the cache with new version data files. This process can be performed automatically at the end of a check-out operation or on demand using the hucache command-line program.
- *Cache maintenance* describes the process of removing data files from the cache that are not being used. This process is performed using the hucache command-line program.

How the Cache Works

Cache usage accesses the cache version data files either using direct access to a locally accessible directory or a CA Harvest SCM agent running within the end-user LAN. When the user executes a check-out for browse or check-out for synchronize request and the client is configured to use remote site caching, the cache is used as follows:

1. The check-out client obtains a version list of items to check out from the CA Harvest SCM server.
2. This version list is used to access the repository cache, either using the cache agent or direct access to the cache directory, and copies all version data files already stored in the cache.
3. For any check-out versions whose version data files are missing from the cache, the client executes a check-out to the CA Harvest SCM server.

Cache Management

Cache management stores version data files in the cache. You can update the cache in the following ways:

- Use a check-out process to store version data files from the check-out that were not included in the cache.

The process of updating the cache using check-out occurs at the end of the check-out operation. After the cache process completes, any files not included in the cache are copied from the check-out directory back to the cache. Cache updating occurs only when the path option is set to preserve or preserve and create the output path.

- Use the hucache command-line utility to store latest working view or snapshot view data files in the cache.

Using hucache is a more explicit mechanism for updating the cache. Latest working view or snapshot view version data objects are stored in the cache on demand. hucache can be used on a regular, scheduled basis to preload the cache so that users do not have to update the cache using check-out.

Note: For more information about the hucache command-line utility, see the *Command Line Reference Guide*.

Important! The date of the file stored in the cache is that of the file that was checked out and used to update the cache. Therefore, if you want to preserve the file date when using the cache, enable the Preserve File Time Stamp option on the check-out for browse process in the CA Harvest SCM Administrator.

Cache Maintenance

Cache maintenance purges old cache data files that have not been accessed over a specified period. You use the hucache command-line utility to perform cache maintenance.

Repository Cache on Remote Site Configuration

You can configure the repository cache in the following ways:

- Using the Preferences page of the CA Harvest SCM Workbench.
- Manually updating the HClient.arg parameter file and creating an hcache.dfo file using the svrenc command-line utility.

Configure the Repository Cache Using the Workbench

To configure the repository cache using the Workbench, on the Workbench, navigate to Tools, Preferences, Remote Site Cache, and complete the dialog.

Note: For information about using Remote Site Cache, see the *Workbench User Guide*.

Configure the Repository Cache Manually

To configure the repository cache manually, use the following HClient.arg parameters for the check-out client (Workbench, hco command line):

-cache=*cache directory*

Specifies the local directory that is to contain the cache.

-cache_host=*cache agent hostname*

Specifies the host name of the remote agent to use for accessing the cache.

-cache_port=*cache agent port number*

Specifies the port number of the remote agent to use for accessing the cache.

-update_cache=y|n

Specifies whether a check-out can update the cache. The default is y.

All the HClient.arg parameters are optional. The -cache_host and -cache_port parameters define a cache agent. Only non-rtserver agents are supported. -cache is mutually exclusive with -cache_host and -cache_port. You can specify either a local directory or a remote agent for caching, but not both.

The cache agent login credentials are supplied using an hcache.dfo file stored in the CA Harvest SCM home directory. You can create or update this file by using the svrenc command-line utility.

Chapter 13: Administering Peer Reviews

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

This section contains the following topics:

[Peer Review Verification and Comment Purging – hpeerreviews UDP](#) (see page 229)

[Specify hpeerreviews Parameters](#) (see page 230)

[Sample Promote Processes Using Peer Review UDPs](#) (see page 230)

[Copy and Edit Sample Processes](#) (see page 231)

[Sample UDP Input](#) (see page 231)

[Configure Peer Review Notification](#) (see page 233)

Peer Review Verification and Comment Purging – hpeerreviews UDP

Some sites may want to apply review enforcement, cleanup, or both, associated with processes such as package promotion or approval. To accomplish these process functions, CA Harvest SCM provides the hpeerreviews virtual server UDP program. This program allows for peer review verification plus comment deletion if you want.

Note: The hpeerreviews UDP documentation uses the term *peer review*. Peer review is synonymous with *review request*.

The hpeerreviews virtual UDP provides the following basic actions:

- Verifies that the primary reviewer has closed and approved the review requests associated with the packages.
- Purges comments and attachments from all review requests associated with the packages.

Note: The term “approved” as it applies to review requests means that the primary reviewer has voted Yes. The term does not mean that the approve process has been used to approve the package.

You create a UDP to perform any of these functions by using the Create UDP process. In the UDP, you specify “hpeerreviews” as the program to be run and select Server for the UDP type. You supply various input parameters depending on the type of verification or purge you want.

Specify hpeerreviews Parameters

The hpeerreviews UDP program provides the following basic actions:

- **ACTION=VERIFY**—Verifies review requests (peer reviews) associated with the selected packages. This option controls whether a process, such as Promote Package, is to continue. By specifying hpeerreviews using the ACTION=VERIFY option, you can prevent a promote if the peer review requirements have not been met. ACTION=VERIFY has the following options:

REQUIRED=

Specifies whether reviews are required or not, YES|NO. The default is NO.

SCOPE=

Specifies whether all reviews must be closed or approved or that only one must be closed and approved, ALL|ANY. The default is ANY.

VERSIONCHECK=

Checks the package for new versions that have been checked in since the reviews were closed, YES|NO. The default is NO.

- **ACTION=PURGECOMMENTS**—Purges all comments in all peer reviews associated with the selected packages. This option provides a database cleanup operation when it is not necessary to preserve review comments. Purging comments is a separate operation from verifying the package, that is, you cannot verify and purge with the same UDP.

Follow these steps:

1. For all UDPs, use "hpeerreviews" in the program and designate it as a server UDP.
2. Enter the ACTION=parameter, PROJECT="[project]" PACKAGENAMES="[package]" as Input: parameters. You can enter other parameters to control the way peer reviews are verified.

Note: No other options are associated with ACTION=PURGECOMMENTS.

Sample Promote Processes Using Peer Review UDPs

The Peer Review Sample UDPs lifecycle template provides examples of the various hpeerreviews UDPs specified in sample Promote Package processes. If this template does not exist in your system, you can download it in the following ways:

- Use the -LP option of hdbsetup to load the new project.
- Use himpenv specifying PeerReviewsUDPs.har as the input file parameter. For example:

```
himpenv -b <broker> -usr <user name> -pw <password> -f PeerReviewsUDPs.har
```

After you load the Peer Review Sample UDPs Lifecycle Template, [copy](#) (see page 231) and paste the supplied Promote Package Processes into any Active Project as needed. The template provides nine sample processes; eight processes for each possible verify parameter combination plus a process that purges comments.

Copy and Edit Sample Processes

You copy and edit any of the sample processes to use them in your implementation.

Follow these steps:

1. Right-click the process you want and select copy from the shortcut menu.
2. Navigate to the state of the project that will use this new process, right-click anywhere in the process list, and select paste.

The new process is created.

3. Edit the process to specify the promote to state, and optionally change the process name or any other parameters.
4. Save the process.

Sample UDP Input

Sample UDP input of all of the possible parameter option combinations demonstrates how you use the program for verifying reviews or purging review comments. These parameter samples appear exactly as you must enter them into the Input: field of the Create UDP dialog.

The hpeerreviews.txt file also provides samples. This file is installed in the %CA_SCM_HOME%\Templates\UDP directory on Windows and the \$CA_SCM_HOME/Templates/UDP directory on UNIX/Linux. You can use this file to copy and paste the specific parameters you want for an hpeerreview UDP specification.

Sample Verifications

The following sample verifies that all reviews have been closed and approved, and reviews are required for all packages.

- Without checking for new versions:

```
PROJECT="[project]" PACKAGENAMES="[package]" ACTION=VERIFY REQUIRED=YES  
SCOPE=ALL
```

- With checking for new versions.

```
PROJECT="[project]" PACKAGENAMES="[package]" ACTION=VERIFY REQUIRED=YES  
SCOPE=ALL VERSIONCHECK=YES
```

The following sample verifies that if there are reviews associated with a package all of them have been closed and approved. Reviews are not required. That is, if a package does not have any reviews, it passes verification. If the package does have reviews, all the reviews must be closed and approved.

- Without checking for new versions:

```
PROJECT="[project]" PACKAGENAMES="[package]" ACTION=VERIFY REQUIRED=NO  
SCOPE=ALL
```

- With checking for new versions:

```
PROJECT="[project]" PACKAGENAMES="[package]" ACTION=VERIFY REQUIRED=NO  
SCOPE=ALL VERSIONCHECK=YES
```

The following sample verifies that if there are reviews associated with a package at least one of them has been closed and approved. Reviews are required. That is, a package must have reviews. If the package has more than one review, only one of them must be closed and approved.

- Without checking for new versions:

```
PROJECT="[project]" PACKAGENAMES="[package]" ACTION=VERIFY REQUIRED=YES  
SCOPE=ANY
```

- With checking for new versions:

```
PROJECT="[project]" PACKAGENAMES="[package]" ACTION=VERIFY REQUIRED=YES  
SCOPE=ANY VERSIONCHECK=YES
```

The following sample verifies that if there are reviews associated with a package at least one of them has been closed and approved. Reviews are not required. That is, if a package does not have any reviews, it passes verification. If the package does have reviews, at least one of them must be closed and approved.

- Without checking for new versions:

```
PROJECT="[project]" PACKAGENAMES="[package]" ACTION=VERIFY REQUIRED=NO  
SCOPE=ANY
```

- With checking for new versions:

```
PROJECT="[project]" PACKAGENAMES="[package]" ACTION=VERIFY REQUIRED=NO  
SCOPE=ANY VERSIONCHECK=YES
```


Sample Purge Comments

The following sample purges all comments from all reviews associated with the package[s] that use ACTION=PURGEComments. Purging comments is a separate operation from verifying the package.

Note: If you specify ACTION=PURGEComments, all specified verify parameters are ignored.

```
PROJECT="[project]" PACKAGENAMES="[package]" ACTION=PURGEComments
```

Configure Peer Review Notification

To configure Peer Review notifications, you add the `-premail` parameter to the HServer.arg server parameter file. The `-premail` parameter specifies the hsmtp command line string that issues the Peer Review notifications.

Example: Use the -premail Parameter

In this example, hsmtp uses the email server at mail3.ca.com using port number 25 with sender's email address cascm.user@scmsserver.

```
-premail="hsmtp -m mail3.ca.com -p 25 -f cascm.user@scmsserver"
```

Note: For information about the hsmtp command line program, see the *Command Line Reference Guide*. For more information about the HServer.arg file, see the *Implementation Guide*.

Chapter 14: Configuring SCM for CA Vision Integration

Configuring the SCM CA Vision integration involves:

- Specifying parameters that instruct the broker to start the Synch server at regular intervals.
- Specifying parameters that allow the Synch server to access the CA Vision Web Server.

See the Implementation Guide for details on configuring the CA Harvest SCM Broker and Server for CA Vision Integration.

The Synch Server is a CA Harvest SCM HServer that performs CA Vision synchronizing operations as follows:

1. The Broker starts the Synch Server at the specified interval.
2. The Synch Server obtains new objects from the CA Vision Server and posts any outbound records to the server as required.
3. After the synchronizing operations are complete, the Synch Server process terminates.

Note: The Synch Server is a unique HServer that only performs these CA Vision-related operations. The server does not service regular CA Harvest SCM transactions. Consequently, it does not waste resources for extended periods of time while it is idle.

Important! All the operations documented in this section are only available to CA Harvest SCM Users that are added to a special User Group, CA Vision Admin. Users that are not in this User Group cannot view these menu options in the Workbench.

This section contains the following topics:

[CA Harvest SCM Project and Lifecycle Configuration](#) (see page 236)

CA Harvest SCM Project and Lifecycle Configuration

Some of the CA Vision integration settings are configured using the CA Harvest SCM Workbench. These configuration processes are available only to members of the CA Vision Admin User Group for the SCM Project.

CA Vision integration involves the following operations:

- Configuring the SCM Project Lifecycle - This operation defines some options and identifies the Lifecycle States that are considered Development and QA States and are needed for certain integration operations and displays.
- Mapping Users - This is an optional operation. Use this option if you want to use the SCM Workbench to post task worklog hours to the CA Vision server. The mapping involves associating an SCM User to a CA Vision user. You can do the mapping automatically or manually.

Configuring SCM Project Lifecycle and Options

Configuring the lifecycles of an SCM Project involves following two basic operations:

- Grouping Lifecycle States
- Specifying certain project-level options that are associated with CA Vision integration

Grouping Lifecycle States

A typical SCM Project consists of many lifecycle states. To simplify and report on the package status, and optionally CA Agile Vision™ task status, you can configure the lifecycle states. This configuration helps identify if a package is in the Implementation phase, QA phase, or in the Completed phase. By specifying the following states, SCM provides more reporting and management features of the associated CA Vision objects and their SCM package relationships:

- Development
- Development Completion
- QA
- QA Completion

For example, a CA Vision Task can be automatically marked Completed when a package is promoted to the designated Development Completion state.

You can specify the following options for a given SCM Project:

Automatically update task status

The Administrator specifies whether to update CA Agile Vision™ tasks to a Completed status automatically when a package is promoted to the Development Completion state.

Store changed version information in associated user stories

Code changes (latest trunk versions in a package) are stored in the User Story when a package is promoted to the Development Completion state. Use this option if you do not want to store this data in the CA Vision server due to the volume of records involved. This information is displayed in the CA Vision client in the Code Change segment of the User Story.

Enforce package association

If enabled, this option adds a verification to the Promote Package process. When this option is set, verify that you associate every promoted package with one or more CA Vision objects. This enforcement is restricted to the packages promotion within the Development state to the Development completion state defined within the project life cycle.

Mapping CA Vision Users to SCM Users

For certain CA Vision integration operations, such as Posting Worklog Hours, map the SCM user ID to its corresponding CA Vision user ID. To perform this mapping, use the Configure CA Vision dialog.

Note: Members of the CA Vision Admin group can access this dialog from the Broker node in the CA Vision View.

Using this dialog, you can perform mapping in the following ways:

- Automatic Mapping - Matches SCM users to CA Vision users. Using this option, you can match a user by using the name or the email address. Matches are case-sensitive.
- Manual mapping - Matches an SCM user with a CA Vision user.

You can use these mapping operations any time. These operations are accumulative. That is, running automatic mapping attempts to match only the users that are not already mapped. The list boxes in the Manual section of the dialog show only the users that are not already mapped.

Using the Map Users Dialog

- Automatic Mapping - The Auto section of the dialog contains three radio buttons:
 - Auto Map by Name
 - Auto Map by Email Address
 - None

To perform automatic mapping, select either Auto Map by Name or Auto Map by Email Address. The Existing mapped user list is disabled, indicating that you cannot manually update that list. Click OK. The transaction runs and populates the mapped user list with the list of SCM users associated with CA Vision users. If no further changes are required, click Cancel to terminate the dialog.

- Manual Mapping - The Manual section of the dialog contains two list boxes, CA Vision User and SCM User. Each list box lists only users that are not currently mapped. To add a new user association, select a user from each list box and click the Add to map button.

To remove a mapped user, select the entry from the mapped user list and click the - icon. After you complete all manual changes, click the OK button to apply them. The mapped user list is updated to contain all mapped users. If no further changes are required, click Cancel to terminate the dialog.

Associating CA Vision Product Releases with CA Harvest SCM Projects

The main association required for the CA Vision integration is that of CA Vision Releases with SCM Projects. The best practice is to associate one CA Vision Release with one SCM Project. That is, for every new release, create an SCM Project and create the association. But SCM does not enforce any rules for this practice. A CA Vision Release can be associated with one or more SCM Projects. Similarly, an SCM Project can be associated with one or more releases.

To associate releases with SCM Projects, members of the CA Vision Admin group can access the Associate CA Vision Product Release dialog from the Broker node in the CA Vision View.

To associate a release with an SCM Project, select the project from SCM Project list box. If any releases are currently associated with the project, they are displayed in the list box. To add a release to the project, select the product from the CA Vision Product list box. Select the Release from the CA Vision Release list box and click Add to add it to the list.

To remove a release association, select the release in the list box and click Remove. After you complete all changes for the required SCM Project, click OK to apply the changes to the database.

Note: If a package in the SCM Project is associated with a CA Vision object in that release, the remove release association operation is not allowed.

Synch Server

Special Synch Server Logs

The Synch server does not execute the normal HServer transactions. To facilitate problem-solving, the Synch server creates and appends to a unique server log file. The name of the log file generated is `yyyymmddHServerSynch.log`, where `yyyymmdd` is the date the log was created. The Synch server appends to the log file, if it exists, so that a complete 24-hour period is written to a single log file.

The Synch Server also produces a more user friendly `yyyymmddCAVisionSynch.log` file. This log contains information on the CA Vision update and synchronization operations performed for that particular 24-hour period.

Activating the Synch Server on Demand

For Workbench users who are members of the CA Vision Admin user group, the Synch server can be driven on demand. To activate the Synch server immediately, right-click the broker node in the CA Vision View. Select Synchronize with CA Vision. A dialog is displayed to confirm your request.

Important! Do not use this option frequently. Salesforce.com limits the requests that can be made to the CA Vision instance of the server. This limit is applied on a 24-hour basis. Excessive Synch server runs can exceed this limit.

Appendix A: Predefined Lifecycle Templates

Note: The instances of Linux in this section refer to both the Linux and zLinux operating environments.

This section contains the following topics:

- [Predefined Lifecycle Templates](#) (see page 241)
- [Software Release Development](#) (see page 242)
- [Deploy Release](#) (see page 242)
- [Formal Problem Tracking](#) (see page 266)
- [New Development](#) (see page 268)
- [OnDemand Project](#) (see page 270)
- [Packaged Application Change Management](#) (see page 274)
- [Parallel Development](#) (see page 275)
- [Production](#) (see page 277)
- [Release](#) (see page 291)
- [Release with Emergency](#) (see page 295)
- [Standard Problem Tracking](#) (see page 298)
- [Third Party Tool](#) (see page 299)
- [Version Control](#) (see page 300)
- [Peer Review Sample UDPs](#) (see page 301)

Predefined Lifecycle Templates

CA Harvest SCM provides ready-made lifecycle templates that you can use when setting up your configuration management system. These lifecycles provide SCM solutions for most software development scenarios and let an organization immediately begin using a configuration management process that meets its needs. The templates can be used as is or customized to fulfill specific needs.

Note: For details about each lifecycle template, use the Project Lifecycle Viewer. The Project Lifecycle Viewer gives you a graphical view of the project lifecycles.

Software Release Development

Software release development creates a product that is known by a single identifier such as a release number, a project name, and so on. After this product is produced, work begins on the next version of the product. This new version is a new and separate product. In this type of development project, you must be able to track each version of the product separately so you can determine the differences between versions. You also need to maintain any version of the software while working on the next version. The following lifecycle templates help to accomplish that type of setup:

- Release
- Release with Emergency
- Parallel Development

More information:

[Release](#) (see page 291)

[Release with Emergency](#) (see page 295)

[Parallel Development](#) (see page 275)

Deploy Release

CA Software Delivery (previously named Unicenter Software Delivery) lets you deploy software from CA Harvest SCM. This support extends the lifecycle support provided by CA Harvest SCM by including the final phase of software development: delivering the updated software. This delivery is not limited to the end user, but could also include intermediate target groups such as a Quality Assurance (QA) team. By creating and moving special CA Harvest SCM packages through the CA Harvest SCM lifecycle, you control CA Software Delivery packages that deploy software.

In CA Harvest SCM, you use special lifecycle processes to maintain the CA Software Delivery package. You decide whether to actually deploy the software yourself or to perform all necessary steps leading up to deployment, leaving this final step to the CA Software Delivery administrator.

This process requires the skills of a CA Harvest SCM administrator and CA Software Delivery administrator and includes three major steps that you can perform at different points in the project lifecycle:

1. Stage the installation files.
2. Create the CA Software Delivery package.
3. Deploy the software.

Note: For details about using CA Software Delivery, see your CA Software Delivery administrator or your CA Software Delivery documentation.

How CA Software Delivery Integration Works

Using CA Harvest SCM and CA Software Delivery together to deploy software requires the following major steps:

1. Verify that you have configured CA Software Delivery Integration.
2. Decide whether to deploy the software *explicitly* or *implicitly*. Deploying the software is the final task in the process of using the CA Software Delivery Integration, but you need to understand the methods and select one *before* you create the Software Deployment package, because certain steps *before* deployment vary according to whether you are deploying explicitly or implicitly.

- In *explicit* deployment, you run the following processes separately on your Software Deployment package, in the order shown: stage deployment package, create deployment package, and deploy. These processes stage, create, and execute the CA Software Delivery deployment package.

Use explicit deployment if you want to be able to run the three processes separately at different times or in different states in the lifecycle. Explicit deployment gives you greater control and flexibility in managing the software deployment process than implicit deployment does.

- In *implicit* deployment, you run *one* process, named promote to release and deploy, on your Software Deployment package. Running this process executes the three explicit processes (stage deployment package, create deployment package, and deploy software) in the proper sequence.

Use implicit deployment if you want to be able to run the three explicit processes with a single action. Implicit deployment gives you greater speed and uniformity in managing the software deployment process than explicit deployment does.

3. Using your “typical” lifecycle or another lifecycle of your choice, complete the source change packages for the software you want to distribute. The source change packages include all of the following:

- The source code updates.
- The installation code.
- The files that comprise the software to be deployed. You specify the names of these files in the Install File, Payload, and optional Uninstall File parameters of the Platform Information form or configuration file.

Note: Typically, users require *several* source change packages. For efficiency, you may want to include all packages in a single package *group*.

4. Promote all source change packages from the Development state to the Test state.
5. Promote the packages or package group with the source code updates, as follows:
 - If you are using *explicit* deployment, promote the source change packages from the Test state to the Release state.
 - If you are using *implicit* deployment, promote all source change packages from the Test state to the Software Delivery state.
6. To create the Software Deployment package, run the create software deployment package process on all source change packages.

Note: The Software Deployment package does *not* manage source code updates. The Software Deployment package manages only the *deployment* of the source change packages.

When you create the Software Deployment package, the USD Package Information form for the package is created automatically.

Important! Deployment of multiple software applications to the same platform using the same SCM package is not supported. Only one software installation can be deployed to a particular platform using an SCM deployment package, that is, you cannot have more than one USD platform information form specifying the same target operating system in the same SCM deployment package.

7. Complete the USD Package Information form.
8. For each platform on which you plan to deploy this package, specify the installation parameters for the software being deployed. The installation parameters define the platforms, installation files, and so on needed to deploy the software in CA Software Delivery. Specify these parameters using *either* a complete USD Platform Information form *or* a complete configuration file attached to the USD Package Information form. For instructions to use a form or configuration file, see How USD Platform Information Parameters Are Specified.

If you are deploying on multiple platforms, specify a *separate* form or configuration file for *each* platform. Occasionally, different *levels* of the same platform require different platform information forms or configuration files. For example, you may need complete one platform information form or configuration file for Windows 2000 and another platform information form or configuration file for Windows 2003 Server. However, you may be able to use the *same* platform information form or configuration file for Windows 2000 and Windows 2003 Server.

For more details, see your CA Software Delivery administrator or your CA Software Delivery documentation.

Note: If applicable, for repeated deployments of the same software, you can save USD Platform Information forms and configuration files and add them to new Software Deployment packages.

9. Stage the installation *files* on the CA Software Delivery server by running the stage process on your Software Deployment package. The stage process copies the installation files from the CA Harvest SCM database to the CA Software Delivery server.

The stage process uses the hsync command line utility to copy the files.

Note: For details about running the command, see the *Command Line Reference Guide*.

Important! If you update a Platform Information form or configuration file *after* staging the files, you must stage the installation files again; if you do not, your form or configuration file updates are not processed when you create and deploy the CA Software Delivery package. Instead, these updates are processed the next time you stage the installation files.

Create the CA Software Delivery package by running the create process on your Software Deployment package. Running this process creates *one* CA Software Delivery *deployment* package on the CA Software Delivery server for *each* USD Platform Information form or configuration file in your Software Deployment package.

For example, if your Software Deployment package includes two platform information configuration files (one for Windows and one for Linux), running this process creates two CA Software Delivery deployment packages (one for Windows and one for Linux).

10. Deploy the software explicitly or implicitly, as follows:

If you are using *explicit* deployment, run following processes separately on your Software Deployment package, in the order shown:

- a. stage deployment package
- b. create deployment package
- c. deploy

These processes stage, create, and execute the CA Software Delivery deployment package on the CA Software Delivery server.

If you are using *implicit* deployment, run *one* process, named promote to release and deploy, on your Software Deployment package. Running this process executes the three explicit processes (stage deployment package, create deployment package, and deploy software) in the proper sequence, without any further input.

Whether you deploy explicitly or implicitly, the CA Harvest SCM server implements the three explicit processes as virtual server UDP functions. That is, the CA Harvest SCM server simulates a new hdeploy command line utility by calling the hdeploy function directly, instead of spawning a process to run it.

11. Optionally, review the records for your package in the HARUSDHISTORY table. This table records information about every process run implicitly or explicitly on a Software Deployment package.

Uninstall and Rollback

CA Harvest SCM lets you uninstall or rollback software that was installed using the CA Harvest SCM integration:

- The uninstall operation removes software that was installed using a CA Harvest SCM package.
- The rollback operation rolls back a version of software, that version must first be uninstalled. Then, the previous version of the software is reinstalled (redeployed). The CA Harvest SCM rollback operation performs these steps when two or more versions of the software were installed using the CA Harvest SCM package.

Note: The CA Harvest SCM package only performs these operations using CA Software Delivery packages that were created using the CA Harvest SCM integration. Other versions of the same software deployed outside of CA Harvest SCM management are not used for uninstall or rollback operations.

Uninstall

The uninstall operation works essentially the same as the deploy package operation; the differences are as follows:

- The software is not copied to the target computers, because the software is presumed to exist and the uninstall command is executed instead of the install command.
- The uninstall operation is always scheduled immediately whereas a deployment can be scheduled immediately or anytime in the future.
- The uninstall operation is only allowed when the software package was successfully deployed, that is, the package status must be set to completed.

Follow these steps:

1. Navigate in the CA Harvest SCM Workbench to the CA Harvest SCM package that was used to install the software.
2. Right-click the package and select Uninstall USD Package from the shortcut menu.
The uninstall is scheduled immediately.

Note: Uninstall after Rollback is not possible using CA Harvest SCM Integration with CA Software Delivery.

More information:

[Package Statuses and Processes](#) (see page 248)

Roll Back

The rollback operation does the following:

1. Schedules an uninstall operation and flags the CA Harvest SCM package as in a pending rollback state.
2. Schedules a deployment for the software that the CA Harvest SCM package deployed.

At least two versions of the software must have deployed successfully using the CA Harvest SCM package. The CA Harvest SCM package tracks the current and previous versions of the CA Software Delivery packages that it created and deployed. Any operations performed with the CA Software Delivery packages external to CA Harvest SCM may result in an error or undesirable results.

Follow these steps:

1. Navigate the CA Harvest SCM Workbench to the CA Harvest SCM package that was used to install the software.
2. Right-click the package and select Rollback USD Package from the shortcut menu.
The uninstall is scheduled immediately. After the status of the uninstall is received from the CA Software Delivery server and the uninstall is successful, the deployment of the previous software package is scheduled.

Package Statuses and Processes

In a Software Deployment package, each platform information form or configuration file is assigned a *platform record* that stores the *status* of the CA Software Delivery deployment package for that platform. For example, when a CA Software Delivery deployment package is staged, it is set to *staged* status. Similarly, when you run the create deployment package process on this package, it is set to *created* status.

If a CA Software Delivery deployment package is in staged status, it can be staged again. If the package is in created status, the only process that can be run on it is *deploy package*. After a package is deployed, it is set to *scheduled* status.

Every time the *table data synchronization interval* is reached, CA Harvest SCM queries the CA Software Delivery server about every package in “scheduled” status. When the reply indicates that the deployment succeeded or failed, the status in the CA Harvest SCM database is set to either *completed* or *failed*. The status of a deployment is typically updated during the synchronization interval that immediately follows the scheduled deploy date and time. If an error occurs during the deployment, the status is updated accordingly when the error is first detected.

The `usdsynchinterval[_version]=minutes` parameter in the `HBroker.arg` file sets the synchronization interval. This parameter defines the interval (in minutes) at which CA Harvest SCM synchronizes certain tables in the CA Harvest SCM database with those in the CA Software Delivery database.

If the attempt to deploy the package is successful, the package status is set to *completed* and the stage deployment package can be run again, explicitly or implicitly. If the attempt to deploy the package is *not* successful, the package remains in created status. Try the following to fix the problem:

- In CA Software Delivery, use the DSM Explorer to attempt to fix the problem with the CA Software Delivery package that you created from CA Harvest SCM.
- In CA Harvest SCM, delete the CA Harvest SCM Software Deployment package and start over. If you want to use the same package name, delete the CA Software Delivery package also, using the CA Software Delivery DSM Explorer.

- If you are deploying on one platform only, delete the Platform Information form or configuration file and create one. Keep in mind that deleting this form or file does not delete the associated CA Software Delivery package.
- If you are deploying on multiple platforms and the deployment succeeds on one or more platforms but fails on other platforms, the platform records are updated to different statuses. In such cases, try using the DSM Explorer at the CA Software Delivery server to resolve the problems with the failed platforms. If your attempts do not succeed, delete the CA Harvest SCM Software Deployment package in CA Harvest SCM and also delete the CA Software Delivery packages on the CA Software Delivery server.

Important! Do not promote a package to a higher state when the deployment at a lower state failed.

Note: For information about configuring CA Software Delivery Integration, see the *Implementation Guide*.

HARUSDHISTORY Table

Every time a CA Harvest SCM user runs a stage, create, or deploy process on a Software Deployment package, CA Harvest SCM creates a record in the HARUSDHISTORY table. Each record in the table includes the CA Harvest SCM package name, the CA Software Delivery package name and version, the date and time, and the result of the process being run. Additionally, if an error occurs, the record includes the web service name, fault code, and fault string for the error.

How to Display Software Deployment Package Data Records

To display the data in the HARUSDHISTORY table, use either the CA Harvest SCM hsql relational database (RDB) query utility or a utility provided by your DBMS.

To display records in the HARUSDHISTORY table using hsql, do the following:

1. In a text file, save the SQL SELECT commands that you want hsql to run.

Important! The hsql utility executes *only* SQL SELECT commands stored in text files. *SELECT must* be the first word in these commands; otherwise, the commands are ignored. The hsql utility cannot alter the contents of the CA Harvest SCM database.

For example, to display all records in the table, enter the following command in your text file, including the quotation marks (" "):

```
"SELECT * FROM HARUSDHISTORY"
```

2. Verify that a CA Harvest SCM server process is running on your computer.
3. Enter one of the following commands. The first command is for Windows, the second for UNIX and Linux.

```
hsql -f filename other parameters
```

```
./hsql -f filename other parameters
```

-f filename

Specifies the name of the input file that contains the SQL SELECT commands that hsql executes.

If you do not enter a complete path name, enter the hsql command from the directory in which the input file resides.

other parameters

Specifies any additional required or optional parameters.

Defaults: hsql reads input from the standard input and write output to the standard output.

Note: For information about using hsql, see the *Command Line Reference Guide*.

A Typical Scenario

Given the power and flexibility of CA Harvest SCM lifecycle management, there are several ways to benefit from the integration with CA Software Delivery. A typical scenario for Development and QA working together to deploy software follows:

1. Developers create change management packages in the Development state. Developers also do the following:
 - Make code changes, unit test, and check in changes.
 - Build and check in the installation code.
 - Promote the packages to Test state.
2. QA tests the changes. Any problems cause packages to be demoted back to Development state to repeat step 1.
3. QA approves the changes packages and promotes them to Software Deployment state.
4. The “Release Packager” creates a Software Deployment Package supplying all necessary parameters including the list of installation code that have been checked in.
5. The Release Packager runs the Promote to Release and Deploy process to deploy the software implicitly.

How to Complete Forms and Form Attachments

When you create a Software Deployment package, the USD Package Information form is created automatically. Before you run the Stage, Create, and Deploy Deployment Package processes (implicitly or explicitly) to create the CA Software Delivery package on the CA Software Delivery server, supply the required information and any optional information you want to include for the Software Deployment package, as follows:

1. Complete the USD Package Information form.
2. For each platform on which you plan to deploy this package, specify the installation parameters for the software being deployed. Specify these parameters using *either* the USD Platform Information form *or* a configuration file attached to this form.

No differences exist in how you complete forms or attachments for implicit or explicit deployment.

Note: For more information about any of the fields or parameter definitions described in this section, see your CA Software Delivery administrator or your CA Software Delivery documentation.

USD Package Information Form

When you create a Software Deployment package, the USD *Package* Information form is created automatically. Use the USD Package Information form to view or modify basic details about the package.

There is *one* USD Package Information Form for a Software Deployment package, whether there are one or several USD *Platform* Information forms or configuration files. The USD Package Information form applies to *all* USD *Platform* Information forms or configuration files that you specify for a Software Deployment package.

USD Package Name Prefix

Defines a package name prefix for the package that you are creating.

The USD package name consists of the package name prefix, followed by the platform operating system name, followed by an internal sequence number starting with 1.

You specify the package prefix, and CA Harvest SCM generates the actual CA Software Delivery package name automatically. The package name cannot be changed.

Note: CA Software Delivery does not allow you to deploy a software package, change its contents, and deploy it again. However, CA Harvest SCM lifecycles typically use this method; for example, by deploying software at the Test state, fixing problems, and redeploying at the Release state. To satisfy the requirement of both CA Software Delivery and typical CA Harvest SCM lifecycles, you must use unique CA Software Delivery package names. In addition, each platform requires a separate CA Software Delivery package. Consequently, to allow multiple deployments, the USD package name requires the prefix, the platform name, and a sequence number.

Example: If you enter Accounting as the prefix value and the platform operating system name is Win_2000, the first USD package name created is:

"Accounting - Win_2000 - 1"

USD Package Version

Specifies the version number of the CA Software Delivery package to be created.

Deploy Date

Specifies the date on which this package is to be deployed. To accept the current date, check the box next to the date or use the calendar drop-down list to select a date.

Important! If you are using this form on the CA Harvest SCM Web Interface, enter the date in *mm/dd/yyyy* format.

Hour

Specifies the hour of the day on which this package is to be deployed. To select an hour, use the hour drop-down list.

Minimum: 00

Limits: 23

Minute

Specifies the minute when this package is to be deployed. To select a minute, use the minute drop-down list.

Minimum: 00

Limits: 59

Vendor

(Optional) Defines the name of the supplier of the software to be installed.

Note: After completing the USD Package Information form, specify the installation parameters for this Software Deployment package.

How to Specify USD Platform Information Parameters

For each platform on which you plan to deploy a Software Deployment package, specify the installation parameters for the software being deployed. Specify these parameters using either a complete USD Platform Information form or a complete configuration file attached to this form. For any *single* platform, you must specify these parameters using the form *only* or an attachment *only*; you *cannot* use a combination of both.

If you are deploying this Software Deployment package on multiple platforms, you can use USD Platform Information forms for some platforms and form attachments for other platforms. Thus, for multiple platforms, you can use any combination of forms and attachments, as long as you provide all required installation parameters for each platform using *either* a complete form *or* a complete form attachment.

Important! You must manually create each USD Platform Information form or configuration file. Only the USD Package Information form is created automatically.

How to Use the USD Platform Information Form or Configuration File

Each form or configuration file specifies information for deploying the software on a specific platform, such as Windows, Solaris, and so on. Specify a unique name for each form or configuration file you create. Thus, if you plan to deploy the same CA Software Delivery package on multiple platforms, you can use the platform name as a prefix or suffix in each form name or file name to help distinguish similar forms or files from each other.

For example, if you are deploying the CA Software Delivery package named Billing2006 on Windows and Novell SUSE Linux, create and use two USD Platform Information forms named Billing2006-Win and Billing2006-NovSUSELnx. Alternatively, create and use two configuration files named Billing2006-Win.txt and Billing2006-NovSUSELnx.txt.

Occasionally, different *levels* of the same platform require different platform information forms or configuration files. For example, you may need to complete one platform information form or configuration file for Windows 2000 and *another* platform information form or configuration file for Windows 2003 Server. However, you may be able to use the *same* platform information form or configuration file for Windows 2000 and Windows 2003 Server.

Note: For more details, see your CA Software Delivery administrator or your CA Software Delivery documentation.

To create and use a USD Platform Information form, do the following:

1. Right-click your Software Deployment package, select Add New Form, click USD Platform Information Form, and save the form with a unique name.
2. Complete the parameters on the form, as explained in the remainder of this section.

To create and use a USD Platform Information configuration file, do the following:

1. Create the file using a text editor and save the file with a unique name.
2. Complete the parameters in the file, as explained in the remainder of this section.
3. Attach the file to your Software Deployment package by opening the package, selecting the Package Information form, and clicking the paper clip icon. Clicking that icon lets you browse your computer and select the file.

The USD Platform Information form or configuration file contains the following sections:

- General Information
- Installation
- Targets
- Procedure Options
- Job Options
- Uninstallation

Each section contains several related parameters. On the form, each section has its own tab. For example, the General Information tab on the form contains the General Information parameters.

Format of the Platform Information Configuration File

The file sections and parameters are named to closely match the names of the Form tabs and fields.

On any platform, instead of using a USD Platform Information form to specify the installation parameters for your Software Deployment package, you can optionally create and use a platform information configuration file in either ini or XML format.

Each section in the platform information configuration file corresponds to the same-named tab on the USD Platform Information form and contains the same parameters. For example, the Installation section of the platform information configuration file corresponds to the Installation tab on the USD Platform Information form and contains the same parameters, for example, Install File, Install Type, Load Path, and so on. Generally, the parameter names on the form correspond one-to-one to the names in the configuration file, but you specify them in different formats. For example, consider the formats for specifying the BasedonPackageName parameter in the General Information section:

- form field: Based on package name - version
- ini parameter: BasedonPackageName=name
- xml parameter: <BasedOnPackageName>name</BasedOnPackageName>

Each section of the form or configuration file uses installation parameters. For each parameter, the name on USD Platform Information form appears first, followed by the parameter's description, and then the equivalent ini and xml parameter specifications.

USD Platform General Information

On the USD Platform Information form or in the USD platform information configuration file, the General Information section specifies basic information about this CA Software Delivery package. The General Information section contains the following parameters.

Operating System

Specifies the target operating system of deployment.

If you are using the Platform Information form, select an option from the drop-down list.

ini parameter: `OperatingSystem=name`

xml parameter: `<OperatingSystem>name</OperatingSystem>`

Based on package name - version

Specifies the name and version of the package on which this update is based. This parameter applies to delta (upgrade) Software Deployment packages *only*.

If you are using the Platform Information form, select an option from the drop-down list.

ini parameters:

`BasedonPackageName=name` and `BasedonPackageVersion=version`

xml parameters:

`<BasedonPackageName>name</BasedonPackageName>` and
`<BasedonPackageVersion>version</BasedonPackageVersion>`

Comment

(Optional) Enables you to add comments to the package.

ini parameter: `Comment=name`

xml parameter: `<Comment>text</Comment>`

Limits: 500 alphanumeric characters

Full USD Package Name

Identifies the full name of the CA Software Delivery package to be created on the CA Software Delivery server when you run the create package process.

Note: This field contains a read-only value that is set at runtime after the installation files are staged and the form is refreshed.

ini parameter: not applicable

xml parameter: not applicable

Last Operation

Identifies the last software delivery operation performed using this CA Harvest SCM package. For example, Stage Package or Uninstall Package.

Note: This field contains a read-only value that is set at runtime after each software delivery operation.

ini parameter: not applicable

xml parameter: not applicable

Result

Identifies the result of the last software delivery operation.

Note: This field contains a read-only value that is set at runtime after each software delivery operation.

ini parameter: not applicable

xml parameter: not applicable

Note: For the list of valid values you can specify for these parameters in your file, see the DSM Explorer on the CA Software Delivery server.

USD Platform Installation

On the USD Platform Information form or in the USD platform information configuration file, the Installation section contains parameters configuring how the software that this CA Software Delivery package contains is installed on the target computer. The Installation section contains the following parameters:

Install File

Defines the name of the file used to perform the installation at the target computer.

ini parameter: InstallFile=*filename*

xml parameter: <InstallFile>*filename*</InstallFile>

Install File Type

Specifies the type of installation to be performed. This value must be one of the following reserved CA Software Delivery values:

- SIP
- CMDFILE
- EXEFILE
- SWDET
- MSI
- SXP
- PKG
- PIF
- IPS
- PALM
- WINCE

ini parameter: `InstallType=type`

xml parameter: `<InstallType>type</InstallType>`

Install Parameters

Defines any parameters to be used during the installation.

ini parameter: `InstallParameters=name`

xml parameter: `<InstallParameters>name</InstallParameters>`

Load Path

Defines the location at the CA Software Delivery server where the installation files are stored and staged. The files stored here are subsequently copied to the CA Software Delivery Package Volume for deployment.

If the path does not exist on the CA Software Delivery server, it is created by CA Harvest SCM using the CA Harvest SCM Remote Agent running on the CA Software Delivery server host.

ini parameter: `LoadPath=path`

xml parameter: `<LoadPath>path</LoadPath>`

Payload

Displays the list of items included in the installation. This list may contain fully qualified file names, item paths, or both. When a path is specified, all items found in the view path are copied to the CA Software Delivery package staging path (Load Path). Processing of the path is recursive as all items found in sub-directories of the path are copied. The parameter may be specified one or more times.

ini parameter: InstallPart=*path*

xml parameter:

```
<Payload>
  <InstallPart>path</InstallPart>
  ...
</Payload>
```

Note: You can optionally have different target states for different platform records. For example, you could define one platform definition named Windows_2000 with target(s) in the Test state, but also specify all platform records with targets in the Release state. The hdeploy process ignores those records that do not have a target assigned for the state in which the CA Harvest SCM package resides.

Limits: 2000 characters

Important! If the deployment requires several files, consider this tip to save time: You can optionally check all files into a single view path and then specify *only* that view path in the Payload parameter.

Targets for USD Platform Information Form Only

On the USD Platform Information form or in the USD platform information configuration file, the Targets section defines the target computers, computer groups, and lifecycle states for this CA Software Delivery package. Because the *formats* for specifying target parameters on a form and in a configuration file are very different, these parameters are described in separate sections. The Targets section on the form contains the following parameters:

Target State

Specifies the lifecycle state or states in which this CA Software Delivery package will be run. Specify *at least one* state from the list of states in the life cycle that you are using to create and run this package.

On the form, from the first Target State drop-down list, select a state in which you want to deploy the package. To specify a second state, select the state from the next Target State drop-down list. Keep selecting states from the Target State drop-down lists until you are finished.

For each state that you specify, you must also specify one target computer *or* one target computer group, but *not* both.

Note: If you have set the context view path, the Target State drop-down list on the form displays *only* the lifecycle states for the current project; otherwise, the drop-down displays all lifecycle states for all projects.

Minimum: 1

Limit: 5

Target Computer and Target Group

Specifies the name of the target computer or target computer group for each target state specified. On the form, for the first target state selected, select a target computer or computer group from the adjacent Target Computer or Target Group drop-down list. Specify *either* one target computer *or* one target computer group, but *not* both.

Similarly, if you specify a second target state, select either a target computer or computer group from the Target Computer or Target Group drop-down list adjacent to the second state.

For each state specified, select either a target computer or computer group, until you are finished.

The Target Computer and Target Group drop-down lists are populated with the computers and computer groups that are defined for CA Software Delivery server. If a computer or computer group does not appear in the drop-down list, you cannot deploy a package to it.

Note: For assistance in defining a computer or computer group to CA Software Delivery, see your CA Software Delivery administrator.

Minimum: 1

Limit: 5

Targets for USD Platform Information Configuration File, ini Only

In the USD platform information configuration file, the Targets section defines the target computers, computer groups, and lifecycle states for this CA Software Delivery package. Because the *formats* for specifying target parameters in an ini or xml configuration file are different, these parameters are described in separate sections.

The Targets section in an *ini* configuration file specifies one or more of the following statements:

state=*[(Group|Computer)]*target

state

Specifies the life cycle state in which this CA Software Delivery package will be run. Specify *at least one* state from the list of states in the lifecycle that you are using to create and run this package.

For each state that you specify, specify one target computer *or* one target computer group, but *not* both.

(Group|Computer)

Specifies whether the target is a computer or computer group. This parameter is required only if the target is defined in CA Software Delivery as *both* a computer and a computer group.

For example, suppose CA Software Delivery includes both a computer and a computer group named Oslo25. To target the computer, specify *state=(Computer)Oslo25*. To target the group, specify *state=(Group)Oslo25*.

target

Specifies the name of a target computer or target computer group that is defined for CA Software Delivery.

If a computer or computer group is not defined, you cannot deploy a package to it.

Note: For assistance in defining a computer or computer group to CA Software Delivery, see your CA Software Delivery administrator.

Minimum: 1 statement

Limit: 5 statements

Example: In an *ini* file, to specify that the package is deployed first at the Development state on the Developers group and second at the Test state on the Quality-Assurance group, use the following parameters:

Development=Developers

Test=Quality-Assurance

Targets for USD Platform Information Configuration File, xml Only

In the USD platform information configuration file, the Targets section defines the target computers, computer groups, and lifecycle states for this CA Software Delivery package. Because the *formats* for specifying target parameters in an *ini* or *xml* configuration file are different, these parameters are described in separate sections.

Similarly, the Targets section in an *xml* configuration file specifies one or more of the following statements:

```
<TargetGroup state=state>target</TargetGroup>
```

```
<TargetComputer state=state>target</TargetComputer>
```

state

Specifies the lifecycle state in which this CA Software Delivery package will be run. Specify *at least one* state from the list of states in the lifecycle that you are using to create and run this package.

For each state that you specify, specify one target computer *or* one target computer group, but *not* both.

target

Specifies the name of a target computer or target computer group that is defined for CA Software Delivery.

If a computer or computer group is not defined, you cannot deploy a package to it.

Note: For assistance in defining a computer or computer group to CA Software Delivery, see your CA Software Delivery administrator.

Minimum: 1 statement

Limit: 5 statements

Example: In an *xml* file, to specify that the package is deployed first at the Development state on the Developers group and second at the Test state on the Quality-Assurance group, use the following parameters:

```
<TargetGroup state=Development>Developers</TargetGroup>
```

```
<TargetGroup state=Test>Quality-Assurance</TargetGroup>
```

Procedure Options

On the USD Platform Information form or in the USD platform information configuration file, the Procedure Options section lets you specify whether users will be logged off target systems and whether target systems will be restarted. You can specify these options before the CA Software Delivery package is run, afterwards, or both. The Procedures Options section contains the following parameters:

Important! The parameters in the Procedure Options section apply to Windows only.

Before Installation Reboot

(Optional) Specifies whether to restart the target systems before the installation.

To enable this feature, select Reboot.

ini parameter: BeforeInstallReboot=Y|N

xml parameter: <BeforeInstallReboot>Y|N</BeforeInstallReboot>

Before Installation Logoff

(Optional) Specifies whether to force users to log off the target systems before the installation starts.

To enable this feature, select Logoff.

ini parameter: BeforeInstallLogoff=Y|N

xml parameter: <BeforeInstallLogoff>Y|N</BeforeInstallLogoff>

After Installation Reboot

(Optional) Specifies whether to reboot the target systems after the installation completes.

To enable this feature, select Reboot.

ini parameter: AfterInstallReboot=Y|N

xml parameter: <AfterInstallReboot>Y|N</AfterInstallReboot>

After Installation Logoff

(Optional) Specifies whether to force users to log off the target systems after the installation completes.

To enable this feature, select Logoff.

ini parameter: AfterInstallLogoff=Y|N

xml parameter: <AfterInstallLogoff>Y|N</AfterInstallLogoff>

Job Options

On the USD Platform Information form or in the USD platform information configuration file, the Job Options section contains parameters for configuring the end-user involvement (if any) for running this CA Software Delivery package. This section also configures runtime options for the CA Software Delivery server. The Job Options section contains the following parameters:

Prompt

(Optional) Specifies whether, before the installation, to prompt the user to accept or decline the installation.

To enable this feature, select Prompt.

Important! This parameter applies to Windows *only*.

ini parameter: Prompt=Y|N

xml parameter: <Prompt>Y|N</Prompt>

Execute Prompt Timeout

(Optional) Specifies whether to run the CA Software Delivery package and install the software automatically if the user on the target computer does not respond to a confirmation prompt before the time-out period for answering the prompt expires. This time-out period is defined by CA Software Delivery.

To enable this feature, select Execute Prompt Timeout.

Important! This parameter applies to Windows *only*. This parameter applies *only* if Prompt is selected.

ini parameter: ExecPromptTimeout=Y|N

xml parameter: <ExecPromptTimeout>Y|N</ExecPromptTimeout>

User Cancel

(Optional) Specifies whether to enable the user on the target computer to cancel an installation after answering Y to a confirmation prompt.

To enable this feature, select User Cancel.

Important! This parameter applies *only* if Prompt is checked.

Possible values: Y or N

ini parameter: UserCancel=Y|N

xml parameter: <UserCancel>Y|N</UserCancel>

Resolve Query Groups

(Optional) Specifies whether CA Software Delivery server performs dynamic computer group evaluation.

To enable this feature, select Resolve Query Groups.

ini parameter: ResolveQueryGroups=Y|N

xml parameter: <ResolveQueryGroups>Y|N</ResolveQueryGroups>

Offline

(Optional) Specifies whether to run this installation offline from the CA Software Delivery scalability server. This option causes the CA Software Delivery agent to disconnect from the CA Software Delivery server while it (the agent) is installing the software on the target.

To enable this feature, select Offline.

ini parameter: Offline=Y|N

xml parameter: <Offline>Y|N</Offline>

Uninstallation Options

On the USD Platform Information form or in the USD platform information configuration file, the Uninstallation section contains parameters for uninstalling the software that this CA Software Delivery package contains from the target computer. The Uninstallation section contains the following parameters:

Uninstall File

Defines the name of the file used to uninstall the software that this package contains from the target computer.

ini parameter: UninstallFile=*name*

xml parameter: <UninstallFile>*name*</UninstallFile>

Uninstall Type

Specifies the type of installation to be uninstalled.

Important! This value *must* be one of the CA Software Delivery reserved values specified for installing this software. You *may* specify the same value for the Install File Type and Uninstall Type parameters, but you are *not* required to do so.

ini parameter: UninstallType=*type*

xml parameter: <UninstallType>*type*</UninstallType>

Uninstall Parameters

(Optional) Defines any parameters needed to uninstall the software that this package contains.

ini parameter: UninstallParameters=*name*

xml parameter: <UninstallParameters>*name*</UninstallParameters>

Sample USD Platform Information Configuration Files

A sample USD Platform Information configuration file in ini format follows:

```
[GeneralInformation]
  OperatingSystem = Any

[Installation]
  InstallFile = install.bat
  InstallType = CMDFILE
  LoadPath = c:\Deploy

  InstallPart = \Test1\InstallParts

[Uninstallation]
  UninstallFile = uninstall.bat
  UninstallType = CMDFILE

[Targets]
  Test = QASystem1
  Software Delivery = ProdSys1
```

A sample USD Platform Information configuration file in xml format follows:

```
<?xml version='1.0' encoding='utf-8'?>
  <USDPlatformInformation>
    <GeneralInformation>
      <OperatingSystem>WIN_2000</OperatingSystem>
    </GeneralInformation>
    <Installation>
      <InstallFile>install.bat</InstallFile>
      <InstallType>CMDFILE</InstallType>
      <LoadPath>c:\Deploy</LoadPath>
      <Payload>
```

```
        <InstallPart>\Test1\USDTest1\InstallParts</InstallPart>
        <InstallPart>\Test1\USDTest1\UninstallParts</InstallPart>
    </Payload>
</Installation>
<Uninstallation>
    <UninstallFile>uninstall.bat</UninstallFile>
    <UninstallType>CMDFILE</UninstallType>
</Uninstallation>
<Targets>
    <TargetComputer state="Software
Delivery">ProdSys1</TargetComputer>
    <TargetGroup state="Test">QASystems</TargetGroup>
</Targets>
</USDPlatformInformation>
```

Formal Problem Tracking

The Formal Problem Tracking lifecycle is used to evaluate problem information and then hold formal reviews to determine the final disposition of modification requests. This lifecycle has no views because it is unnecessary to see or change any repository items. All the information being tracked is in the forms and packages; this lets management track when a modification request was created, the length of time taken to investigate the information, and the disposition of that investigation.

The emphasis in this lifecycle is the Review Board state. Every package must be reviewed in the Review Board state before it can be promoted. You can schedule meetings in the following ways:

- The Review Board is notified every time a request is promoted to the Review Board state, and then a chairperson schedules a meeting to review the requests.
- An external job to CA Harvest SCM is created that totals the requests in the Review Board state on a weekly basis and then forwards this information as a meeting notice.

During the Review Board meeting, a member promotes packages in accordance with the Review Board decisions, and can generate a report that details the results of the meeting.

Note: An alternative is to allow a single person to be responsible for the Review Board state. This person reviews each request when it is promoted, and if it is a minor request, has the authority to promote it with no Review Board approval needed. This person should be someone other than the one who promotes the requests at the Review Board meeting; two different approval processes are required, each with different rights. A user-defined process is then added to the promotion of the request to validate its priority before it is promoted.

The lifecycle can be used as a stand-alone project or can be added to another lifecycle. With a stand-alone project, requests are initiated in one project for all projects and then moved to the appropriate project when ready to be fixed. This is particularly helpful when maintaining one version of a product while working on the next version. The fix might be needed in either version or in both. By generating the request in a stand-alone project, you can select its destination.

How Formal Problem Tracking Works

The flow through the Formal Problem Tracking lifecycle is as follows:

1. Create a request using the Create Package process in the Create Request state. This process creates a package in the Investigate state with an associated Modification Request form named MR- n (n is an incremental number for each new form). This package moves the Modification Request through the lifecycle. When the package is created, a notification is automatically forwarded to the Problem Manager with this message:

[package] has been created on [date] at [time] by [user].

2. In Investigate, update the package form with analysis information. After completing Investigate, promote the package to Invalid or to Review Board:

- The Promote to Invalid process promotes packages to the Invalid state. Any member of the Developer or Problem Manager groups can execute this process. A notification is automatically forwarded to the Development Manager with this message:

Date: [date]

Time: [time]

User: [user]

The following packages:

[package]

have been promoted from [state] to [nextstate].

- The Promote to Review Board process promotes packages to the Review Board state for disposition. Any member of the Developer or Problem Manager groups can execute this process. When the process is executed, a notification is automatically forwarded to the Review Board with this message:

[package] is ready for the Review Board. Please see [package] for more details.

3. The Review Board appraises the request and either demotes it to Investigate for more information or promotes it to Invalid, Fact of Life (known problem but will never be fixed), Schedule, or to Development to be fixed:
 - The Invalid state holds all modification requests that are invalid. If the request needs to be reevaluated, or if it becomes active, use the Demote to Investigate process to enter the request into the lifecycle again. Any user can invoke this process. An automatic notification of the demotion is sent to the Development Manager with this message:

[package] has been demoted from [state] to [nextstate] and must be investigated again.

[package] was demoted by [user] on [date] at [time].
 - The Fact of Life state holds all modification requests that are known to be problems, but will not be fixed. A modification request can be demoted to the Review Board state for review again.
 - The Schedule state holds all modification requests that need to be worked on in the development project. When the Development Manager is ready for the modification request, move the package into the development project or demote it to Review Board for further review. A notification is automatically forwarded to the Development Manager with this message:

User: [user]

The following packages:

[package]

have been moved from the state [state] in project [project] to the state[next state] in project [next project].

New Development

The New Development lifecycle is used to capture into one state the software development milestones. This lifecycle allows an organization to have a separate work area for tracking the project requirements, working on project documentation, doing rapid prototypes, and for test plans and procedures. The results of this lifecycle are snapshots that are used as the baseline for a maintenance lifecycle such as the Release or Production lifecycles.

How New Development Works

The flow through the New Development lifecycle flow is as follows:

1. Create a separate repository for each view and attach it to the project's baseline view.
2. Create a package in the Requirements state for each major requirement based on the requirements document. Promote these packages to the appropriate states and begin work:
 - In the Project Documentation state, documents are created such as user guides, release notes, and so on.
 - The Test Script state is used for writing test plans and procedures for testing the project.
 - In the Rapid Prototype state, code is developed and checked in to CA Harvest SCM. These packages are then promoted to the Testing state for testing.
3. Capture major progress milestones of the Development view using the take snapshot process. This process allows you to return to any baseline. At the completion of the development phase, you can create a maintenance project and attach the snapshots to the baseline view.

New Development - Alternatives

In this lifecycle, each view requires a separate repository because when you take a snapshot, it includes only the files that are active in the current view; the files cannot be seen from the other views. When the snapshot is attached to the new project, it allows you to attach only one copy of the repository.

Alternatively, to use only one repository, you must attach all the states to the same view; therefore, when you take the snapshot, it includes all the latest revisions of all the items in the repository.

OnDemand Project

With the On Demand Project lifecycle template, you (administrators) can set up a project that streamlines project creation for users to create projects using the Workbench, and lets you track who, when, and how the the project was requested, approved, rejected, and retired.

The OnDemand Project lifecycle has four states including Project Request, Approved Requests, Rejected Requests, and Retired Projects. The usage scenario is as follows:

1. The user who is granted access to a project can invoke the project request (create package) process to request the creation of a project. The user then opens the package's associated form to specify required information for the project creation:
 - Create a project based on an existing project template.
 - Create a project repository based on the following options:
 - Create a repository for the project.
 - Associate the baseline repository from an existing project.
 - Take a new snapshot from an existing project's state.
 - Associate an existing snapshot from an existing project.
2. Reviewers approve or reject the Project Request package.
 - When a reviewer approves the package, the following occurs:
 - a. A prelinked server UDP is invoked to create the project based on the request specification, and a notification is sent.
 - b. The project is created, and then the package is moved automatically to the Approved Requests state.
 - When a reviewer rejects the package, the package is moved to the Rejected Requests state.

Note: The Reopen Requests process in the Rejected Requests state lets the user move the package back to the Project Request state for a further review cycle.
3. When the OnDemand Project is completed or retired, the user invokes the Retire Project process to move the package to the Retired Projects state.

The status of project becomes inactive and the project is unavailable for access from the Workbench.

How Administrators Set Up an OnDemand Project

You perform the following tasks as an administrator to set up an OnDemand Project:

1. Log in to the Administrator application and navigate to the Lifecycle Templates folder.
2. Right-click the OnDemand Project template, and select Copy To from the shortcut menu.

The Copy Project dialog appears.

3. Name the project, for example, OnDemand Project Tracking Model, and select the Active Projects folder from the Copy To drop-down list.
4. (Optional) If you want access control duplicated, select the Duplicate Access Control check box. Click OK.

Important! If you do not duplicate access control, you must set access so that users can use the project.

The project is created in the Active Projects folder.

5. Review the access control for all processes in each state of the project you created and grant user group access accordingly.
6. (Optional) Use a text editor to edit the OnDemand Project XML form template to modify fields to meet your needs, for example:
 - Modify the Requested By dbcombo control in the form to show the User Name value instead of the Real Name value (the full name of the user).
 - Modify the project Template dbcombo selection menu to limit specific template types that a user can select when project is created.
7. After you update the XML form template, you can invoke the hformsync command to update the XML form template to the database.
8. Update the Notification processes to meet your custom needs.
9. Log out of the Administrator application.

The OnDemand Project you created is available on the Workbench to the users to whom you granted access.

More information:

[Copy Project](#) (see page 48)

[Set Object-Level Access](#) (see page 117)

[How to Modify a Form Type by Editing the XML File](#) (see page 140)

How to Use an OnDemand Project Life Cycle

The OnDemand Project form lets you create a project lifecycle. You perform the following tasks to complete the form and use an OnDemand Project lifecycle:

1. Log in to the Workbench and navigate to the OnDemand Project lifecycle that the administrator created.
2. Right-click the Project Request state and select Project Request from the shortcut menu.

A Project Request package with an OnDemand Project Request form are created.

3. Double-click the OnDemand Project Request form.

The form appears in the Workbench form editor.

4. Complete the fields in the form. Use *one* OnDemand Project form for an OnDemand Project package. The following fields are not self-explanatory:

New Project Name

(Required) Specifies the name of the project.

Template

(Required) Specifies the lifecycle of the new project.

Copy of Existing Project

(Required) Specifies if the new project repository should be copied from an existing repository.

Repository Name

(Only required if the value in Copy of Existing Project is set to No.) Specifies a name for the new repository and associates the repository with the new project.

Create Project User Group

(Required) Specifies if a user group should be created for the new project.

Select/Deselect Project Group Users

(Optional) Specifies users to remove in from the user group that you created for this project.

Note: This is toggle field that works with the Project User Group Members field.

Project User Group Members

(Read-only) Specifies users to add to the user group that you created for this project.

Note: You can disregard the commas after using the Select/Deselect toggle to remove users.

Project Name

(Only required if the value in Copy of Existing Project is set to Yes.) Specifies the project name that has the associated the repository to be copied to the new project.

Repository/Snapshot Option

(Only required if the value in Copy of Existing Project is set to Yes.) Specifies if the existing project's repository should be associated with the new project. Options for this field are:

- New—Indicates that a new snapshot should be take from the project's state as specified in the State Name field.
- Baseline—Indicates that the existing project's repository (baseline version) should be associated with the new project.
- Snapshot—Indicates that the existing project's snapshot should be associated as the baseline for the new project. If this option is selected, the value in the Existing Snapshot field is required.

State Name

(Only required if the value in Repository/Snapshot Option is set to New.) Specifies the state name that has the associated the repository to be copied to the new project.

Existing Snapshot

(Only required if the value in Repository/Snapshot Option is set to Snapshot.) Specifies the name of the existing snapshot to be used for the baseline for the new project.

5. Save and close the form.

The OnDemand Project lifecycle is set up.

6. Request reviewers to approve or reject the request.

- If the request is approved, the package is promoted to the Approved Requests state and a project is created automatically.
- If the request is rejected, the package is promoted to the Rejected Requests state and a project is not created.

7. Promote projects after their usage is complete to the Retired Projects state.

The Retired Projects state lets Administrators and management track project usage.

More information:

[Baselines](#) (see page 22)

[Repositories in the Baseline](#) (see page 23)

[Configure Baseline](#) (see page 52)

Packaged Application Change Management

This lifecycle lets you record the change process in third-party development projects such as SAP R/3, Oracle Financials, and PeopleSoft. The Packaged Application Change Management lifecycle is used by organizations that want to protect application components from changes that are potentially disruptive or have unacceptable risk associated with them. This lifecycle enables application changes to be initiated, documented, notified, tracked, approved, and promoted. The application changes move through the lifecycle, the rapidity of which depends on the risk and complexity of the change. This lifecycle can also be integrated with existing help desk applications.

How Packaged Application Change Management Works

The flow through the Packaged Application Change Management lifecycle is as follows:

1. Create a request using the Create Package process in the Initiate Request state. This process creates a package in the Initiate Request state with an associated Application Change Request form named PACM-*n* (*n* is an incremental number for each new form).
2. In the Initiate Request state, the application manager evaluates, prioritizes, and documents the package. After the package has been assigned to a developer, it is promoted to the Development state to be worked on, tested, and promoted to QA.
3. In the QA state, quality reviews of the package are initiated and the application manager evaluates the results of the change from the test results. In the case of customized program code, a review is done. If rejected, the Change Owner consults all stakeholders to decide if the package should be demoted for rework or completely rejected.
4. After the package is approved by QA, it is promoted to the Plan Implementation state. In this state, the Change Owner prepares and documents a detailed implementation plan for the package. This plan might include a back-out procedure and a training schedule. Then the package is promoted to Implementation and the change is moved into production. The Package Application Coordinator creates a schedule for the implementation plan and makes available all required resources for the implementation.

5. In the Review state, the QA Coordinator reviews the steps that were used to implement the change and assesses any benefits of the change. The Review state holds all Application Change Requests that have been rejected.
6. The Closed state is the final state for successful Application Change Requests.

Parallel Development

The Parallel Development lifecycle is a variation of the Release lifecycle and shares most of its attributes. Like the Release lifecycle, the Parallel Development lifecycle is linear. Packages are grouped into larger tasks, which in turn are part of a larger release. The support of concurrent development and the merging of changes across projects add greater functionality to the development phase. Versions in the Closed state can also be merged with versions in other projects. The View Previous Release state is used to evaluate the differences between versions, as captured by snapshot views.

More information:

[Define a Cross Project Merge Process](#) (see page 77)

How Parallel Development Works

The flow through the Parallel Development lifecycle is as follows:

1. In the Assigned Changes state, packages are initially defined and the lifecycle begins. To submit a bug report, use the Create Package process to create a package, named with the default prefix Bug #- in this state or you can move a package into the Assigned Changes state from a Problem Tracking project. After a package is approved, move it to a Problem Tracking project or promote it to Coding.
2. Check out for update the item that is associated with the appropriate package in the Coding state. Modify the item and check it in, using the same package with which it was checked out. Continue development in Coding until packages are ready for Unit Testing.

3. Developers or Development Managers can generate a report that shows the packages that are in each state. The following Package Distribution UDP (hsq! -t) calls the HSQL query utility to generate this report:

```
SELECT
projectname, statename, COUNT(*) AS
number_of_packages
FROM
harpackage, harstate, harEnvironment
WHERE
harstate.stateobjid = harpackage.stateobjid AND
harpackage.envobjid = harEnvironment.envobjid AND
harEnvironment.projectname = 'Default Release Model'
GROUP BY
projectname, statename
```

4. Promote the packages to Unit Testing and perform unit testing. If errors occur during testing, demote the package to Coding to be fixed. When testing is complete, the Development Manager approves the package. Packages accumulate in the Unit Testing state until Quality Assurance (QA) is ready for them; then promote the packages to the QA state.
5. After the packages are promoted from the Unit Testing state to the Quality Assurance state, the changes associated with each package are visible in the QA view. QA uses the new versions to validate the new version of the product. If errors are found during testing, demote the package to Coding so the package can be modified and the new changes incorporated; then promote the package to Unit Testing and then to QA. When testing is complete, promote the packages to Closed.
6. In the Closed state, the SCM Administrator reviews the results of QA testing. If not approved, the SCM Administrator demotes the packages to the Quality Assurance state.

The SCM Administrator can use the List Changes process to view the versions that are different. This process compares snapshot views and working views. By comparing snapshot views in the Snapshot state, line-by-line changes can be generated for each item in the repository that differs in the two snapshots. Only the SCM Administrator can invoke this process.

Using the cross project merge process, the SCM Administrator can merge changes from another project into the current project and can resolve merge conflicts using the Resolve Merge Conflicts process.

In the View Previous Release state, the SCM Administrator can use the compare view process to generate an overview report. This report summarizes the differences between any two views, either snapshot or working, which exist in any project.

Note: You can also benefit from using Concurrent Development in this lifecycle.

Production

Production software development takes a different approach than release software. The emphasis in the Production lifecycle is to keep an application operational; the concept of a release does not exist. In a production project, you do not need to track previous versions of the software, but you do need to be able to look at changes between production versions. Because of this difference, the Production lifecycle relies heavily on views and the software that is available in a view at a specific time. Financial institutions, insurance companies, assembly lines, and other commercial organizations are typical production projects.

In the Production lifecycle the Development, Emergency, and Production states share the same view. This helps ensure that when an item is checked out in Development, the latest version is also in Production. If a version is checked out for Update in the Development view and then checked back in, the version is available in Production automatically. Therefore, the only mode of check-out allowed for update is concurrent update; when a version is checked out in Development, it is placed on a branch. This lets users change versions in Development without affecting Production.

Note: When a new file is added to the system, it must be checked in to the Merge state. If you attempt to check in the new file to the Development state, it is automatically seen in Production, which is not wanted. A separate view must be used when you check in the file. A notification process in the Development state sends a mail message notifying the merge engineer that the package has been promoted, and that the new file needs to be checked in.

How Production Works

The flow through the Production lifecycle is as follows:

1. Production needs a modification.

Create a request for the modification using the Create Package process in the Development state. This process creates a package in the Development state with an associated Modification Request form named MR-*n* (*n* is an incremental number for each new form). This package moves the Modification Request through the lifecycle. Any user can create a package.

2. Work on changes, notify the merge engineer, and promote the package.

Check out the items that need modifying using the package you created. Use the Concurrent Update mode to place the versions on a branch. After you make the change, check in the file. Because an item cannot be added to the Production view until it has been tested, notify the merge engineer that the new item must be added to the repository. You must modify the body of the notify process to include the directory name where the file is located, the file name, and the package it must be associated with. Promote the package to the Merge state.

3. Merge the package.

Execute the concurrent merge process using the package. If merge issues exist, use the interactive merge process to resolve conflicts. When all merge issues are resolved, promote the package from the Merge state to the Test state.

4. Test packages.

In the Test state, you can check out changes without affecting the Production view and isolate changes so that updates in the Merge view do not affect the Test view. You can use UDPs to help verify packages:

- To verify that items associated with a package are not also part of an emergency fix, run the Check for Emergency Changes UDP.
- To verify that items being checked out are not branch versions, run the Check for Branch Version UDP.

If you find errors, demote the package to the Merge state so the item can be checked out for update and changed. When the change is complete, promote the package again to the Test state so that the modification can be verified. Promote packages to the Staging state when testing is complete.

5. Accumulate packages in the Staging state and then promote them to Production.

The promote to Production process is scheduled so it does not affect the current operation of the software. The test manager must approve packages, but any tester can promote them. The promote process fails if a package dependency exists for a package that belongs to the Emergency Fix package group. If it does, the package cannot be promoted until the Emergency Fix package has been promoted to the Staging area.

6. (Emergency packages only) Create a package in the Emergency state if a fix is an emergency.

Emergency packages are created on branches and remain on branches until they are merged into production and tested. Check out for Concurrent Update the items that need to be modified. After modification, check in these files. You can run the Make Part of Emer Pkg Group UDP to add a newly created package to the Emergency package group. An emergency build is created and the resulting executable is moved out of the lifecycle to the production computer. Promote the emergency package to the Merge state to integrate it with other changes.

7. (Emergency packages only) Use the interactive merge process to resolve any merge issues and promote the package to Test.

Because an item cannot be added to the Production view until it has been tested, a mail notification that the new item must be added to the repository, is sent to the merge engineer. You must modify the body of the notify process to include the directory name, where the file is located, the file name, and the package it must be associated with. After you have entered this information, click Notify to send the message.

If branch versions need more work, you can add the Validate All on Branch UDP to the Demote to Production process to verify that you demote packages that include branch versions only.

After testing completes successfully, promote the package from Test to Staging and run the Remove from Emergency UDP to remove it from the Emergency Fix package group.

How to Identify and Demote Emergency Packages

As an option to identify and demote packages from Staging easily, name all Emergency Fix packages with a unique identifier.

Example: Use a Unique Identifier

1. Prefix the names of all packages created in the Emergency state with ER- (Emergency Request).
2. Add a demote process from Staging to Test.
3. Add a post-linked UDP to the demote process to check if a package name begins with ER-; if it does, add that package to the Emergency Fix package group.

Check for Emergency Changes UDP

The Check for Emergency Changes UDP verifies that the items associated with the current package are not also part of an emergency fix.

The prerequisite for the Check for Emergency Changes UDP is to configure the CA Harvest SCM server as follows:

- A UDP user that has access to update package in this lifecycle.
- Encrypt the ID and password of that user in a dfo file named udpuser.dfo.
- Place the UDP created for this template in CA_SCM_HOME (or in a location that is in the path).
- Verify that Perl is installed on the CA Harvest SCM server.

The Check for Emergency Changes UDP program follows:

```
perl -S HCheckEmer.pl [broker] "[project]" "<UDP User Encryption File>" ["package"]
```

The Check for Emergency Changes UDP content follows:

```
# Reading Environment Variable
$CA_SCM_HOME = $ENV{CA_SCM_HOME};
if ($^0 eq "MSWin32") {
    $temp = $ENV{TEMP};
}
else {
    $temp = "\\tmp";
}
# Reading the passed variables
die "ERROR!!! Wrong number of variables passed. ($#ARGV)." if ($#ARGV < 3);
$broker = shift;
$env = shift;
$enc = shift;
$slash = "\\" if ($^0 eq "MSWin32");
$slash = "/" if ($^0 ne "MSWin32");
# Name of the emergency package group
$emerpkggrp = "Emergency Package";

$encfile = $CA_SCM_HOME . $slash . $enc;
# Validate location
die "ERROR!!! Could not find the system variable CA_SCM_HOME ($CA_SCM_HOME)." if
(length($CA_SCM_HOME) <= 0);
die "ERROR!!! Could not find the system variable TEMP ($temp)." if (length($temp) <=
0);
die "ERROR!!! Could not find the location for CA_SCM_HOME ($CA_SCM_HOME)." if (! -e
$CA_SCM_HOME);
die "ERROR!!! Could not find the location for TEMP ($temp)." if (! -e $temp);
die "ERROR!!! Could not find the encrypted file ($encfile)." if (! -e $encfile);

# Error code:
$exit_code = 0; # Default is success (0 - success and 1 - fail).

$package = "";
$sql = "";
@sqlresult = "";# Result of the the execution of the SQL script.
```



```
$errmsg = "\n\nWARNING!!!!!! The following files have a dependency with the emergency
packages.\n\n";
```

```
# To remove the packages from the emergency package group we need to first
# find out the package group id. Then we will go in a loop to check if the package
# being promoted is part of the emergency package group. If the package is part of
the
# emergency package group, then we will remove the link between the package group and
# the package.
```

```
for $package (@ARGV) {
```

```
    # First we check if the package is part of the Emergency Fix package group
```

```
    $sql = "Select ltrim(rtrim(COUNT(*)))
FROM harPackageGroup G, harPkgsInPkgGrp PG, harPackage P , harEnvironment E
WHERE PG.pkggrpobjid=G.pkggrpobjid
AND G.pkggrpname=\'$emerpkggrp\'
AND PG.packageobjid=P.packageobjid
AND P.packagename=\'$package\'
AND P.envobjid=E.envobjid
AND E.environmentname=\'$env\''";
```

```
    # Running the SQL script
    @sqlresult = runsql($sql);
```

```
    # If the first value is a number greater than 0 then the package is part of
the Emergency Package group
    if ($sqlresult[0] == 0) {
```

```
        # Now we run a SQL statement which will check if there is any code
dependency
```

```
        # between thepackage being promoted and the emergency package.
        $sql = "SELECT hp.pathfullname||\'\\\'\\\'\\\'||I.itemname as \"File Name\"
FROM harVersions V, harItems I, harpathfullname hp
WHERE V.packageobjid IN (SELECT PG.packageobjid
                        FROM harPackageGroup G, harPkgsInPkgGrp PG
                        WHERE G.pkggrpname=\'$emerpkggrp\'
                        AND PG.pkggrpobjid=G.pkggrpobjid)
AND I.itemobjid=V.itemobjid
AND I.parentobjid=hp.itemobjid
```

```
INTERSECT
SELECT hp.pathfullname||'\'\'\'\'|I.itemname
      FROM harVersions V, harItems I, harPackage P, harEnvironment E,
harpathfullname hp
      WHERE P.packagename='\'$package\'
            AND P.envobjid=E.envobjid
            AND E.environmentname='\'$env\'
            AND V.packageobjid=P.packageobjid
            AND I.itemobjid=V.itemobjid
            AND I.parentobjid=hp.itemobjid";

# Running the SQL script
@sqlresult = runsql($sql);

# Check to see if we got a result.
chomp($sqlresult[0]);
if ($sqlresult[0] eq "HSQL query has Zero results.") {
    # We don't do anything
} # End of the if clause for checking if there is an error.

else {

    # We found a row so we will append that to the list of errors
    $exit_code = 1;

    $tmpval = "";

    for $tmpval (@sqlresult) {
        $errmsg .= "$tmpval\n";
    } # End of the For loop for capturing the error messages
} # End of the else clause of no row selected.

} # End of the emergency package check

} # End of the for each package loop

# Print the error message
if ($exit_code) {

    print "$errmsg\n";

} # End of the if clause for printing the error message

exit $exit_code;
```

```
#####
#      Subroutine to run SQL Statements against Harvest database.

sub runsql
{
  my($sql) = @_ ;

  my $runloc = "$temp";

  # Generating the temporaty file locations where the sql statements will be written.
  my($tmpfilesql) = $runloc . "/" . "hsql" . time() . ".sql";
  my($tmpfileres) = $runloc . "/" . "hsql" . time() . ".dat";

  # Fixing the slashes for the NT operating system
  if ($^O eq "MSWin32") {
    $tmpfilesql =~ s/\/\\/g;
    $tmpfileres =~ s/\/\\/g;
  }

  # Generating the SQL statements.
  open (SQL,">$tmpfilesql");
  print SQL "$sql;\n";
  close(SQL);

  # Running the SQL Statement.
  @QList = `hsql -nh -t -b $broker -eh \"$encfile\" -f \"$tmpfilesql\"`;
  if ($QList[0] eq "") {

    shift(@QList);
  }

  unlink($tmpfilesql);
  unlink($tmpfileres);

  return (@QList);
}
```

Check for Branch Version UDP

The Check for Branch Version UDP verifies that only trunk versions can be checked out of CA Harvest SCM and not branch versions.

The prerequisite for the Check for Branch Version UDP is to configure the CA Harvest SCM server as follows:

- A UDP user that has access to update package in this lifecycle.
- Encrypt the ID and password of that user in a dfo file named udpuser.dfo.
- Place the UDP created for this lifecycle in CA_SCM_HOME (or in a location that is in the path).
- Verify that Perl is installed on the CA Harvest SCM server.

The Check for Branch Version UDP program follows:

```
perl -S hartrunkonly.pl [user] ["version"]
```

The Check for Branch Version UDP content follows:

```
# Defining the program variables
```

```
if ($^O eq "MSWin32")
{
    $tmploc = "$ENV{'TEMP'}" || exitError($usr, "harTrunkOnly-001: TEMP variable
is not set.");
}
else {
    $tmploc = "\\tmp";
}

# Temporary Variable
$tmpval = "";

# Name of the user who is running the process
$user = "$ARGV[0]";

# Full path of the file being checked out
$filepath = "";
```

```

# Version of the file being checked out
$fileversion = "";

# Error Message to be printed
$errmsg = "You cannot check out a branch version of the following file(s).\n\nPlease
rerun the check out process by selecting only the trunk version of this file(s).\n\n";
$errmsg .=
"-----\n";

# Error code:
$exit_code = 0; # Default is success (0 - success and 1 - fail).

# Going to the versions being checked out.
shift;

#####
# Looping through the list of files being checked out to verify that the user
# did not select a trunk version for check out.

for $tmpval (@ARGV) {

    # Split the file path and its version.
    ($filepath, $fileversion) = split(";", $tmpval);

    # Check to see if the file version contains a period (.). If there is no period
then
    # the user is checking out a trunk version.
    if ($fileversion =~ /[.]/) {
        # period found. We have a problem.
        # Set the exit code so that we can raise an error.
        $exit_code = 1;

        # Set the error message.
        $errmsg .= "\nFile: $filepath\nVersion: $fileversion\n";
    }
    else {
        # No Period found so there is no problem with this version.
    }

} # End of the for loop

# Check to see if we found an error. If we found an error then we notify the user and
stop the
# check out process.

if ($exit_code) {

```

```
        # Call the sub routine which will print error and stop the check out process.
        exiterror("$errmsg");

    } # End of the error check.

#####
# Subroutine to print error

sub exiterror {
    print "\nERROR!!!!!!! @_ \nExiting.....\n";
    exit(1);
}
#####
```

Make Part of Emer Pkg Group UDP

The Make Part of Emer Pkg Group UDP adds the newly created package to the Emergency package group.

The prerequisite for the Make Part of Emer Pkg Group UDP is to configure the CA Harvest SCM server as follows:

- A UDP user that has access to update package in this lifecycle.
- Encrypt the ID and password of that user in a dfo file named udpuser.dfo.
- Place the UDP created for this lifecycle in CA_SCM_HOME (or in a location that is in the path).
- Verify that Perl is installed on the CA Harvest SCM server.

The Make Part of Emer Pkg Group UDP program follows:

```
perl -S HMakeEmer.pl [broker] "[project]" ["package"] "<UDP User Encrypted File>"
```

The Make Part of Emer Pkg Group UDP content follows:

```
# Reading Environment Variable
$CA_SCM_HOME = $ENV{CA_SCM_HOME};
if ($^O eq "MSWin32") {
    $temp = $ENV{TEMP};
}
else {
    $temp = "\\tmp";
}
```

```

# Reading the passed variables
die "ERROR!!! Wrong number of variables passed. ($#ARGV)." if ($#ARGV < 3);
$broker = $ARGV[0];
$prj = $ARGV[1];
$pkg = $ARGV[2];
$enc = $ARGV[3];
$slash = "\\" if ($^O eq "MSWin32");
$slash = "/" if ($^O ne "MSWin32");
$epkggrp = "Emergency Package";

$encfile = $CA_SCM_HOME . $slash . $enc;
# Validate location
die "ERROR!!! Could not find the system variable CA_SCM_HOME ($CA_SCM_HOME)." if
(length($CA_SCM_HOME) <= 0);
die "ERROR!!! Could not find the system variable TEMP ($temp)." if (length($temp) <=
0);
die "ERROR!!! Could not find the location for CA_SCM_HOME ($CA_SCM_HOME)." if (! -e
$CA_SCM_HOME);
die "ERROR!!! Could not find the location for TEMP ($temp)." if (! -e $temp);
die "ERROR!!! Could not find the encrypted file ($encfile)." if (! -e $encfile);

print `hup -b $broker -en \"$prj\" -p \"$pkg\" -apg \"$epkggrp\" -eh \"$encfile\"`;

```

Remove from Emergency UDP

The Remove from Emergency UDP removes a package from the Emergency Fix group list.

The prerequisite for the Remove from Emergency UDP is to configure the CA Harvest SCM server as follows:

- A UDP user that has access to update package in this life cycle.
- Encrypt the ID and password of that user in a dfo file named udpuser.dfo.
- Place the UDP created for this life cycle in CA_SCM_HOME (or in a location that is in the path).
- Verify that Perl is installed on the CA Harvest SCM server.

The Remove from Emergency UDP program follows:

```
perl -S hrememer.pl [broker] "[project]" "<UDP User Encrypted File>" ["package"]
```

The Remove from Emergency UDP content follows:

```
# Reading Environment Variable
$CA_SCM_HOME = $ENV{CA_SCM_HOME};
if ($^0 eq "MSWin32") {
    $temp = $ENV{TEMP};
}
else {
    $temp = "\tmp";
}
# Reading the passed variables
die "ERROR!!! Wrong number of variables passed. ($#ARGV)." if ($#ARGV < 3);
$broker = shift;
$prj = shift;
$enc = shift;
$slash = "\\" if ($^0 eq "MSWin32");
$slash = "/" if ($^0 ne "MSWin32");
$epkggrp = "Emergency Package";

$encfile = $CA_SCM_HOME . $slash . $enc;
# Validate location
die "ERROR!!! Could not find the system variable CA_SCM_HOME ($CA_SCM_HOME)." if
(length($CA_SCM_HOME) <= 0);
die "ERROR!!! Could not find the system variable TEMP ($temp)." if (length($temp) <=
0);
die "ERROR!!! Could not find the location for CA_SCM_HOME ($CA_SCM_HOME)." if (! -e
$CA_SCM_HOME);
die "ERROR!!! Could not find the location for TEMP ($temp)." if (! -e $temp);
die "ERROR!!! Could not find the encrypted file ($encfile)." if (! -e $encfile);

foreach $pkg (@ARGV) {

    print `hup -b $broker -en \"$prj\" -p \"$pkg\" -rpg \"$epkggrp\" -eh
    \"$encfile\"`;
}
```

Validate All on Branch UDP

The Validate All on Branch server UDP verifies that only packages that include unmerged versions are demoted.

The Remove from Emergency UDP program follows:

```
perl -S hcheckbranch.pl [broker] "[project]" "<UDP User Encryption File>" ["package"]
```


The Remove from Emergency UDP default input follows:

```
# Reading Environment Variable
$CA_SCM_HOME = $ENV{CA_SCM_HOME};
if ($^0 eq "MSWin32") {
    $temp = $ENV{TEMP};
}
else {
    $temp = "\tmp";
}
# Reading the passed variables
die "ERROR!!! Wrong number of variables passed. ($#ARGV)." if ($#ARGV < 3);
$broker = shift;
$env = shift;
$enc = shift;
$slash = "\\" if ($^0 eq "MSWin32");
$slash = "/" if ($^0 ne "MSWin32");

$encfile = $CA_SCM_HOME . $slash . $enc;
# Validate location
die "ERROR!!! Could not find the system variable CA_SCM_HOME ($CA_SCM_HOME)." if
(length($CA_SCM_HOME) <= 0);
die "ERROR!!! Could not find the system variable TEMP ($temp)." if (length($temp) <=
0);
die "ERROR!!! Could not find the location for CA_SCM_HOME ($CA_SCM_HOME)." if (! -e
$CA_SCM_HOME);
die "ERROR!!! Could not find the location for TEMP ($temp)." if (! -e $temp);
die "ERROR!!! Could not find the encrypted file ($encfile)." if (! -e $encfile);

# Error code:
$exit_code = 0; # Default is success (0 - success and 1 - fail).

$package = "";
$sql = "";
@sqlresult = "";      # Result of the the execution of the SQL script.

$serrmsg = "\n\nWARNING!!!!!!! The following packages have files with trunk
versions.\n\n";

# Create package List
foreach $pkg (@ARGV) {

    $pkglist .= " \' " . $pkg . "\',";
}
chop($pkglist);
```

```
$mysql = "Select rtrim(P.packagename), rtrim(I.itemname), rtrim(V.mappedversion)
from harPackage P, harEnvironment E, harVersions V, harItems I
where P.packagename in ($pkglist)
and E.environmentname=\'$env\'
and P.envobjid=E.envobjid
and V.packageobjid=P.packageobjid
and I.itemobjid=V.itemobjid
and V.mappedversion NOT LIKE \'%.%\'";
```

```
# Run the query to get the list of packages that contain trunk versions.
```

```
@sqlresult = runsql($mysql);
chomp($sqlresult[0]);
# If we got a value back then we flag an error.
if ($sqlresult[0] eq "HSQL query has Zero results.") {
    # We don't do anything
} # End of the if clause for checking if there is an error.

else {

    # We found a row so we will append that to the list of errors
    $exit_code = 1;
    $tmpval = "";

    for $tmpval (@sqlresult) {
        $errmsg .= "$tmpval\n";
    } # End of the For loop for capturing the error messages
} # End of the else clause of no row selected.

# Print the error message
if ($exit_code) {

    print "$errmsg\n";

} # End of the if clause for printing the error message

exit $exit_code;
```

```
#####
# Subroutine to run SQL Statements against CA Harvest SCM database.
```

```
sub runsql
{
    my($sql) = @_;
```

```

my $runloc = "$temp";

# Generating the temporary file locations where the sql statements will be written.
my($tmpfilesql) = $runloc . "\/" . "hsql" . time() . ".sql";
my($tmpfileres) = $runloc . "\/" . "hsql" . time() . ".dat";

# Fixing the slashes for the NT operating system
if ($^O eq "MSWin32") {
    $tmpfilesql =~ s\/\/\\\/g;
    $tmpfileres =~ s\/\/\\\/g;
}

# Generating the SQL statements.
open (SQL,">$tmpfilesql");
print SQL "$sql;\n";
close(SQL);

# Running the SQL Statement.
@QList = `hsql -nh -t -b $broker -eh \"$encfile\" -f \"$tmpfilesql\"`;
if ($QList[0] eq "") {

    shift(@QList);
}

unlink($tmpfilesql);
unlink($tmpfileres);

return (@QList);
}
#####

```

Release

The Release lifecycle is used to make many changes to an application and release those changes as a group identified by a release number. That release is used as an initial point for the next development effort.

The Release lifecycle has the following states: Development, Test, Release, and View Snapshots. Changes are isolated in each state so that they do not affect the released application. Developers or Development Managers can use the Package Distribution Report UDP to generate a report that shows the packages that are in each state.

How Release Works

The flow through the Release lifecycle is as follows:

1. Create a package in the Development state or move a package into the Development state from a Problem Tracking project using the cross project merge process. To create a request in the Development state, select the Create Package process. This process creates a package with an associated Modification Request form named MR-*n* (*n* is an incremental number for each new form). This package moves the Modification Request through the lifecycle. For modification, check out an item for update and associate it with the appropriate package. After modifications are finished, check the file in with the associated package. When development is complete, promote the package to Test.
2. In Test, verify this package with its new version of the item. If errors occur during testing, demote the package to Development. After Development resolves errors, again promote the package to Test. When testing is complete, promote the package to Release.

An alternative scenario follows:

1. The Project Lead receives the design document and creates one package for each function that needs to be modified. At the same time, the Review Board decides which new problems need to be worked on. Developers are assigned specific packages.
2. Developers check out the items for update associated with the appropriate package. Developers modify files and compile new changes. If no compiler errors exist, basic unit tests are performed on the code and the completed code files are checked in. Development continues until the files are ready for integration testing. If errors occur during integration tests, the items are again checked out with the appropriate package, modified, and checked in. When integration testing is complete, the Project Lead promotes the packages to the Test state.

Note: You can use the concurrent development to place each package on a separate branch. After modifying the branch versions, use the concurrent merge process to create one version on the trunk. Then you can run the interactive merge process to integrate changes between two versions and remove the merge tag. After the interactive merge is complete, you can promote the packages to the Test state.

3. When the Test group is notified of the promotion, all the items are checked out to create the Test build. Testers use this build to perform verification and validation tests. If problems occur during testing, the Test group works with the Developers to determine if the package needs to be demoted to have the fix incorporated or if a new package needs to be created. After testing is complete, all packages in the Test state are promoted to Release.

4. When the Release view contains the versions that should be used as the starting point of another release lifecycle, a snapshot is taken with the current date and time and Visible to Other Projects enabled. This snapshot can be attached to the baseline view of the new project that begins with the versions captured by the snapshot. In addition, these snapshots can be viewed in the View Snapshots state. Users can also check out old releases from this state using the snapshot.
5. The Release group creates the release media by checking out the executable items from CA Harvest SCM. Release code can also be distributed using a software distribution package such as the Safer program.

Example: Flow in Release

An example of the flow through the Release lifecycle follows:

1. A file modification is required and a package for it is created in the Development state.
2. This package has an associated modification request form that details the changes that are needed.
3. The items are checked out for update, associated with the package, and are modified by Development.
4. The modified files are checked in to CA Harvest SCM.
5. The package is promoted to the Test state.
6. When testing is complete, the package is promoted to the Release state.
7. To begin work on the next release or if an emergency fix needs to be implemented, a snapshot is taken of the Release view.
8. This snapshot is attached to the baseline view of a new project where additional development is isolated.
9. Changes made for one project can be implemented in another project by performing a cross project merge. A cross project merge lets you select the changes that you want to implement in both projects.

Package Distribution Report UDP

The Package Distribution Report UDP lists how many packages are located in each state of the project.

The prerequisite for the Package Distribution Report UDP is to configure the CA Harvest SCM server as follows:

- A UDP user that has access to update package in this lifecycle.
- Encrypt the ID and password of that user in a dfo file named udpuser.dfo.
- Place the UDP created for this lifecycle in CA_SCM_HOME (or in a location that is in the path).
- Verify that Perl is installed on the CA Harvest SCM server.

The Package Distribution Report UDP program follows:

```
perl -S PkgDistReport.pl [broker] "[project]" "<Encrypted File name>"
```

The Package Distribution Report UDP content follows:

```
# Reading Environment Variable
$CA_SCM_HOME = $ENV{CA_SCM_HOME};
if ($^O eq "MSWin32") {
    $temp = $ENV{TEMP};
}
else {
    $temp = "\tmp";
}

# Reading the passed variables
die "ERROR!!! Wrong number of variables passed. ($#ARGV)." if ($#ARGV < 2);
$broker = $ARGV[0];
$prj = $ARGV[1];
$enc = $ARGV[2];
$slash = "\\" if ($^O eq "MSWin32");
$slash = "/" if ($^O ne "MSWin32");

$encfile = $CA_SCM_HOME . $slash . $enc;

# Validate location
die "ERROR!!! Could not find the system variable CA_SCM_HOME ($CA_SCM_HOME)." if
(length($CA_SCM_HOME) <= 0);
die "ERROR!!! Could not find the system variable TEMP ($temp)." if (length($temp) <=
0);
die "ERROR!!! Could not find the location for CA_SCM_HOME ($CA_SCM_HOME)." if (! -e
$CA_SCM_HOME);
die "ERROR!!! Could not find the location for TEMP ($temp)." if (! -e $temp);
die "ERROR!!! Could not find the encrypted file ($encfile)." if (! -e $encfile);
```

```
$temp sql = $temp . $slash . "hsql$$" . ".sql";

# Create a temp file with the query

$sql = "SELECT environmentname AS Environment, statename as State, COUNT(*) AS
number_of_packages FROM harpackage, harstate, hareenvironment WHERE
harstate.stateobjid = harpackage.stateobjid AND harpackage.envobjid =
hareenvironment.envobjid AND hareenvironment.environmentname = \"prj\" GROUP BY
environmentname, statename";

open ($sql, ">$temp sql");
print $sql $mysql;
close ($sql);

# Executing the query
print `hsql -t -b $broker -eh \"$encfile\" -f \"$temp sql\"`;

unlink($temp sql)
```

Release with Emergency

The Release with Emergency lifecycle is used to make many changes to an application and release those changes as a group identified by a release number. That release is used as an initial point for the next development effort. In addition, this lifecycle is designed to handle any emergency (off hours) changes. The changes are made in the Emergency state as a branch. The branches are deployed to production for emergency purposes and the changes are merged with development code in the Development state.

The Release with Emergency lifecycle has the states: Development, Test, Release, View Snapshots, and Emergency. Changes are isolated in each state (except between Emergency and Product which share the same view) so that they do not affect the released application. Developers or Development Managers can use the Package Distribution UDP to generate a report that shows the packages that are in each state.

How Release with Emergency Works

The flow through the Release with Emergency lifecycle is as follows:

1. Create a request package in the Development state or move a package into the Development state from a Problem Tracking project using the cross project merge process. To create a request in the Development state, select the Create Package process. This process creates a package with an associated Modification Request form named MR-n, where n is an incremental number for each new form. This package moves the Modification Request through the lifecycle. For modification, check out an item for update and associate it with the appropriate package. After modifications are finished, check the file in with the associated package. When development is complete, promote the package to Test.
2. In Test, verify this package with its new version of the item. If errors occur during testing, demote the package to Development. After Development resolves errors, again promote the package to Test. When testing is complete, promote the package to Release.

An alternative scenario follows:

1. The Project Lead receives the design document and creates one package for each function that is modified. At the same time, the Review Board decides which new problems need to be worked on. Developers are assigned specific packages; some are assigned to work on the problems, and some do modifications to incorporate the new functions for the next release.
2. Developers check out the items for update associated with the appropriate package. Developers modify files and compile new changes. If no compiler errors exist, basic unit tests are performed on the code and the completed code files are checked in. Development continues until the files are ready for integration testing. If errors occur during integration tests, the items are again checked out with the appropriate package, modified, and checked in. When integration testing is complete, the Project Lead promotes the packages to the Test state.

Note: You can use the concurrent development to place each package on a separate branch. After you modify the branch versions, use the concurrent merge process to create one version on the trunk. Then you can run the interactive merge process to integrate changes between two versions and remove the merge tag. After the interactive merge is complete, you can promote the packages to the Test state.

3. When the Test group is notified of the promotion, all the items are checked out to create the Test build. Testers use this build to perform verification and validation tests. If problems occur during testing, the Test group works with the Developers to determine if the package must be demoted to have the fix incorporated or if a new package must be created. After testing is complete, all packages in the Test state are promoted to Release.

4. When the Release view contains the versions that should be used as the starting point of another release lifecycle, a snapshot is taken with the current date and time and Visible to Other Projects enabled. This snapshot can be attached to the baseline view of the new project that begins with the versions captured by the snapshot. In addition, these snapshots can be viewed in the View Snapshots state.
5. The Release group creates the release media by checking out the executable items from CA Harvest SCM. Release code can also be distributed using a software distribution package such as the Safer program.
6. (Emergency packages only) Create a package in the Emergency state if a fix is an emergency.

Emergency packages are created on branches and remain on branches until they are merged into production and tested. Check out for Concurrent Update the items that must be modified. After modification, check in these files. You can run the Make Part of Emer Pkg Group UDP to add a newly created package to the Emergency package group. An emergency build is created and the resulting executable is moved out of the lifecycle to the production computer. Promote the emergency package to the Development state to integrate it with other changes.

7. (Emergency packages only) Use the interactive merge process to resolve any merge issues. If branch versions need more work, that work can be performed in the Development state before performing a merge. Promote the package to Test.
8. (Emergency packages only) After testing completes successfully, promote the package from Test to Release and run the Remove from Emergency UDP to remove it from the Emergency Fix package group.

Example: Typical Flow in Release with Emergency

An example of the flow through the Release with Emergency lifecycle follows:

1. A file modification is required and a package for it is created in the Development state.

This package has an associated modification request form that details the changes that are needed.
2. The items are checked out for update, associated with the package, and Development modifies them.
3. The modified files are checked in to CA Harvest SCM.
4. The package is promoted to the Test state.
5. When testing is complete, the package is promoted to the Release state.
6. To begin work on the next release, a snapshot is taken of the Release view.

7. This snapshot is attached to the baseline view of a new project where additional development is isolated.
8. Changes made for one project can be implemented in another project by performing a cross project merge. A cross project merge lets you select the changes that you want to implement in both projects.

Example: Emergency Flow in Release with Emergency

An example of the emergency flow through the Release with Emergency lifecycle follows:

1. An emergency change is required to the production version of the code.
2. A package is created in the Emergency state.
3. The code file is checked out as a branch in the Emergency state and the necessary modifications are made.
4. The modified file is checked back in to the Emergency state and the code is deployed to production.
5. Any other changes that affect the same file and are ready to go to production are stopped until the emergency changes are merged into Development.
6. The emergency package is promoted to the Development state to be merged with the trunk versions.
7. The merged change is promoted to the Test state for further testing.
8. When testing is complete, the package is promoted to the Release state.

Standard Problem Tracking

The Standard Problem Tracking lifecycle lets management track when a modification request was created, the length of time taken to investigate the problem information, and the disposition of that investigation.

The lifecycle can be used as a stand-alone project or can be added to another lifecycle. With a stand-alone project, requests are initiated in one project for all projects and then moved to the appropriate project when ready to be fixed. This is particularly helpful when maintaining one version of a product while working on the next version. The fix might be needed in either version or in both. By generating the request in a stand-alone project, you can select its destination.

The Standard Problem Tracking lifecycle has no views because it is unnecessary to see or change any repository items; all of the information being tracked is in the forms and packages.

How Standard Problem Tracking Works

The flow through the Standard Problem Tracking lifecycle is as follows:

1. Create a request using the Create Package process in the Create Request state. This process creates a package in the Investigate state with an associated Modification Request form named MR-*n* (*n* is an incremental number for each new form).
2. In Investigate, update the form with analysis information and promote the package to Reject or move it to the Development project to be fixed.

Third Party Tool

Some organizations check in their entire software library to CA Harvest SCM and track their third-party software and its associations with their projects. Using this lifecycle, a common archive of all software used in constructing a product includes all the applications that are required to build the product such as the compilers, linkers, and Dynamic Link Libraries (DLLs).

How Third Party Tool Works

The flow through the Third Party Tool lifecycle is as follows:

1. When software is initially received, duplicate the third-party archive project and rename the duplicated project to the software name. Then create a repository for the third-party software and load it with the first release of the software. Associate the repository with the duplicated project.
2. Decide whether to load your repository with the initial code or to check in the code.
 - If you load the repository first, you only attach the baseline to the project. After it is attached, this baseline is visible in all views.
 - If you check in the initial baseline, attach an empty repository to your project, create a package, and check in all the required files. This check-in creates initial versions (version 0) of the items and they are only visible in the Code view.
3. After the Code view contains the current third-party versions, promote the package to the Test state. Tests verify that the new version of the code works with the products that use the code; test the new version of the code in all projects that use the product. If the tests are successful and you want to incorporate the changes, promote the package to the Export Changes state.
4. The Export Changes state holds all versions of the third-party software that have completed testing and that should be used as the baseline view for the next release project. Use the Take Snapshot process to create a snapshot of the Export view, specify the current date and time, and enable the Visible to Other Projects button. This snapshot can then be used for the next release project.

How to Check In a New Baseline

You can check in code to your repository to create a baseline as follows:

1. Create a package in the Receive Code state.
2. Check out all items for Reserve Only.
3. Check in all files of the new software. CA Harvest SCM does one of three things:
 - If the file did not exist, it checks in the new file.
 - If the item exists, and no differences exist, it releases the reserved tag only.
 - If the item exists, and differences exist, it checks in the new version of the file.
4. When the check-in is complete, any items that still have a reserved tag are items that no longer exist or were not part of the release. If you receive a complete update, use the Find Version dialog to delete all the items with a reserved tag.
 - a. In the Find Version dialog, select all the items with a reserved tag.
 - b. Print the list to the output and save the output.
 - c. Click OK.
 - d. Select all the items, and use the Delete Versions process to remove the reserved tag.
 - e. For each item that was displayed in the output, select the item, and select the remove item process. This process marks the item as deleted in the view, but the item remains in the database.

More information:

[How Third Party Tool Works](#) (see page 299)

Version Control

The primary purpose of change control is to track versions of code. The Version Control lifecycle tracks the versions of software and tracks problems associated with that software. The lifecycle is a stand-alone project with two states:

- The Work state, in the Work view, is where you check in files, check out items, and create baselines.
- The Snapshot state does not have a view; you can review the baselines in the Work state.

The Version Control lifecycle is a basic model and a good starting point for any organization. It helps control software without adding significant overhead to the organization. After you are accustomed to controlling the versions of your code, you can use this as a baseline for moving into a more complex model.

How Version Control Works

The flow through the Version Control lifecycle is as follows:

1. Set up the project with one package that tracks all of the changes to the code. The create package process creates a package in the Work state. This package moves the change through the lifecycle and is used as a default during check-in and check-out operations. All files are checked in and out of the Work state.

To separate changes, you can create other packages in this lifecycle. You can also use packages to distinguish different modifications. For example, you might have a package for all modifications for Version 1.0, all changes to the GUI interface, or all items that were changed or added for a new feature.

2. When a baseline is needed, check in all software to the Work state and execute the take snapshot process. This process captures every item in the project and the latest versions that correspond to the date and time specified on the snapshot. After this process is complete, continue working or go to the Snapshot state and select the snapshot to view.
3. The Snapshot state contains all of the snapshot views taken for this project. You can execute the list versions or the compare view processes to view version information.

Peer Review Sample UDPs

The Peer Review Sample UDPs Lifecycle Template is not a complete Lifecycle Project. This template provides sample Peer Review promote processes that have hpeerreviews pre-linked and post-linked UDPs predefined with the various parameter combinations. The CA Harvest SCM administrator can implement an approve or promote process using the hpeerreviews UDP. To do this setup, the administrator copies the desired UDP type to the target state in an active project and configures it as necessary.

CA Harvest SCM provides the following processes in the Development state of this template:

- Promote to Merge - Purge Comments
- Promote to Merge - Verify - Not Required/ALL
- Promote to Merge - Verify - Not Required/ALL/New Version Check

- Promote to Merge - Verify - Not Required/ANY
- Promote to Merge - Verify - Not Required/ANY/New Version Check
- Promote to Merge - Verify - Required/ALL
- Promote to Merge - Verify - Required/ALL/New Version Check
- Promote to Merge - Verify - Required/ANY
- Promote to Merge - Verify - Required/ANY/New Version Check

Appendix B: Using the System Variables

This section contains the following topics:

[System Variables](#) (see page 303)

[clientpath and viewpath Variable Usage](#) (see page 305)

[Lists](#) (see page 305)

[Password System Variable](#) (see page 306)

System Variables

CA Harvest SCM includes system variables that you can use in UDP and notify processes. Some of these variables always have a value; the value for other variables depends on the process being executed.

The following table lists each system variable and provides a brief description and the processes in which the variable has a value:

Variable	Description	Has Value in Pre-Linked	Has Value in Post-Linked
clientpath	Location in the destination directory	Check-in, Check-out	Check-in, Check-out
date	Current date	Any process	Any process
project	Current project name	Any process	Any process
broker	Current broker name	Any process	Any process
file	The complete path and file name of a file (or set of files) in the external file system	Check-in for update and release, or update and keep, interactive merge for version name	Check-in for update and release, or update and keep, check-out, interactive merge for version name
itemobjid	The object ID for an item or item path	Delete version	Delete version
nextproject	The next project as defined by a move package process	None	Move package
nextstate	The next state as defined by a demote, move package or promote process	Demote, promote For a approve process, the values are "approve" or "reject."	Demote, move package, promote For a approve process, the values are "approve" or "reject."

Variable	Description	Has Value in Pre-Linked	Has Value in Post-Linked
package	Current package name or list of names	Approve, check-in for update and release, or for update and keep, check-out for update and concurrent update, create package, demote, interactive merge, move package, notify, promote, UDP	Approve, check-in for update and release, or for update and keep, check-out for update and concurrent update, concurrent merge, create package, demote, interactive merge, move package, notify, promote, UDP
password	Plain-text password of the currently logged in CA Harvest SCM user.	Any process	Any process
pathversion	The complete path name (or set of path names) with a version number appended	Delete version	Delete version
state	Current state	Any process	Any process
time	Current time	Any process	Any process
topackage	Target package name	Switch package	Switch package
user	Current user name	Any process	Any process
version	The complete path and item name (or set of item names) with a version number appended when applicable	Check-out, delete version, interactive merge for version number, list version, notify	Check-in, check-out, concurrent merge, delete version, interactive merge for version number, list version, notify
view	Current view name	Any process	Any process
viewpath	Location in the CA Harvest SCM repository	Check-in, Check-ou	Check-in, Check-ou

You can use system variables in a notify process as part of a message or as a parameter in a UDP. The following message illustrates a way to use system variables (enclosed in brackets) in a notify process:

Date: [date]

Time: [time]

Package [package] has been promoted from [state] to [nextstate] by user [user].

More information:

[Password System Variable](#) (see page 306)

clientpath and viewpath Variable Usage

The clientpath system variable expands to a single value. In certain situations, such as when you drag-and-drop multiple files from different locations into a check-in dialog (check-in from the state level), multiple client paths can occur. In this case, clientpath resolves to the client path of the last item dragged into the dialog, and information about the client paths of individual files is determined by examining the file system variable.

When multiple items with multiple client paths are checked in from the package or version level, the clientpath variable resolves to the client path of the last version (last version name alphabetically) shown in the check-in dialog. If multiple items with multiple client paths are checked in from the state level, the clientpath variable resolves to the client path shown in the From field of the check-in dialog.

When performing a check-in process from the package or version level (not the state level), the viewpath variable always evaluates to a backward slash (\). When the check-in process is performed from the state level, the viewpath variable expands to the value set in the To field of the check-in dialog.

- The values in the first check-in tab, Check In Reserved, will appear in the prelink UDP.
- The values in the second check-in tab, Check In Files, will appear in the postlink UDP.

In the CA Harvest SCM Web Interface, the clientpath and viewpath variables only contain valid values in pre-linked or post-linked UDPs called from the check-in and check-out processes. For example, the stand-alone UDP process in the Web Interface is not able to resolve the clientpath and viewpath values from the set context.

Lists

When the current process selects multiple objects, the value of some system variables can be a list of the [version], [file], or [package] objects. Consider using a list when you design messages that contain these variables. For example, consider the following message:

The following versions:

[version]

Have been checked out by user [user] for package [package].

When this message is expanded, it appears as follows:

The following versions:

```
/App1 Rep/isactive/getack.c;0  
/App1 Rep/isactive/msg.h;2  
/App1 Rep/isactive/writemsg.c;3  
Have been checked out by user john for package  
STR-0043
```

[version] is expanded to include a list of versions. The full path of the version is included, and the version number is appended to the item name after the semi-colon (;).

Windows users can specify the system variables for object names with blanks in two formats:

- Quotes located in brackets expand the system variable list members quoted individually:

[“file”] evaluates to “*file name 1*” “*file name 2*” ...
[“package”] evaluates to “*package name 1*” “*package name 2*” ...
[“version”] evaluates to “*version name 1*” “*version name 2*” ...
- Quotes located outside the brackets expand the system variable list members as one quoted list:

"[file]" evaluates to "*file name 1 file name 2 file name 3...*"
"[package]" evaluates to "*package name 1 package name 2 package name 3...*"
"[version]" evaluates to "*version name 1 version name 2...*"

Password System Variable

The password system variable is disabled by default because it has the following potential for abuse:

Instead of defining a UDP that allows the current user to execute a command-line utility, someone could define a UDP to discover CA Harvest SCM passwords. Therefore, use password system variables only at sites where access to define UDPs is limited to trusted users.

SQL scripts let you update a table column in the CA Harvest SCM database to enable the password system variable. The scripts are located in the database directory under CA_SCM_HOME. To run the scripts, use the following syntax:

Oracle:

```
sqlplus harvest schema owner/password @EnableSysVarPw.sql
```

SQL Server:

```
osql -d DBname -i EnableSysVarPw_sqlserver.sql -U owner -P password -e -b -o log_file
```


Index

A

access

- Admin Form Type • 107
 - Admin Project • 107
 - Admin Repository • 107
 - Admin User • 107
 - Admin User Group • 107
 - Execute • 111
 - form types • 116
 - hsq! • 163
 - item paths • 111
 - items • 111
 - object-level • 109
 - objects • 105
 - Process • 111
 - Project • 109
 - reports • 118
 - Project Access • 118
 - Repository Access • 118
 - Repository • 115
 - SCM-level • 107
 - securable object • 105
 - State • 110
 - summary • 118
 - View Form Type • 116
- Account Locked • 39
- active items • 199
- active reference directories • 199
- active view approach • 199, 202
- Administration utilities
 - hsq! • 162
 - Relational Database Query • 162
- Administrator application
 - logging in • 18
- administrator, tasks • 17
- Application Change Request form type • 274
- approve properties • 64
- arguments, hsmtp • 122
- audit log • 168
 - examples • 177
 - querying • 176
 - resource relationships and hierarchy • 172
- authentication
 - case-sensitivity • 36

- User Properties dialog • 36

B

- baseline
 - working views • 26
- baselines • 22
- baselining • 29
- BLOB storage • 160

C

- CA Harvest SCM
 - administrators • 17
 - audience • 17
- case-sensitivity
 - user name • 36
- check boxes • 129
- Check for Branch Version UDP • 284
- Check for Emergency Changes UDP • 279
- check-in properties • 67
- check-out properties • 70
- combo boxes • 129
- command line utilities
 - hchu • 39
 - hformsync • 139, 140
 - hpolget • 39
 - hpolset • 39
 - hrefresh • 209
 - hsync • 200
 - husrmgr • 38
 - husrunk • 39
 - password policy • 39
- Comment form • 125
- compare view properties • 73
- configuring CA Software Delivery Integration
 - using the Deploy Release Model lifecycle • 242
- create package properties • 74
- cross project merge properties • 77
- custom form types
 - previewing • 136
 - testing • 136

D

- database
 - adding form tables • 139

- tables • 180
- database combo boxes • 129
- database view
 - HARAUDITLOGVIEW • 172, 176
- database, maintaining
 - Oracle • 159
 - SQL Server • 161
- Defect Tracking form type • 125
- delete package properties • 79
- delete version properties • 79
- deleting, items • 95
- demote properties • 66
- Deploy Release Model lifecycle • 242
- directory structure • 23
 - access • 63
- duplicate repository function • 95

E

- editing, custom form types • 133
- email subject setup
 - hsmtp • 124
- encrypted password files
 - hrefresh command • 221
- end of file markers • 97
- end of line markers • 97
- ESD Change Request form type • 125

F

- file conversion • 96
- form controls
 - check boxes • 129
 - combo boxes • 129
 - database combo boxes • 129
 - database format • 133
 - dates • 129
 - label layout • 133
 - labels • 129
 - lines • 129
 - radio button sets • 129
 - tab controls • 129
 - text edit • 129
 - types • 129, 135
- form database tables • 153
- form properties • 133
- form type access • 116
- form types
 - adding form tables to the database • 153
 - Application Change Request • 125, 274

- Comment • 125
- creating • 127
- Defect Tracking • 125
- designing • 128
- editing • 133
- enabling the Harweb-based Form Editor • 147
- ESD Change Request • 125
- field inventory • 132
- form controls • 129, 135
- JavaScripts scripts • 149
- Modification Request • 125, 267, 277, 292, 299
- multiple pages • 131
- OnDemand Project • 271
- Problem Report • 125
- Q and A • 125
- Testing Info • 125
- USD Package Information • 243, 251
- USD Platform Information form or configuration
 - file • 243, 253
- User Contact • 125
- Formal Problem Tracking lifecycle • 266
- formsync command line utility • 139, 140
- full reference directories • 199
- full view approach • 199, 201

H

- Harweb-based Form Editor • 147
- hchu • 39
- hmail • 121
- hpolget • 39
- hpolset • 39
- hrefresh command • 209
 - argument files • 218
 - encrypted password files • 221
 - HRefresh.cfg • 216
 - inventory • 210
 - setup • 215
 - user-defined processes • 209, 222
- HRefresh.arg
 - arguments • 218
 - defining • 218
 - hrefresh command • 218
- HRefresh.cfg
 - hrefresh command • 216
 - permissible values • 216
 - sample file • 216
- hsmtp • 121
 - arguments • 122

- hsmtp.arg • 122
- notify process • 122
- setting up • 122
- setting up the email subject • 124
- hsqldb • 162
 - access • 163
- hsync command • 200
 - exclusion lists • 204
 - nightly builds • 207
 - promotional file management • 207
 - purge option • 204
 - reference directories • 200
- husrunlk • 39
- HUsrUnlk.sql • 39

I

- item access • 111
- item paths
 - access • 111
 - creating • 101
 - deleting • 102
 - renaming • 101
- items • 95
 - baseline • 23
 - deleting • 95
 - moving • 95
 - removing • 95
 - renaming • 95

J

- JavaScript scripts
 - form type field event • 152
 - form type field validation • 151
 - form type initialization • 150

L

- labels • 129
- layout properties • 133
- lifecycles • 21
 - auditing • 168
 - defining • 46
 - Deploy Release Model • 242
 - designing • 33
 - Formal Problem Tracking • 266
 - New Development • 268
 - OnDemand Project • 270
 - Packaged Application Change Management • 274
 - Parallel Development • 275

- planning • 46
- Production • 277
- Release • 291
- Release with Emergency • 295
- Standard Problem Tracking • 298
- Third-Party Tool • 299
- Version Control • 300
- lines • 129
- linked processes • 62
 - auditing • 168, 176
- list version properties • 80
- lists, system variables • 305
- load repository function • 99
- logging in
 - Administrator application • 18

M

- mail interfaces
 - MAPI • 121
 - SMTP • 121
- mail utilities • 121
 - hmail • 121
 - hsmtp • 121
- maintaining the CA Harvest SCM database
 - maintaining the CA Harvest SCM database, Oracle • 159
 - SQL Server • 161
- Make Part of Emer Pkg Group UDP • 286
- MAPI mail interface • 121
- menus
 - Reports • 56
- Modification Request form type • 125, 267, 277, 292, 299
- move item properties • 81
- move package properties • 82
- move path properties • 83

N

- New Development life cycle • 268
- nightly builds
 - hsync command • 207
 - setup example • 207
- notify process
 - hsmtp • 122
 - variables • 303
- notify properties • 84

O

- Object-level
 - access • 109
- objects
 - access • 105
 - processes • 59
 - projects • 21
- OnDemand Project
 - form • 271
 - implementing • 271
 - using • 270, 271
- Oracle, maintaining the database • 159

P

- Packaged Application Change Management lifecycle
 - 274
- packages
 - movement • 32
 - skipping states • 32
- Parallel Development lifecycle • 275
- password policy
 - command line utilities • 39
 - hchu • 39
 - hpolget • 39
 - hpolset • 39
- passwords
 - system variable • 306
- PostCmds command
 - example • 216
- post-linked processes • 62
- PreCmds command
 - example • 216
- pre-linked processes • 62
- previewing, custom form types • 136
- process • 59
 - access • 63
 - defaults • 59
 - defining • 60
 - executing • 60, 63
 - linking • 62
 - post-linked • 62
 - pre-linked • 62
 - types • 60
- Process Access • 111
- process properties
 - approve • 64
 - check-in • 67
 - check-out • 70

- compare view • 73
- create package • 74
- cross project merge • 77
- delete package • 79
- delete version • 79
- demote • 66
- list version • 80
- move item • 81
- move package • 82
- move path • 83
- notify • 84
- promote • 65
- remove item • 86
- remove path • 87
- rename item • 88
- rename path • 89
- switch package • 90
- take snapshot • 91
- user-defined process • 92

Production lifecycle • 277

- Project
 - access • 109
- projects • 21
 - active • 49
 - folders • 49
 - inactive • 49
 - OnDemand Project • 270
- promote properties • 65
- promotional file management
 - hsync command • 207
 - setup example • 207
- Public user groups • 41

Q

- Q and A form type • 125

R

- rearrange repository function • 101
- reference computers • 199
- reference definitions • 199
- reference directories • 199
 - defining • 216
 - refreshing • 199, 200
 - scenario • 200
 - synchronizing • 199
- Relational Database Query Utility • 162
- Release lifecycle • 291
- Release with Emergency lifecycle • 295

remove item properties • 86

remove path properties • 87

removing

items • 95

rename item properties • 88

rename path properties • 89

renaming, items • 95

reports

access • 118

Administrator • 56

Approval Definition • 56

Lifecycle Definition • 56

Project Access • 56

Project Summary • 56

Repository Summary • 56

Secure SCM Access • 56

User Groups • 56

User List • 56

Users • 56

Project Access • 118

Repository Access • 118

Reports menu • 56

repositories • 95

access • 97

Administration application • 95

baselines • 22

creating • 95

directory structure • 23

duplicating • 95

end of file markers • 97

end of line markers • 97

file case interpretation • 96

file conversions • 96

loading • 99

read-only • 23

rearranging • 101

Repositories tab • 95

Repository Access • 97, 115

S

Sarbanes-Oxley Act • 176

scripts

field event for form types • 152

field validation for form types • 151

HUsrUnlk.sql • 39

initialization scripts for form types • 150

JavaScript form types • 149

securable objects • 105

access • 105

servers

repositories • 95

setting access • 108, 117

SMTP mail interface • 121

snapshot views • 28

baselining • 29

creating • 28

deltas • 29

states • 28

SQL Server • 161

SQL Server, maintaining the database • 161

Standard Problem Tracking lifecycle • 298

State Access • 110

states • 31

connecting • 31

data views • 31

demote process • 31

processes • 59

promote process • 31

view associations • 31

storage

BLOB • 160

switch package properties • 90

synchronizing reference directories • 199

system variables • 303

clientpath • 305

lists • 305

password • 306

viewpath • 305

T

tab controls • 129, 131

take snapshot properties • 91

Testing Info form type • 125

testing, custom form types • 136

text edit fields • 129

Third-Party Tool lifecycle • 299

trunks • 23, 29, 30

U

Unicenter Software Delivery Integration

using the Deploy Release Model lifecycle • 242

explicit and implicit deployment • 243

USD Package Information form • 243, 251

USD Platform Information form or

configuration file • 243, 253

unlocking users • 39

- husrunlk • 39
- HUsrUnlk.sql • 39
- User Properties • 39
- USD Package Information form type • 125, 243, 251
- USD Platform Information form or configuration file
 - 243, 253
 - sample files • 265
- USD Platform Information form type • 125
- User Contact form type • 125
- user groups
 - access control • 41
 - administrator • 41
 - approval • 41
 - creating • 35
 - defining • 41
 - notify • 41
 - Public • 41
- User Manager utility • 38
- user name
 - case-sensitivity • 36
- User Properties
 - authentication • 36
 - locked accounts • 39
- user-defined process
 - Check for Branch Version • 284
 - Check for Emergency Changes • 279
 - hrefresh command • 222
 - Make Part of Emer Pkg Group • 286
 - Package Distribution • 275
 - Validate All on Branch • 288
 - variables • 303
- user-defined process properties • 92
- users
 - creating • 35
 - defining • 35, 36
 - modifying • 36
 - unlocking • 39
 - User Properties dialog • 36

V

- Validate All on Branch UDP • 288
- variables • 303
 - clientpath • 305
 - password • 306
 - viewpath • 305
- Version Control lifecycle • 300
- versions
 - base • 29

- View Form Type
 - access • 116
- views
 - baseline • 22
 - snapshot • 28
 - working • 25

W

- working views • 25
 - baseline • 26
 - lifecycle use • 26