



Composite Data Virtualization

Composite PS Promotion and Deployment Tool

Studio Guide

Composite Professional Services

April 2014

Composite Data Virtualization

TABLE OF CONTENTS

INTRODUCTION	5
Purpose.....	5
Audience	5
DISTRIBUTION	6
PS Promotion and Deployment Tool Studio Distribution	6
PDToolStudio Contents	6
VERSION CONTROL AND CIS	7
VCS Topologies.....	7
How Version Control Works in CIS	10
VCS Workspace	11
The Check-Out Process	11
The Check-In Process	12
The Forced Check-In Process	13
Security	14
Resource Locking	14
File Names	14
Supported Version Control Systems	15
Version Control and CIS Resources	15
PREREQUISITES FOR THE COMPUTER THAT HOSTS PD TOOL STUDIO.....	17
Composite Information Server (CIS).....	17
Computer OS	17
Version Control System (VCS) Client Installation.....	17
Subversion (SVN) Client Installation	17
About TortoiseSVN	18
Perforce (P4) Client Installation	18
Concurrent Versions System (CVS) Client Installation	19
Microsoft Team Foundation Server (TFS) Client Installation	20
CONFIGURING VERSION CONTROL FOR PD TOOL	21
CIS VCS Configuration Process for Studio	21
CIS VCS Configuration Process Overview	22
<i>Step 1: Prepare the VCS Repository (VCS Administrator)</i>	23
<i>Step 2: Install the PS Promotion and Deployment Tool Studio (PDToolStudio)</i> ...	23
<i>Step 3: Configure VCS Environment Properties</i>	27
<i>Step 4.1: Initialize VCS Workspace</i>	38
<i>Step 4.2: Initialize VCS Base Folders (VCS Administrator)</i>	42
<i>Step 5: Enable VCS in Studio</i>	45
MANUAL CONFIGURATION TO ENABLE VCS IN STUDIO	45
AUTOMATED CONFIGURATION TO ENABLE VCS IN STUDIO	48
<i>Step 6: Test VCS</i>	50
EXCEPTIONS AND MESSAGES.....	56
VCS SPECIFIC INFORMATION.....	57

Subversion specific information	57
Perforce specific information.....	57
CVS specific information	60
TFS specific information	60
CONCLUSION	68
Concluding Remarks.....	68

DOCUMENT CONTROL

Version History

Version	Date	Author	Description
1.0	08/30/2011	Mike Tinius	Initial revision for PD Tool Studio
2.0	10/04/2011	Mike Tinius	Added Microsoft Team Foundation Server (TFS) support
2.1	01/26/2012	Mike Tinius	Added property P4DEL_LINK_OPTIONS to address Perforce delete link permission issue.
2.2	02/22/2012	Mike Tinius	Added VCS_MESSAGE_PREPEND to allow a statically defined message to be prepended to the VCS Check in or Forced Check in message.
2.3	07/11/2012	Mike Tinius	Added info for drive substitution and network drive mapping. Improved documentation.
2.4	08/15/2012	Mike Tinius	Update drive mapping reference
2.5	04/17/2013	Mike Tinius	Added the ability to generate the Composite Studio "Enable VCS" property file into the user's directory.
3.0	08/30/2013	Mike Tinius	Updated documentation to Cisco format.
3.1	2/18/2014	Mike Tinius	Prepare docs for open source.
3.2	3/6/2014	Mike Tinius	Add TFS specific documentation
3.5	4/12/2014	Mike Tinius	Added ExecutePDToolStudio.bat –vcsinitBaseFoldersfor VCS base folder initialization.

Related Documents

Composite Products Referenced

Composite Product Name	Version
Composite Information Server	5.1, 5.2, 6.0, 6.1, 6.2

INTRODUCTION

Purpose

This document describes how the “**PS Promotion and Deployment Tool Studio**” (**PDTool Studio**) is used to configure Composite Studio to communicate with a version control system (VCS). Once completed, the user will be able to check in and check out CIS (Composite Information Server) objects from Studio directly.

The instructions provided here are for the Studio client setup only. It is expected that a VCS server instance and a VCS repository are already configured. It is also expected that VCS user credentials have been provided.

More details will be provided in this document on how the import/export resource process works alongside the checkout/checkin code process, see the section “*Version Control and CIS*”.

This document has integrated the content from Composite Tech notes including: VCS TechNote-5.1, VCS TechNote-5.2 and VCS TechNote-6.0. User's do not need to use those documents when using PD Tool Studio.

Audience

This document is intended to provide guidance for the following users:

- Architects
- Developers.
- Administrators
- Operations personnel.

DISTRIBUTION

PS Promotion and Deployment Tool Studio Distribution

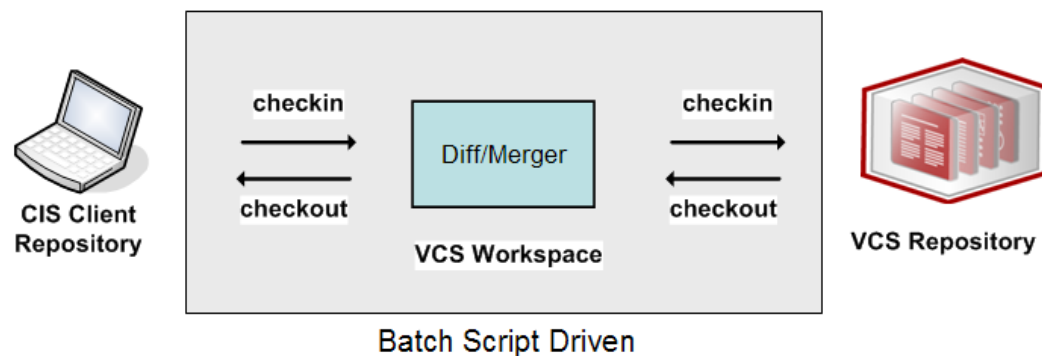
The distribution file for Studio is PDToolStudio.zip.

PDToolStudio Contents

1. **bin** – contains the batch scripts for execution.
2. **dist** – contains the PDTool Studio distribution jar file with the necessary java classes for executing the deployment.
3. **docs** – contains documentation.
4. **lib** – contains libraries referenced by the java classes.
5. **resources/config** – contains the deployment tool configuration files for the spring and logging frameworks as well as studio.properties for PDTool Studio configuration.

VERSION CONTROL AND CIS

Composite Information Server (CIS) supports version control systems to maintain versions of CIS resources. Using Composite Studio or the command line interface and batch script files, CIS users can maintain distinct revisions of CIS resources generated during development using the version control system (VCS) of their preference. The CIS architecture for using a VCS is system agnostic and compatible with a broad array of version control systems, enabling tracking of an artifact's lifecycle or comparing (diffing) its contents across revisions.



The extensibility of the VCS integration framework offered in CIS [5.1, 5.2 or 6.0] allows it to adapt to the special needs of the VCS used by a specific Composite customer.

VCS Topologies

CIS can be used in both single and multi-client environments and share the same VCS. There are four VCS topologies described in this section that are supported by the VCS Studio integration scripts. The VCS topologies include: Single-Node, Multi-Node, Multi-User Managed VCS Access and Multi-User Direct VCS Access.

Single-Node Topology

Single-Node refers to a the scenario where a single Studio user is connected to their own CIS development server and the Studio user has the ability to check-in and check-out their own CIS resources to a VCS repository. In this case, each Studio user has their own workspace which is linked to the same location in the VCS as the other clients. Diagram 1 below depicts the configuration for a Single-Node Topology. Take note of this configuration as the property file is specifically configured for this scenario to set `VCS_MULTI_USER_TOPOLOGY=false`. The result of this setting is that Studio will perform the regular check-in process.

Single-Node Topology

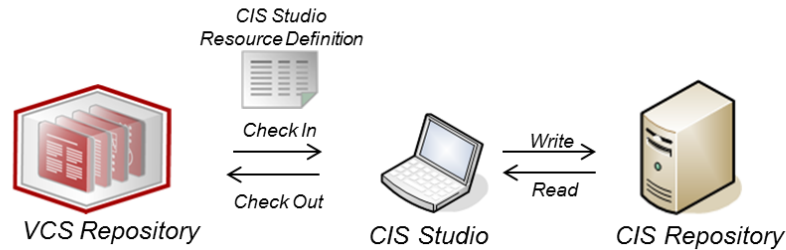


Diagram 1: Single-Node Topology

Multi-Node Topology

Multi-Node refers to the scenario where each Studio user is connected to their own CIS development server and each Studio user has the ability to check-in and check-out their own CIS resources to the shared VCS repository. In this case, each Studio user in the multi-node topology has their own workspace which is linked to the same location in the VCS as the other Studio users. Diagram 2 below depicts the configuration for a Multi-Node Topology. Take note of this configuration as the property file is specifically configured for this scenario to set `VCS_MULTI_USER_TOPOLOGY=false`. The result of this setting is that Studio will perform the regular check-in process.

Multi-Node Topology

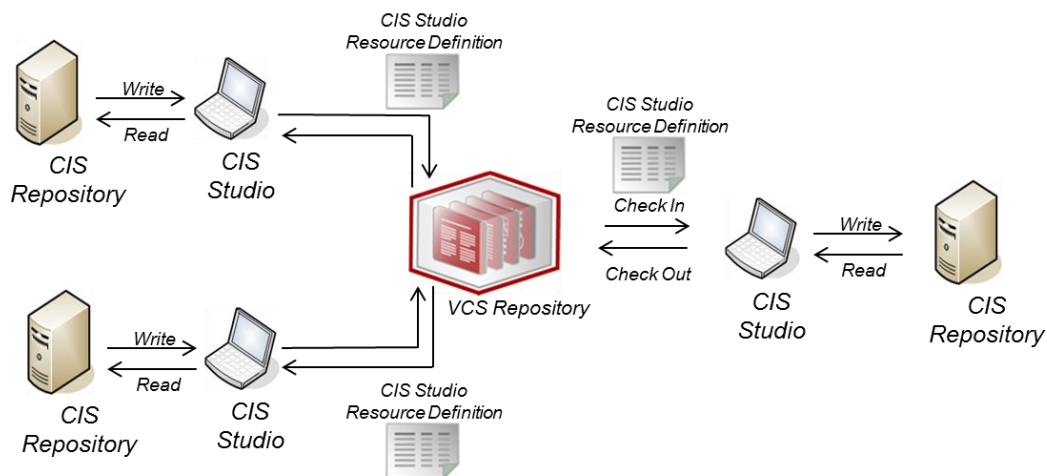


Diagram 2: Multi-Node Topology

Multi-User Topology (Direct VCS Access)

Multi-user Direct VCS Access refers to the scenario where multiple Studio users are directly connected to a shared CIS development server and all Studio users have the ability to directly check-in or check-out CIS resources to the VCS repository. In this scenario, each Studio user

will have their own workspace which is linked to the shared VCS repository. Because each user is pointing to a shared CIS instance, special care must be taken so that the each user's workspace stays synchronized with VCS repository. Diagram 3 below depicts the configuration for a Multi-User Direct Access Topology. Take note of this configuration as the property file is specifically configured for this scenario to set `VCS_MULTI_USER_TOPOLOGY=true`. It is very important that in this type of configuration that the `VCS_MULTI_USER_TOPOLOGY` environment variable be set to true. In fact it is the only scenario where it is set to true. The result of this setting is that Studio will perform the “forced check-in” process. A forced check-in will synchronize the user's workspace first and then perform a diffmerger to determine what in the CIS tree that the user clicked on needs to be checked in to the VCS repository. The script will automatically redirect the user from the regular check-in to the forced check-in process.

Multi-User Topology (Direct VCS Access)

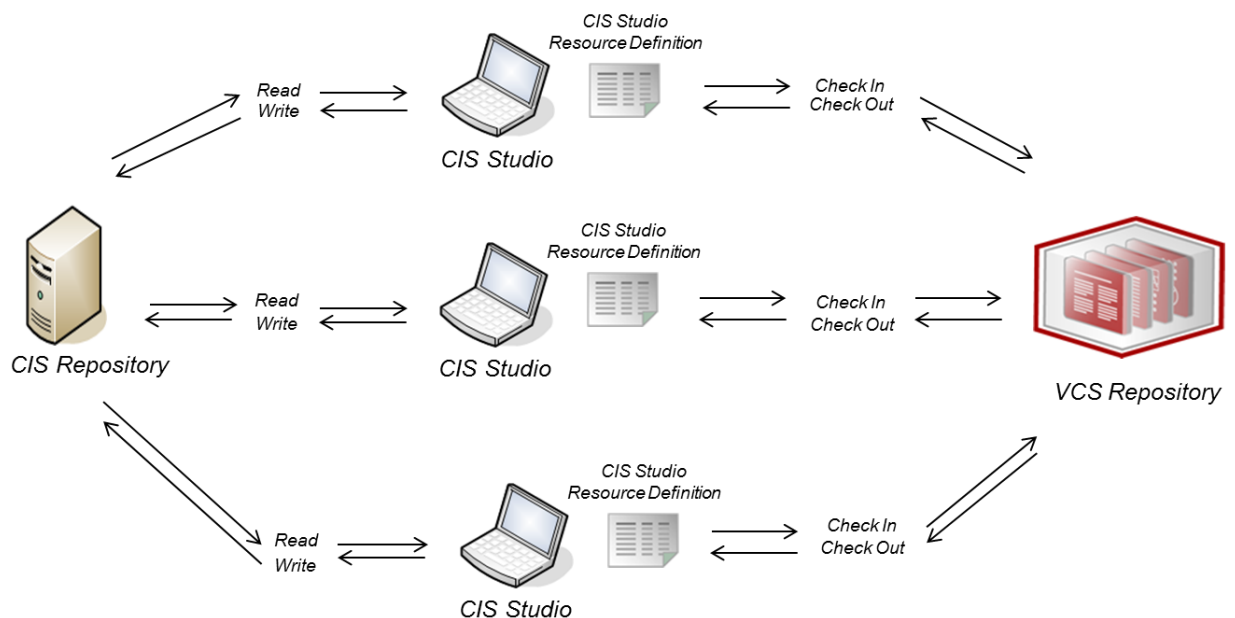


Diagram 3: Multi-User Topology (Direct VCS Access)

Multi-User Topology (Managed VCS Access)

Multi-User Managed VCS Access refers to the scenario where multiple Studio users are connected to a shared CIS development server but only one “Managed” Studio user has the ability to check-in and check-out CIS resources to the VCS repository. Think of this Studio user as the Manager for the group. All VCS related activity is controlled through this single control point. In this case, only the “Managed” Studio user in the multi-user topology has their own workspace which is linked to the VCS repository. Diagram 4 below depicts the configuration for a Multi-User Topology. Take note of this configuration as the property file is specifically configured for this scenario to set `VCS_MULTI_USER_TOPOLOGY=false`. The result of this setting is that Studio will perform the regular check-in process.

Multi-User Topology (Managed VCS Access)

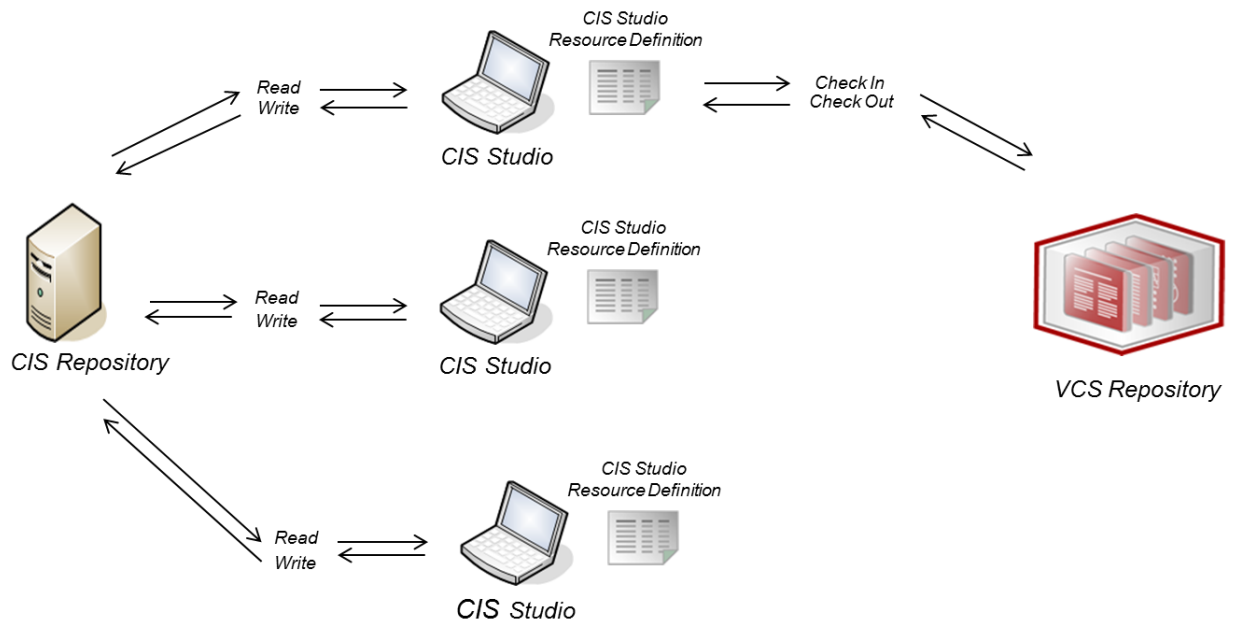


Diagram 4: Multi-User Topology (Managed VCS Access)

How Version Control Works in CIS

VCS integration with CIS is provided using a generic export and import mechanism from within Studio or from the command line. You can check in or check out individual Studio resources or entire directories to and from files in a workspace staging area in the file system. The staged resource files are then compared with the contents of the VCS and processed (that is created, update, or deleted) by batch script files modified for your VC system.

Version control can be invoked using either the Composite Studio interface (Windows clients only) or from the command line. In both cases, this implementation provides integration with Subversion, Perforce and Concurrent Versions System. Standard VCS-specific operations such as commit, update, or revert are not provided within Studio but rather within the batch scripts.

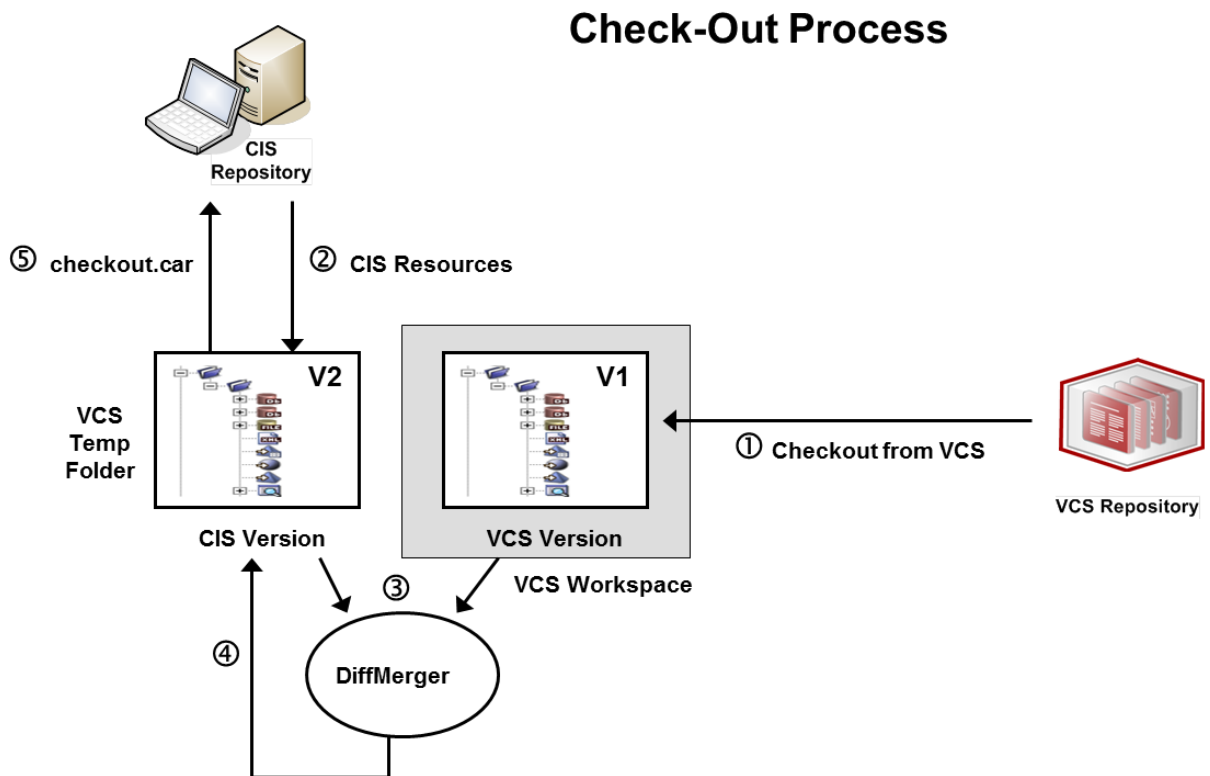
The Lifecycle Listener controls the VCS-specific lifecycle processes required to pre-process and post-process files that are checked in and checked out of the VCS. Sample listeners are provided for Subversion; otherwise a listener can be implemented in Java. Contact Composite Professional Services for an engagement.

VCS Workspace

The CIS VCS implementation utilizes a workspace that acts as a bridge between the CIS and VCS repositories. The VCS workspace is a user-defined folder in the file system that reflects the Composite Studio resource tree folders and data source files with one file per resource.

The Check-Out Process

The process to check-out resources archived in a version control system is illustrated below.



The check-out process shown in the illustration involves these steps:

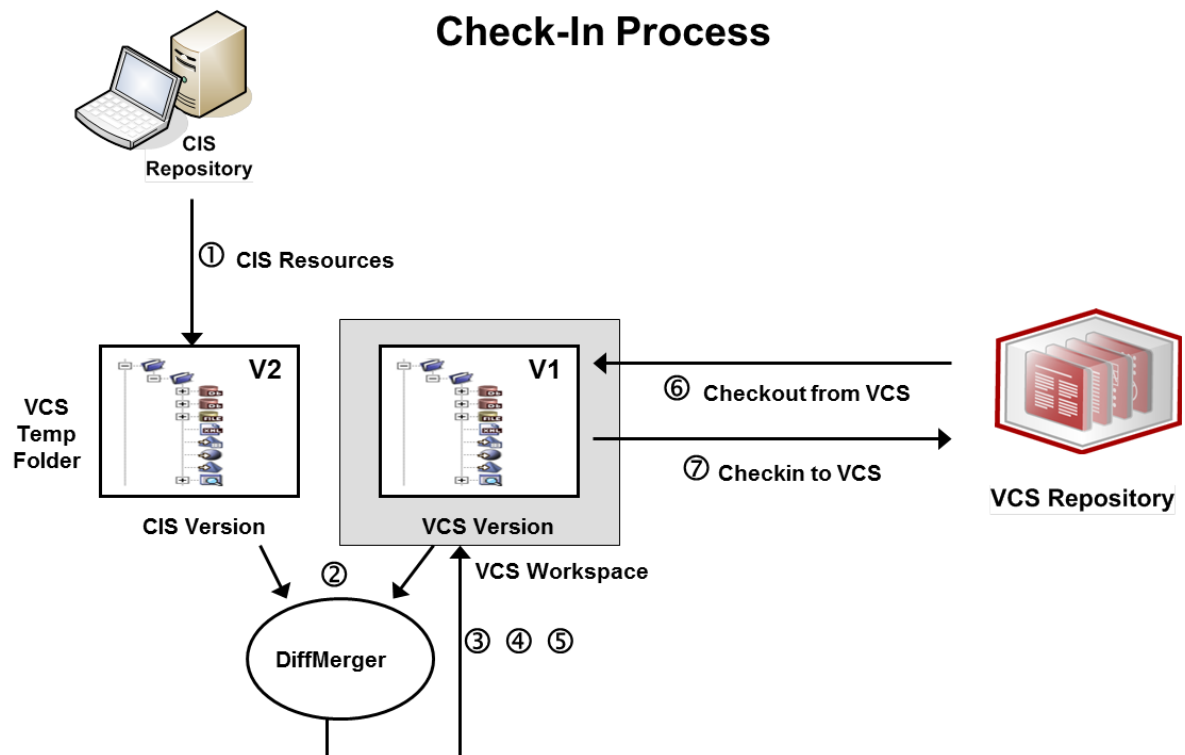
1. A current snapshot of the specified CIS resources is obtained from the VCS workspace (V1).
2. The current version of the same resource subtree in CIS is captured into the VCS temp folder (V2).
3. The DiffMerger batch script is applied against the CIS and VCS versions to compute V1-V2. V1-V2 captures the required modifications (that is resource creations, updates or deletions) required to convert subtree V2 into V1.
4. The DiffMerger batch script is applied to V2, using as input V1 and V1-V2, to modify the contents of subtree V2, so that they match the contents of V1 and package all resource changes in a checkout.car file.
5. The checkout.car file is imported into CIS repository.

6. The VCS Workspace is cleaned up; V2, the checkout.car file, and the V1-V2 results are deleted.

Each of the above processes provides feedback indicating its successful completion or the reasons why it failed.

The Check-In Process

The check-in process to save CIS resources in a version control system is illustrated below.



The check-in process shown in the illustration involves these steps:

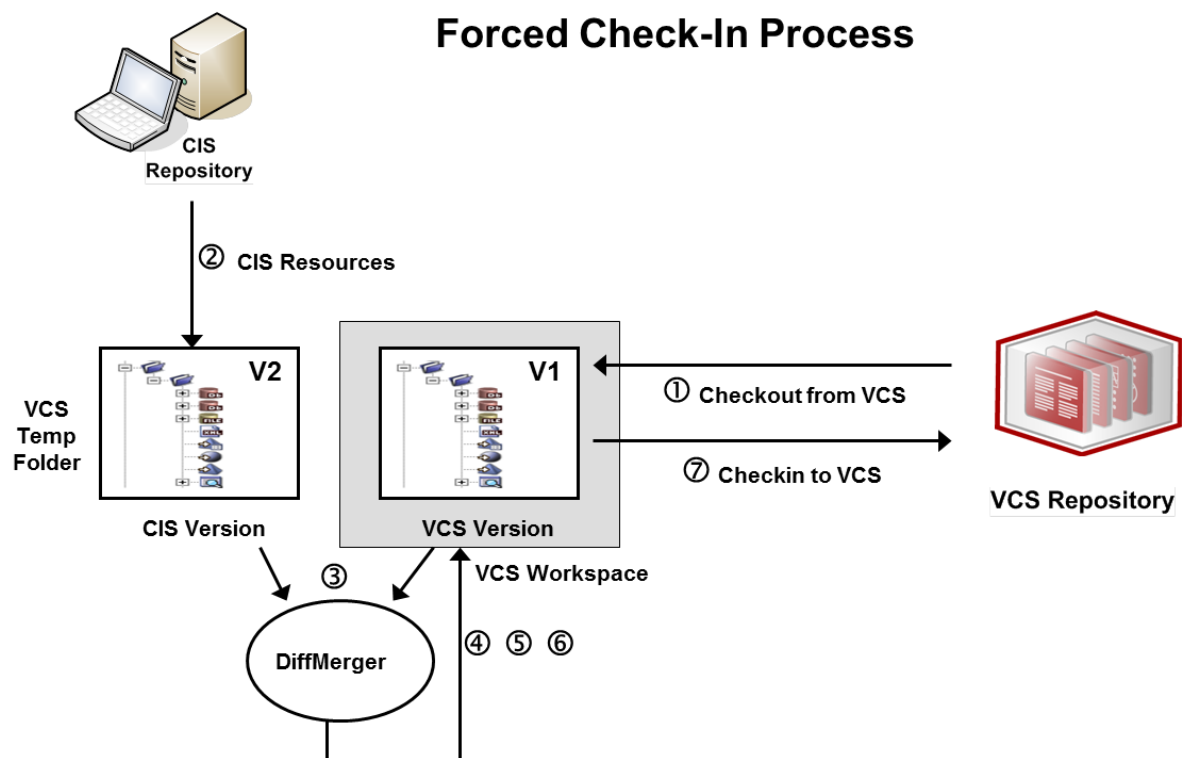
1. The current version of the same resource subtree in CIS is captured into the VCS temp folder (V2).
2. The DiffMerger batch script is applied against the CIS and VCS versions to compute V2-V1. V2-V1 captures the required modifications (that is resource creations, updates or deletions) required to convert subtree V1 into V2.
3. The VCS-specific pre-processor in the Lifecycle Listener is applied on V1, using as input V2 and V2-V1, to prepare the workspace V1 for the application of the modifications described in V2-V1.
4. The DiffMerger batch script is applied to V1, using as input V2 and V2-V1, to modify the contents of workspace V1, so that they match the contents of V2.

5. The VCS-specific post-processor in the Lifecycle Listener is applied on V1, using as input V2 and V2-V1, to prepare the workspace V1 for checking into the VCS.
6. A current snapshot of the specified CIS resources is obtained from the VCS workspace (V1).
7. V1 is checked into the VCS.
8. V2 and V2-V1 are deleted.

Each of the above processes provides feedback indicating its successful completion or the reasons why it failed.

The Forced Check-In Process

The forced check-in process to save CIS resources in a version control system is illustrated below.



The check-in process shown in the illustration involves these steps:

1. A current snapshot of the specified CIS resources is obtained from the VCS workspace (V1).
2. The current version of the same resource subtree in CIS is captured into the VCS temp folder (V2).

3. The DiffMerger batch script is applied against the CIS and VCS versions to compute V2-V1. V2-V1 captures the required modifications (that is resource creations, updates or deletions) required to convert subtree V1 into V2.
4. The VCS-specific pre-processor in the Lifecycle Listener is applied on V1, using as input V2 and V2-V1, to prepare the workspace V1 for the application of the modifications described in V2-V1.
5. The DiffMerger batch script is applied to V1, using as input V2 and V2-V1, to modify the contents of workspace V1, so that they match the contents of V2.
6. The VCS-specific post-processor in the Lifecycle Listener is applied on V1, using as input V2 and V2-V1, to prepare the workspace V1 for checking into the VCS.
7. V1 is checked into the VCS.
8. V2 and V2-V1 are deleted.

Each of the above processes provides feedback indicating its successful completion or the reasons why it failed.

Security

The same security rules apply as in a non-version-controlled CIS environment. The VCS check-in and check-out processes are CIS security sensitive as follows:

- CIS authentication is required.
- CIS authorization rules are applied:
 - Check-in by a certain CIS user applies only to those CIS resources that are read-accessible by that user.
 - Check-out by a certain CIS user may be performed if all the resources being checked out are write-accessible by that user.

Resource Locking

The VCS check-in/check-out processes use the identical resource locking semantics applied to a regular CAR export/import. In particular, a locked resource may be checked-in by any user who has read access to the resource; however a previous version of a locked resource may be checked-out only by the lock owner.

File Names

When working against a VCS, CIS resources need to be represented in the form of (.cmf) files that are stored on the file system of the CIS user's workspace. Different file systems impose restrictions on the characters that are allowed in file names while CIS does not impose any restrictions on resource names. To ensure that names are valid, CIS resource names are normalized before being used as a file name. During the normalization process, characters that are not letters, digits, '-' or '_' are "escaped" or represented through a format that involves only the "safe" characters, listed above. Specifically, all characters for which the following method returns false are escaped using the form _xxxx, where xxxx is the hexadecimal

character code. The underscore character '_' is escaped using '__' (that is, two underscore characters). For example a folder name that contains spaces "/shared/My Folder" will be encoded as "/shared/My_0020Folder".

```
private static boolean isAccepted(char c) {  
    return Character.isLetterOrDigit(c) || c == '_' || c == '-';  
}
```

The utility named "vcs_name_codec", located in the <CIS_HOME install directory>\bin folder, can be used to determine the encoded/decoded form of resource/file names.

Usage: vcs_name_codec [-encode] [-decode] <namespacePath>









Case collisions on case-insensitive operating systems occurring within the same check-in session will cause the check-in to fail.

Supported Version Control Systems

The architecture of CIS version control is VCS-agnostic so that any version control system is supported. The VCS-specific commands to perform version control operations including update, check in, check out, and revert are specified within the PS Promotion and Deployment Tool. Composite Professional Services is available for engagements to enhance the PS Promotion and Deployment tool to work with other VCS systems besides what is supported out-of-the-box.

Version Control and CIS Resources

The following CIS resources can be placed in version control:

-  ▪ Containers (a.k.a. FOLDERS)
-  ▪ Data sources
-  ▪ Definition sets
-  ▪ Links (published services)
-  ▪ Procedures (SQL Script, XQuery, XSLT, etc.)
-  ▪ Tables (Relational Table, View, Flat File, etc.)
-  ▪ Trees (XML Files)
-  ▪ Triggers

The following objects are not amenable to version control in this release:

- System (built-in) resources (for example, built-in procedure 'Print')
- Composite Designer resources *
- Server configuration files *
- Statistics files
- Capabilities files
- Users, groups or domains

These objects are in standard file form and could be put under version control independently of the facilities provided by CIS version control.

Each supported artifact is persisted in a human readable file form in the file system and is versioned to reflect file format changes across product release versions. The file hierarchy in the file system reflects the resource hierarchy in Studio.

PREREQUISITES FOR THE COMPUTER THAT HOSTS PD TOOL STUDIO

Composite Information Server (CIS)

The Promotion and Deployment Tool Studio (PD Tool Studio) supports CIS 5.1, CIS 5.2 and CIS 6.0.

Computer OS

The PD Tool Studio is supported on a number of different OS platforms that CIS runs on including Windows 7 and Windows XP.

Version Control System (VCS) Client Installation

The computer must have a VCS command line client tool installed. This document discusses three version control systems (Subversion, Perforce and Concurrent Versions System). Each VCS is described in the following sections.

1. [**ACTION:**] Have an administrator install a VCS client and provide you with the connection information to the VCS Server.

Subversion (SVN) Client Installation

The computer must have a VCS command line client tool installed. The recommended command line tool for Subversion tool is:

```
CollabNet Subversion Command-Line Client v1.6.15 (for Windows)
http://www.collab.net/downloads/subversion/
```

When installing the above client tool, use all default values.

Regardless of the client tool chosen, it is required that the command “**svn**” be available from any directory, without being fully qualified with a directory path. If you are not sure, go to any directory and type the command below. It should display some help information.

```
C:\temp>svn help
usage: svn <subcommand> [options] [args]
Subversion command-line client, version 1.6.15.
Type 'svn help <subcommand>' for help on a specific subcommand.
```

Version information for Subversion:

```
Type 'svn --version' to see the program version and RA modules
or 'svn --version --quiet' to see just the version number.
svn, version 1.6.15 (r1038135)
  compiled Nov 24 2010, 15:10:19
Copyright (C) 2000-2009 CollabNet.
Subversion is open source software, see http://subversion.apache.org/
```

This product includes software developed by CollabNet (<http://www.Collab.Net/>).

The following repository access (RA) modules are available:

- * ra_neon : Module for accessing a repository via WebDAV protocol using Neon.
 - handles 'http' scheme
 - handles 'https' scheme
- * ra_svn : Module for accessing a repository using the svn network protocol.
 - with Cyrus SASL authentication
 - handles 'svn' scheme
- * ra_local : Module for accessing a repository on local disk.
 - handles 'file' scheme
- * ra_serf : Module for accessing a repository via WebDAV protocol using serf.
 - handles 'http' scheme
 - handles 'https' scheme

About TortoiseSVN

TortoiseSVN 1.7.1 or higher may be used if the command line client is installed. Previous versions did not have a command line client and thus could not be used.

Perforce (P4) Client Installation

The computer must have a VCS command line client tool installed. The recommended command line tool for Perforce tool is:

Perforce 2010.2 (for Windows)

<http://www.perforce.com/perforce/downloads/index.html>

When installing the above client tool, use all default values.

Regardless of the client tool chosen, it is required that the command “**p4**” be available from any directory, without being fully qualified with a directory path. If you are not sure, go to any directory and type the command below. It should display some help information.

```
C:\temp>p4 help
```

```
Perforce -- the Fast Software Configuration Management System.
```

```
p4 is Perforce's client tool for the command line. Try:
```

```
p4 help simple          list most common commands
p4 help commands       list all commands
p4 help command        help on a specific command
p4 help charset        help on character set translation
p4 help configurables  list server configuration variables
p4 help environment    list environment and registry variables
p4 help filetypes      list supported file types
p4 help jobview        help on jobview syntax
```

```
p4 help revisions      help on specifying file revisions
p4 help usage          generic command line arguments
p4 help views          help on view syntax

The full user manual is available at 

### Version information for Perforce:


```

```
Type 'p4 -V' to see the program version

Perforce - The Fast Software Configuration Management System.
Copyright 1995-2011 Perforce Software. All rights reserved.
Rev. P4/NTX86/2010.2/295040 (2011/03/25).
```

Concurrent Versions System (CVS) Client Installation

The computer must have a VCS command line client tool installed. The recommended command line tool for CVS tool is:

```
Concurrent Versions System Command-Line Client 1.11.12 (for Windows)
http://www.nongnu.org/cvs/
```

When installing the above client tool, use all default values.

Regardless of the client tool chosen, it is required that the command “**cv**s” be available from any directory, without being fully qualified with a directory path. If you are not sure, go to any directory and type the command below. It should display some help information.

```
C:\temp>cvs help
Usage: cvs [cvs-options] command [command-options-and-arguments]
  where cvs-options are -q, -n, etc.
    (specify --help-options for a list of options)
  where command is add, admin, etc.
    (specify --help-commands for a list of commands
    or --help-synonyms for a list of command synonyms)
  where command-options-and-arguments depend on the specific command
    (specify -H followed by a command name for command-specific help)
  Specify --help to receive this message

The Concurrent Versions System (CVS) is a tool for version control.
For CVS updates and additional information, see the CVS home page at
http://cvs.nongnu.org/
```

Version information for CVS:

```
Type 'cvs -v' to see the program version

Concurrent Versions System (CVS) 1.11.22 (client/server)
```

Copyright (C) 2006 Free Software Foundation, Inc.

Senior active maintainers include Larry Jones, Derek R. Price, and Mark D. Baushke. Please see the AUTHORS and README files from the CVS distribution kit for a complete list of contributors and copyrights.

CVS may be copied only under the terms of the GNU General Public License, a copy of which can be found with the CVS distribution kit.

Specify the --help option for further information about CVS

Microsoft Team Foundation Server (TFS) Client Installation

The computer must have a VCS command line client tool installed. The recommended command line tool for Microsoft Team Foundation Server tool is:

Team Explorer Everywhere (TEE) 11

<http://www.microsoft.com/en-us/download/details.aspx?id=30661> - TEE-CLC-11.0.0.1306.zip

Microsoft Visual Studio Team Foundation Server 2010 or 2012 is the collaboration platform at the core of Microsoft's application lifecycle management solution that helps enable teams to reduce risk, streamline interactions and eliminate waste throughout the software delivery process. Team Explorer 2010 or 2012 is the client SKU that allows you to access the Team Foundation Server functionality. Install Team Explorer Everywhere 11 which comes as a separate download.

Microsoft Visual Studio Team Foundation Server 2005 is also supported.

In order for TFS integration to work, the command line tools for TFS must be installed on the machine Studio resides on. The main client tool needed is "TF.CMD" which does all the communicating to the TFS server.

TF.CMD comes with Visual Studio. A customer may have a license with Microsoft to provide these tools as a separate installation called "Team Explorer" which is a "light" version of Visual Studio, along with graphical and command-line client tools to connect to TFS. Regardless, the TF.CMD binary, at a minimum, needs to be available on the machine.

If any modifications are needed to the batch files included, a very useful online reference for TF.CMD commands can be found at: [http://msdn.microsoft.com/en-us/library/z51z7zy0\(v=VS.80\).aspx](http://msdn.microsoft.com/en-us/library/z51z7zy0(v=VS.80).aspx).

Version information for Team Explorer Everywhere:

```
C:\temp>tf help
```

```
Team Explorer Everywhere Command Line Client (version 11.0.0.201306181526)
```

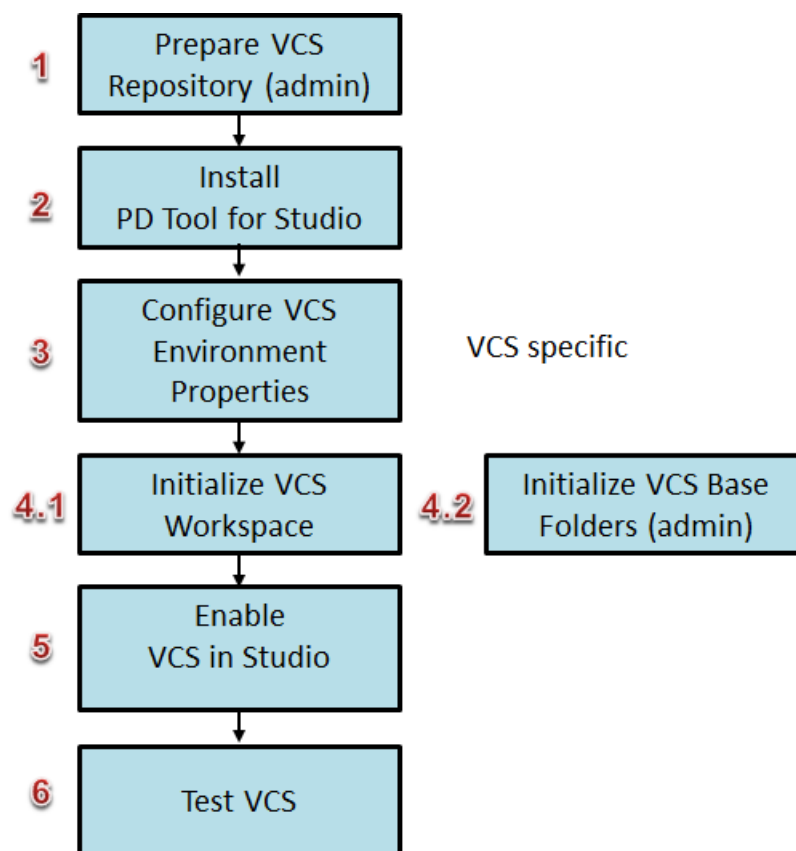
```
Type tf help <command name> for command line description.
```

CONFIGURING VERSION CONTROL FOR PD TOOL

The process to configure version control for CIS involves creating the VCS repository for CIS files, setting up a workspace, connecting the workspace to the VCS, editing property files that customize the version control process for the specific VCS environment and enabling VCS in Studio.

An overview of the process is shown below and described in the following sections.

CIS VCS Configuration Process for Studio



Follow the steps in this section to implement version control for CIS. Before you start the process:

- Be able to create a VCS repository (step 1 below). Generally, this isn't done on the same machine running Studio. You might need to contact the VCS administrator for help or permission to do this.

CIS VCS Configuration Process Overview

1. [Step 1: Prepare the VCS Repository \(admin\)](#)

This step is performed by an administrator and is used to prepare a place in the VCS repository for CIS Objects. The recommendation is to have at least one folder “cis_objects” to hold the Composite repository objects.

2. [Step 2: Install the PS Promotion and Deployment Tool Studio \(PDToolStudio\)](#)

This step performs the actual installation of the PDTool zip file.

3. [Step 3: Configure VCS Environment Properties](#)

This step configures the deploy.properties file with the necessary environment specific values.

4. Step 4: Initialize VCS

- a. [Step 4.1: Initialize VCS Workspace](#)

This step initializes the user’s workspace which maps the local workspace directory to the VCS and then checks out the CIS objects thus synchronizing the VCS with the local workspace.

- b. [Step 4.2: Initialize VCS Base Folders \(admin\)](#)

This step is performed by an administrator after initializing their workspace. It is used to check in the Composite repository base-level folders. It is also used to check in any intermediate folders leading up to the CIS folders that the user will be checking in.

5. [Step 5: Enable VCS in Studio](#)

This step is an iterative step used to configure the VCS Module XML configuration file and the servers XML configuration file. This provides an alternative approach to the deploy.properties file when configuring VCS-specific connection information.
deployment plan tells PDTool what to methods to execute.

6. [Step 6: Test VCS](#)

This step is an iterative step used for testing the VCS configuration. Continue testing and refining the property files as needed.

Step 1: Prepare the VCS Repository (VCS Administrator)

[STOP] FOR NON-ADMINISTRATORS: Proceed to Step 2.

[STOP] FOR ADMINISTRATORS ONLY: This step is for Administrators only. This step is provided so that the user may understand the bigger picture that a VCS Repository is required to be installed somewhere on the Customer' network. A user would never have a VCS Repository installed on the same machine as they are installing CIS and Studio. Please contact your version control administrator and request access URL's and login credentials for your version control system.

1. Define the VCS repository to capture the CIS source files. You might need to request that the VCS system administrator do this for you.

Subversion example:

Open a command prompt and enter this command:

```
svnadmin create C:\cisvcsrepository --fs-type fsfs
```

where cisvcsrepository is a new repository folder in your Subversion system.

This new directory will appear in your root.

Step 2: Install the PS Promotion and Deployment Tool Studio (PDToolStudio)

Installation of the Studio scripts is straightforward as it is nothing more than a zip file.

1. System Requirements:

- **Java 6** - JRE 1.6 is required to be present on the system.
- **Read/Write Access** – The script must have read/write access to the file system in the directory that the script executes in. The location of the VCS Workspace must have read/write access by the user. Log files will be written as well.
- **VCS Client** – a suitable VCS client has been installed on the client-machine.
 - Note: When using Subversion, Tortoise-SVN is not a suitable client.

[STOP] ALL: Please contact your version control administrator to acquire a suitable version control client for your computer and install it. It will be required when configuring PD Tool Studio.

2. Installation Instructions:

- a. **Project Folder** - Select a project folder

- i. **Note:** Normal installation would be installing PDToolStudio into the Composite Software "conf/studio" directory. For example: C:\Program Files\Composite Software\CIS 5.2.0\conf\studio. Therefore, the location of the project folder for the unzipped PDToolStudio would be:

C:\Program Files\Composite Software\CIS 5.2.0\conf\studio\PDToolStudio

- ii. **Note:** If a computer is to be used by multiple users logging in then both the PD Tool Studio scripts and workspace must be installed into the APPDATA directory where the user data is kept. **APPDATA** is a Windows system default environment variable which points to the logged in user's working directory. The location would look something like "C:\Users\<user>\AppData\Roaming" where <user> represents the user name for the logged in user. If the logged in user is "user1", then the location of the project folder for the unzipped PDToolStudio would be:

C:\Users\user1\AppData\Roaming\PDToolStudio

The reason behind installing the PDToolStudio scripts and workspace in the User's AppData directory is due to the fact that the stuiod.properties contains the VCS username and encrypted VCS password.

- b. **Copy** – Copy the PDToolStudio.zip to the project folder
- c. **Unzip** – Unzip PDToolStudio.zip to the project folder you selected in step a.
- d. **Configure setVars.bat**
 - i. Edit PDToolStudio/bin/setVars.bat
 - ii. Set JAVA_HOME=<your java path>
 - 1. The JAVA_HOME environment variable points to a JRE 1.6 home on your computer. Composite installs with JRE 1.6 if you do not have one installed. For example: set JAVA_HOME=C:\Program Files\Java\jre6.
 - iii. Set MIN_MEMORY=-Xms256m
 - 1. The minimum java heap memory to use. The default is "-Xms256m".
 - iv. Set MAX_MEMORY=-Xmx512m
 - 1. The maximum java heap memory to use. The default is "-Xmx512m".
 - v. Set CONFIG_PROPERTY_FILE=studio.properties
 - 1. Environment variable. It is recommended to leave it set as the default CONFIG_PROPERTY_FILE=studio.properties. This refers the configuration file in PDToolStudio/resources/config.
 - vi. Set SUBSTITUTE_DRIVE=S:

1. The purpose of this is to shorten the overall path and avert the “filename too long” issue when checking in or checking out resources.
2. The drive letter to use as a substitution for the location of PDToolStudio home directory. Change this variable if you already have a “S:” mapped. It can be any unused drive letter. The actual “SUBST” command is performed in the ExecutePDToolStudio.bat batch file.

a. **Note, PDTool uses “P:” for its substitution variable so as to avoid collisions when PDTool and PDToolStudio are installed on the same client.**

3. SUBST primer

- a. List drives: type “SUBST” on the command line.
- b. Delete an existing substitution: type “SUBST /D [drive:]” on the command line.
- c. What if PDTool Executes and you get the message: “Drive already SUBSTed”?
 - i. Either change the drive letter starting with step 2 above or delete the substitution using SUBST /D [drive:]

4. Note: substitution is set by default. To disable the use of substitution, simply set the variable to nothing: [set SUBSTITUTE_DRIVE=]

5. All other variables below SUBSTITUTE_DRIVE should be left alone unless mapping a network drive.

vii. To Map a network drive set the PROJECT_DIR to the computer name and path of PDTool

1. PROJECT_DIR=\\<computer-name>\<path-to-PDTool>

viii. Save and exit.

e. **Configure log4j.properties (defaults can be used out-of-the-box)**

The following instructions are used if editing log4j.properties is desired.

- i. **Note: The default values can be used out-of-the-box. Adjust values to turn on debugging.**

Instructions for modifying log4j.properties

- ii. **Edit** PDToolStudio/resources/config/log4j.properties using a text editor.
- iii. It is up to the user to determine what levels of logging are required. This rule is at the heart of log4j. It assumes that levels are ordered. For the standard levels in order, we have DEBUG < INFO < WARN < ERROR < FATAL.
- iv. The following describes the “app.log” settings. The app.log is used for output of the invoked Java Actions.

```
log4j.rootCategory=WARN,stdout,FileAppender

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ISO8601} %t %p [%c] - <%m>%n
log4j.appender.FileAppender=org.apache.log4j.RollingFileAppender
# Forward slashes (/) must be used when defining log file path
log4j.appender.FileAppender.File=../logs/app.log
log4j.appender.FileAppender.MaxFileSize=1MB
log4j.appender.FileAppender.MaxBackupIndex=1
log4j.appender.FileAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.FileAppender.layout.ConversionPattern=%d{ISO8601} %t %p [%c] -
<%m>%n
log4j.appender.FileAppender.Threshold=DEBUG
```

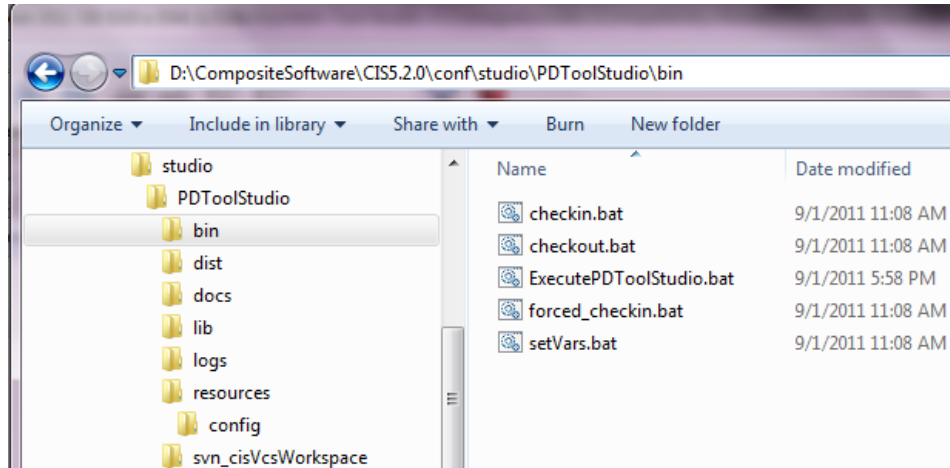
Modify to DEBUG to get more output information if there are issues. The following describes logging general behavior settings:

```
# This rule is at the heart of log4j. It assumes that levels are ordered.
# For the standard levels in order, we have DEBUG < INFO < WARN < ERROR < FATAL.
# Set to DEBUG if there are script issues and you want to get more output
information
# PDTool Command-line Orchestration Debugging
log4j.logger.com.compositesw.ps.deploytool.CisDeployTool=INFO
# PDTool Module Debugging for Command-line and Ant (Applies to *ALL* Modules)
log4j.logger.com.compositesw.ps.deploytool=INFO
# Common Utility Framework Debugging includes:
# [CommonUtils, PropertyManager, PropertyUtil, ScriptExecutor, XMLUtils,
JdbcConnector]
log4j.logger.com.compositesw.ps.common.util=INFO
# VCS Diffmerger and Archive Services Debugging
log4j.logger.com.compositesw.cmdline=INFO
# Spring Framework Debugging
log4j.logger.org.springframework=WARN
```

Note: This should be left at ERROR

```
# Note: This should be left at ERROR
log4j.logger.com.compositesw.ps.common.scriptutil=ERROR
```

- f. An example of PDToolStudio.zip unzipped into “C:\Program Files\Composite Software\CIS 5.2.0\conf\studio” is shown below:



Step 3: Configure VCS Environment Properties

1. **Modify studio.properties** – located in ../resources/config directory. There are also several example studio.properties files that can be used when setting up VCS for Subversion, Perforce or CVS.

- a. **General Instructions for studio.properties**

- i. **PROJECT_HOME** is a variable referenced and is automatically set upon invocation of the scripts and is based on relative location of PDToolStudio/bin.
- ii. **APPDATA** is a Windows system default environment variable which points to the logged in user's working directory. The location would look something like “C:\Users\<user>\AppData\Roaming” where <user> represents the user name for the logged in user.
- iii. Always use forward slashes for both Windows and Unix paths and URLs.
- iv. Variables may use \$ or % notations. It is not operating system specific.
- v. Variables may resolve to this property file, Java Environment (-DVAR=val) or the System Environment variables
- vi. Surround variables with two \$ or two % when concatenating strings (e.g. \$VCS_TYPE\$_cisVcsWorkspace)

- b. **Section: General Behavior Environment Properties**

```
# =====
# Behavioral Environment Variables [default values set]
```

```

=====
# DEBUG=true|false :: Turn on when debugging this script
# Debug Level 1: Debug PDTool script only
DEBUG1=false
# Debug Level 2: Debug ExecuteAction, ExecuteVCS
DEBUG2=false
# Debug Level 3: Debug 3rd level scripts invoked from ExecuteAction and ExecuteVCS
DEBUG3=false
# Diffmerger Verbose allows the VCS Diffmerger process to output more information when
set to true [Default=true]
DIFFMERGER_VERBOSE=true
#-----

```

c. Section: VCS Environment Variables

- i. Set VCS_MULTI_USER_TOPOLOGY=[true|false]
 1. **Single-Node Topology:** Set VCS_MULTI_USER_TOPOLOGY=false
 2. **Multi-Node Topology:** Set VCS_MULTI_USER_TOPOLOGY=false
 3. **Multi-User (Managed) Topology:** Set VCS_MULTI_USER_TOPOLOGY=false
 4. **Multi-User (Direct) Topology:** Set VCS_MULTI_USER_TOPOLOGY=true

```

=====
# VCS Environment Variables [Optional]
#-----
# Note: If you are not using VCS then it is not necessary to set these variables
#
#-----
# VCS Topology Scenarios
#-----
# Instructions:
# There are four VCS scenarios described below. What is important is whether the VCS
Multi-User [Direct VCS Access] Topology is being used (true) or not (false). The
default is to set VCS_MULTI_USER_TOPOLOGY=false.
# -----
# 1. Single-Node Topology
# -----
# Single-Node refers to a the scenario where a single Studio or PD Tool user is
connected to their own CIS development server and the Studio user or PD Tool client has
the ability to check-in their own CIS resources to a VCS repository.
# -----
# 2. Multi-Node Topology
# -----
# Multi-Node refers to a the scenario where each Studio user or PD Tool client is
connected to their own CIS development server and *EACH* Studio user or PD Tool client
has the ability to check-in their own CIS resources to the central VCS repository.
# -----
# 3. Multi-user Topology [Managed VCS Access]

```

```
# -----
# Multi-user Managed VCS Access refers to a the scenario where multiple Studio users or
# PD Tool clients are connected to a central CIS development server and only one *MANAGED*
# Studio user or PD Tool client has the ability to check-in CIS resources to the VCS
# repository. Therefore, the check-in process is managed through a single control point.
# -----
# 4. Multi-user Topology [Direct VCS Access]
# -----
# Multi-user Direct VCS Access refers to a the scenario where multiple Studio users or
# PD Tool clients are directly connected to a central CIS development server and all
# Studio users or PD Tool clients have the ability to *DIRECTLY* check-in CIS resources to
# the VCS repository. This scenario "requires" that forced_checkin be called so that the
# each individual workspace is synchronized with the central CIS repository first and then
# the
# VCS scripts perform a diffmerger to determine what to check-in to the VCS repository.
# In this scenario, the Studio user or PD Tool client is automatically redirected from the
# check-in process to the forced_checkin process even the Studio user does not check the
# forced check-in box. This is required for this scenario.
# -----
# Set the Topology mode here:
# -----
VCS_MULTI_USER_TOPOLOGY=false
#
#-----
# VCS_MULTI_USER_DISABLE_CHECKOUT - In a multi-user, central CIS development server
# environment it may be advantageous to disable the ability for users to perform checkout
# so that they do no inadvertantly remove resources from other people. In this scenario,
# it would be preferred to have a single CIS administrator checkout resources into the
# central CIS repository.
VCS_MULTI_USER_DISABLE_CHECKOUT=false
#-----
#
#
```

- i. Set VCS_TYPE according to the chart shown below. For Team Foundation Server (TFS), there are two versions supported which are identified by tfs2005 or tfs2010 or tfs2012. The type tfs2005 supports commands for TFS 2005 and 2008 whereas tfs2010, tfs2012, or tfs2013 supports commands for TFS 2010, TFS 2012 or TFS 2013.

```
#-----
# VCS_TYPE - The type of VCS being used [svn, p4, cvs, tfs2005, tfs2010, tfs2012,
# tfs2013 etc.]
# Note: This gets added to the end of the VCS_WORKSPACE_HOME folder for workspace
# clarification
# Subversion=svn
# Perforce=p4
# Concurrent Versions Systems=cvs
# Team Foundation Server=tfs2005 or [tfs2010, tfs2012, tfs2013]
VCS_TYPE=svn
#
# VCS_FULL_COMMAND - [true|false] -
# Execute the VCS command with the full path (true) or the VCS command only (false).
# When set to false, the VCS_COMMAND must be in the system path
VCS_EXEC_FULL_PATH=true
#-----
#
#
```

```

#-----
# VCS_HOME - VCS Client Home directory where the VCS executable lives.
#   Note: This could be a /bin directory.
#   It must be where the VCS_COMMAND is found.
VCS_HOME=/usr/bin
#-----
#
#-----
# VCS_COMMAND - The actual command for the given VCS Type [svn,p4,cvs]
VCS_COMMAND=svn
#-----
#
#-----
# VCS options - options are specific to the VCS type being used and are included in the
command line (not set as environment variables)
# Subversion examples:
# --non-interactive --no-auth-cache --trust-server-cert --config-dir c:\
VCS_OPTIONS=--non-interactive --no-auth-cache --trust-server-cert

# Workspace Initialization. Create new workspace equates to:
#   TFS: tf workspace -new -collection:${VCS_REPOSITORY_URL} ${VCS_WORKSPACE_NAME} -
noprompt /login:${VCS_USERNAME},${VCS_PASSWORD} ${VCS_OPTIONS}
${VCS_WORKSPACE_INIT_NEW_OPTIONS}
#       e.g. E:/dev/vcs/TEE-CLC-11.0.0/tf.cmd workspace -new -
collection:http://hostname:8080/tfs/DefaultCollection wks -noprompt /login:user,*****
/location:server
#   SVN: not applicable
#   P4: not applicable
#   CVS: not applicable
VCS_WORKSPACE_INIT_NEW_OPTIONS=/location:server

# Workspace Initialization. Link workspace to VCS repository equates to:
#   TFS: tf.cmd workfold -map -collection:{VCS_REPOSITORY_URL} ${TFS_SERVER_URL}
${VCS_WORKSPACE_DIR}+"/"+"${VCS_PROJECT_ROOT} -workspace:${VCS_WORKSPACE_NAME} -noprompt
/login:${VCS_USERNAME},${VCS_PASSWORD} ${VCS_OPTIONS} ${VCS_WORKSPACE_INIT_LINK_OPTIONS}
#       e.g. E:/dev/vcs/TEE-CLC-11.0.0/tf.cmd workfold -map -
collection:http://hostname:8080/tfs/DefaultCollection /Composite 62/cis objects
W:/wks/Composite 62/cis_objects -workspace:wks -noprompt /login:user,*****
#   SVN: svn import -m "linking workspace to the VCS repository" .
"${VCS_REPOSITORY_URL}/${VCS_PROJECT_ROOT}" ${SVN_OPTIONS} ${SVN_AUTH}
${VCS_WORKSPACE_INIT_LINK_OPTIONS}
#   P4: (UNIX) p4 client -o ${VCS_WORKSPACE_INIT_LINK_OPTIONS} | p4 client -i
${VCS_WORKSPACE_INIT_LINK_OPTIONS}
#   P4: (Windows) p4 client workspacename ${VCS_WORKSPACE_INIT_LINK_OPTIONS}
[manual intervention is required to acknowledge this action on windows only.]
#   CVS: cvs import -m "linking workspace to the VCS repository" ${VCS_PROJECT_ROOT}
INITIAL start ${VCS_WORKSPACE_INIT_LINK_OPTIONS}
VCS_WORKSPACE_INIT_LINK_OPTIONS=

# Workspace Initialization. Get resources from VCS repository equates to:
#   TFS: tf.cmd get -all -recursive ${TFS_SERVER_URL} -noprompt
/login:${VCS_USERNAME},${VCS_PASSWORD} ${VCS_OPTIONS} ${VCS_WORKSPACE_INIT_GET_OPTIONS}
#       e.g. E:/dev/vcs/TEE-CLC-11.0.0/tf.cmd get -all -recursive
/Composite 62/cis_objects -noprompt /login:user,*****
#   SVN: svn co "${VCS_REPOSITORY_URL}/${VCS_PROJECT_ROOT}" ${SVN_OPTIONS}
${SVN_AUTH} ${VCS_WORKSPACE_INIT_GET_OPTIONS}
#   P4: p4 sync ${VCS_WORKSPACE_INIT_GET_OPTIONS}

```

```

# CVS: cvs co ${VCS_PROJECT_ROOT} ${VCS_WORKSPACE_INIT_GET_OPTIONS}
VCS_WORKSPACE_INIT_GET_OPTIONS=

# VCS Base Folder Initialization. Add Options:
#
# TFS: tf.cmd add ${fullResourcePath} -recursive -noprompt
/login:${VCS_USERNAME},${VCS_PASSWORD} ${VCS_OPTIONS} ${VCS_BASE_FOLDER_INIT_ADD}
#
# e.g. E:/dev/vcs/TEE-CLC-11.0.0/tf.cmd add
P:/TFSww/Composite 62/cis objects/shared/test00 $/Composite 62/cis objects -noprompt
/login:user,*****
#
# SVN: svn add ${fullResourcePath} ${SVN_AUTH} ${VCS_OPTIONS}
${VCS_BASE_FOLDER_INIT_ADD}
#
# P4: p4 add ${fullResourcePath} ${VCS_BASE_FOLDER_INIT_ADD}
#
# CVS: cvs add ${fullResourcePath} ${VCS_BASE_FOLDER_INIT_ADD}
VCS_BASE_FOLDER_INIT_ADD=

# Resource Checkin. Checkin resources to VCS equates to:
#
# TFS:
#
# Folder: fullResourcePath: execFromDir+"/"+resourcePath"
#
# Check out folder for editing: tf.cmd checkout ${fullResourcePath} -
lock:Checkout -recursive -noprompt /login:${VCS_USERNAME},${VCS_PASSWORD} ${VCS_OPTIONS}
${VCS_CHECKOUT_OPTIONS}
#
# Check in folder: tf.cmd checkin ${fullResourcePath} -
comment:@${filename} -recursive -noprompt /login:${VCS_USERNAME},${VCS_PASSWORD}
${VCS_OPTIONS} ${VCS_CHECKIN_OPTIONS}
#
# File: fullResourcePath: execFromDir+"/"+resourcePath"+"_"+resourceType+".cmf"
#
# Check out file for editing: tf.cmd checkout ${fullResourcePath} -lock:Checkout
-noprompt /login:${VCS_USERNAME},${VCS_PASSWORD} ${VCS_OPTIONS} ${VCS_CHECKOUT_OPTIONS}
#
# Check in file: tf.cmd checkin ${fullResourcePath} -comment:@${filename}
-noprompt /login:${VCS_USERNAME},${VCS_PASSWORD} ${VCS_OPTIONS} ${VCS_CHECKIN_OPTIONS}
#
# SVN:
#
# Folder: fullResourcePath: execFromDir+"/"+resourcePath"
#
# Check in folder: svn commit ${fullResourcePath} -m "${Message}"
${SVN_AUTH} ${VCS_OPTIONS} ${VCS_CHECKIN_OPTIONS}
#
# File: fullResourcePath: execFromDir+"/"+resourcePath"+"_"+resourceType+".cmf"
#
# Check in file: svn commit ${fullResourcePath} -m
"${Message}" ${SVN_AUTH} ${VCS_OPTIONS} ${VCS_CHECKIN_OPTIONS}
#
# P4:
#
# Folder: fullResourcePath: execFromDir+"/"+resourcePath"
#
# Check in folder: p4 submit -d "${Message}" ${fullResourcePath}
${VCS_CHECKIN_OPTIONS}
#
# File: fullResourcePath: execFromDir+"/"+resourcePath"
#
# Check in file: p4 submit -d "${Message}"
${fullResourcePath} ${VCS_CHECKIN_OPTIONS}
#
# CVS:
#
# Folder: fullResourcePath: execFromDir+"/"+resourcePath"
#
# Check in folder: cvs commit -m "${Message}" ${fullResourcePath}
${VCS_CHECKIN_OPTIONS}
#
# File: fullResourcePath: execFromDir+"/"+resourcePath"+"_"+resourceType+".cmf"
#
# Check in file: cvs commit -m "${Message}" ${fullResourcePath}
${VCS_CHECKIN_OPTIONS}
#
# Check-in options are specific to the commit part of the command. For example, TFS
might be -associate:1 which associates a work item with a submission.
VCS_CHECKIN_OPTIONS=

# A comma separated list of base-level commands that are required for checkin.
VCS_CHECKIN_OPTIONS is validated against this list.
#
# For example, it may be required by TFS to have the -associate command present on the
check-in command line.

```

```

VCS_CHECKIN_OPTIONS_REQUIRED=

# Resource Checkout. Checkout resources to VCS equates to:
#
#   TFS:
#       Folder: fullResourcePath: execFromDir+"/"+resourcePath"
#       Check out folder: tf.cmd get ${fullResourcePath} -version:${Revision}
#       -recursive -noprompt /login:${VCS_USERNAME},${VCS_PASSWORD} ${VCS_OPTIONS}
#       ${VCS_CHECKOUT_OPTIONS}
#       File: fullResourcePath: execFromDir+"/"+resourcePath"+"_"+"resourceType+".cmf"
#       Check out file: tf.cmd get ${fullResourcePath} -version:${Revision}
#       -noprompt /login:${VCS_USERNAME},${VCS_PASSWORD} ${VCS_OPTIONS} ${VCS_CHECKOUT_OPTIONS}
#
#   SVN:
#       Folder: fullResourcePath: execFromDir+"/"+resourcePath"
#       Check out folder: svn update ${fullResourcePath} -r ${Revision}
#       ${SVN_AUTH} ${VCS_OPTIONS} ${VCS_CHECKOUT_OPTIONS}
#       File: fullResourcePath: execFromDir+"/"+resourcePath"+"_"+"resourceType+".cmf"
#       Check out file: svn update ${fullResourcePath} -r ${Revision}
#       ${SVN_AUTH} ${VCS_OPTIONS} ${VCS_CHECKOUT_OPTIONS}
#
#   P4:
#       Folder: fullResourcePath: execFromDir+"/"+resourcePath"
#       Check out folder: current: p4 sync ${VCS_CHECKOUT_OPTIONS}
#                               revision: p4 sync @${Revision}
#       ${VCS_CHECKOUT_OPTIONS}
#       File: fullResourcePath: execFromDir+"/"+resourcePath"+"_"+"resourceType+".cmf"
#       Check out file: current: p4 sync "${fullResourcePath}"
#       ${VCS_CHECKOUT_OPTIONS}
#                               revision: p4 sync "${fullResourcePath}@${Revision}"
#       ${VCS_CHECKOUT_OPTIONS}
#
#   CVS:
#       Folder: fullResourcePath: execFromDir+"/"+resourcePath"
#       Check out folder: cvs update -j${Revision} ${fullResourcePath}
#       ${VCS_CHECKOUT_OPTIONS}
#       File: fullResourcePath: execFromDir+"/"+resourcePath"+"_"+"resourceType+".cmf"
#       Check out file: cvs update -j${Revision} ${fullResourcePath}
#       ${VCS_CHECKOUT_OPTIONS}
VCS_CHECKOUT_OPTIONS=
# A comma separated list of base-level commands that are required for checkout.
VCS_CHECKOUT_OPTIONS is validated against this list.
VCS_CHECKOUT_OPTIONS_REQUIRED=
#-----
#
#-----

```

- ii. The VCS_REPOSITORY_URL is important to get correct. The URL is coupled with the VCS_PROJECT_ROOT which provides a pointer into the VCS Repository to store the CIS objects. The URL can point to any part of the VCS repository. There are two parts to keep in mind when considering your VCS repository. The URL and the folder within the repository that represents a the CIS “project” root folder. The CIS project folder will serve as a container for various CIS objects including /services, /shared and potentially /users. All of these folders are at the same level of the hierarchy. The base project URL may contain many project artifacts including other source code, documentation and project scripts. Additionally it should contain a Composite CIS object folder. Lastly, the URL should contain two sets of forward slashes after the http: portion of

the URL. The four slashes are converted to http:// during conversion of the URL. This is done to help preserve network path URL's when they are used. Let's consider an example:

5. **Base Repository URL:** <http://myhost.composite.com/svn/sandbox>

- a. Other project source code
- b. Documentation
- c. Project scripts
- d. **CIS Project Root Folder:** [cis_objects](#)
 - i. /services/databases
 - ii. /services/webservices
 - iii. /shared
 - iv. /users

```
# VCS_REPOSITORY_URL - This is the base URL to identify the VCS server.
# Note: The scripts use the combination of the VCS_REPOSITORY_URL and
# the VCS_PROJECT_ROOT to identify the baseline to checkin and checkout
# in the VCS. The VCS_PROJECT_ROOT also gets used in the folder structure
# of the local workspace.
#
# Subversion - The base HTTP URL in subversion
#
# Command Format: [http://hostname.domain/svn/basename]
#
# Example: http://myhost.composite.com/svn/sandbox
#
# Perforce - The Repository URL is the host and port in perforce -
#
# Command Format: [hostname:port]
#
# Example: myhost:1666
#
# CVS - Command Format:
# [:method:][user][:password]@hostname[:port]/repository_path
#
# 1) Local access only with no host and port
#
# Example: :local:/home/cvs
#
# 2) Remote access rules and examples:
#
# (1) There is no colon ":" following the port.
#
# (2) There is a colon ":" following the hostname if there is no port.
#
# (3) The repository folder path on the CVS server follows the port if
# present or hostname.
#
# (4) Username and Password are included in this URL therefore VCS_USERNAME
# and VCS_PASSWORD are ignored.
#
# Example: :pserver:user1:password@remotehost:2401/home/cvs
#
# Example: :pserver:user1:password@remotehost:/home/cvs
#
# Example: :pserver:user1@remotehost:/home/cvs
#
# 3) Use substitution variables to identify user and password. These
# variables get replaced at runtime with values passed in.
#
# Example:
# :pserver:<VCS_USERNAME>:<VCS_PASSWORD>@hostname:2401/home/cvs
#
```

```
# TFS - The base HTTP URL in Team Foundation Server
#
# Command Format: [http://hostname.domain:8080/tfs/basename]
#
# Example: http://myhost:8080/tfs/TeamCollection/TeamProject
#
VCS_REPOSITORY_URL=http://myhost.composite.com/svn/sandbox
#-----
#
#-----
# VCS_PROJECT_ROOT - This is root name of the project on the VCS Server
#
# Subversion - The project folder name
#
# Perforce - The depot folder name
#
# CVS - The project folder name
#
# TFS - The project folder name
VCS_PROJECT_ROOT=cis_objects
#-----
#
#-----
```

- iii. The VCS_WORKSPACE_HOME provides the base location of the workspace. If the user wants to locate the workspace in the same directory as the PDToolStudio then all simply set **VCS_WORKSPACE_HOME=\$PROJECT_HOME**. However, if the computer will be used for more than one user or the user simply wants to place the workspace in the user's working directory, then set **VCS_WORKSPACE_HOME=\$APPDATA**.

```
# VCS_WORKSPACE_HOME - This is the CIS VCS Workspace Home.
#
# It is recommended to set the location to PDToolStudio home [e.g. $PROJECT_HOME].
#
# The user does have the flexibility to place the VCS workspace in a location other
# than PDToolStudio home. [e.g. $APPDATA]
VCS_WORKSPACE_HOME=$PROJECT_HOME
#
# VCS_WORKSPACE_NAME:: The name of the workspace folder. This is not a directory but
# simply a name. The shorter the better.
#
# If running PDToolStudio on the same machine as PDTool then the workspace names
# should be different.
#
# Variables can be used to construct the name. Surround variables with 2 $ or 2 %
# signs when concatenating strings.
#
# e.g. $VCS_TYPE$sw - $VCS_TYPE$ gets evaluated as a variable. "sw" is a string
# that gets concatenated. Result: svns
#
# For perforce, make sure all instances of PDToolStudio/PDTool use their own
# workspace name in the event that you have them installed in more than one place.
#
# Suggestions: Use w=windows: [$VCS_TYPE$ww]. Use u for UNIX: [$VCS_TYPE$uw].
# Use s for studio: [$VCS_TYPE$sw].
#
# VCS_WORKSPACE_DIR:: VCS Workspace Dir is a combination of the VCS_WORKSPACE_HOME and
# a workspace directory name "VCS_WORKSPACE_NAME".
VCS_WORKSPACE_NAME=$VCS_TYPE$sw
```

```

VCS_WORKSPACE_DIR=$VCS_WORKSPACE_HOME/$VCS_WORKSPACE_NAME
#
#-----
#
#-----
# VCS_USERNAME - (optional) This is the username for the user logging into the VCS
Server.
#   If VCS_USERNAME is not set, then the specific VCS Server type may prompt the user
for a username and password each time.
#   Some VCS Servers, will ask to store the user and password locally for subsequent
use.
VCS_USERNAME=<your-username>
#-----
#
#-----
# VCS_PASSWORD - (optional) This is the password for the user logging into the VCS
Server.
#   If VCS_USERNAME is not set, VCS_PASSWORD is ignored.
#   If set in this file, execute the following command to encrypt the password:
#       Unix: ./ExecutePDToolStudio.sh -encrypt ../resources/config/studio.properties
#       Windows: ExecutePDToolStudio.bat -encrypt ../resources/config/studio.properties
VCS_PASSWORD=<your-password>
#-----
#
#

```

- iv. The VCS_IGNORE_MESSAGES provides a way to identify specific VCS messages that should be ignored when thrown so that the VCS Module does not throw an exception.

```

#-----
# VCS_IGNORE_MESSAGES - A comma separated list of messages for the VCS Module to ignore
upon execution.
#   CVS:
#   Perforce:      No files to submit
#   Subversion:
#   TFS:           No files checked in,could not be retrieved because a writable
file by the same name exists,already has pending changes,because it already has a
pending change that is not compatible,There are no remaining changes to check in
VCS_IGNORE_MESSAGES=No files to submit
#-----
#
#-----

```

- i. The VCS_MESSAGE_PREPEND provides a way to prepend a static message onto the message for Check in or Forced Check in.

```

#-----
# VCS_MESSAGE_PREPEND - A static message that gets prepended onto all check-in or forced
check-in messages. Some organizations require certain text to always be present in the
check in message. This allows that to occur.

```

```
VCS_MESSAGE_PREPEND=SCR:
```

```
#-----
```

```
#-----
```

d. Section: Specific VCS Environment Variables

- i. Depending on which VCS is being used (Subversion, Perforce or Concurrent Versions Systems) will dictate which section the user may want to modify. In reality, only SVN_EDITOR and P4EDITOR need to be changed to an editor. For Windows, the default editor in the path is "notepad". All other variables may remain the same.

```
#####
#### [SUBVERSION] USER MODIFIES [OPTIONAL] #####
# Subversion [svn] specific environment variables are set here
#####
# Subversion editor for messages
# [Default-change if desired but must be in path (UNIX: vi, Windows: notepad) ]
SVN_EDITOR=notepad
#
# SVN_ENV tells the system which SVN environment variables need to be set at execution
time
SVN_ENV=SVN_EDITOR
#-----
#
#####
#### [PERFORCE] USER MODIFIES [OPTIONAL] #####
# Perforce [p4] specific environment variables are set here
#####
# Perforce editor for messages
# [Default-change if desired but must be in path (UNIX: vi, Windows: notepad)]
P4EDITOR=notepad
# P4CLIENT must contain "exactly" the same folder name that is defined at the end of
VCS_WORKSPACE_DIR which is also VCS_WORKSPACE_NAME
# example: If [ VCS_WORKSPACE_DIR=$VCS_WORKSPACE_HOME/$VCS_TYPE$_workspace ] then
P4CLIENT=svn_workspace
P4CLIENT=$VCS_WORKSPACE_NAME
# example: set P4PORT=localhost:1666
# [Default-do not change]
P4PORT=$VCS_REPOSITORY_URL
# The environment must be set with the default username
# example: set P4USER=<VCS_USERNAME>
# example: set P4PASSWD=<VCS_PASSWORD>
# [Default-do not change] - Use substitution variables to identify user and password.
These variables get replaced at runtime with values passed in.
P4USER=<VCS_USERNAME>
```

```

P4PASSWD=<VCS_PASSWORD>

#

# Perforce Delete Workspace Link options
# Space separated list of options to pass into the command to delete the workspace link
between the file system and Perforce Depot repository.

# In the example below -f is shown as the optional option. If P4DEL_LINK_OPTIONS is
left blank then "p4 client -d ${VCS_WORKSPACE_NAME}" is executed.

# p4 client [-f] -d ${VCS_WORKSPACE_NAME}
P4DEL_LINK_OPTIONS=

#

# P4_ENV tells PD Tool which P4 environment variables need to be set at execution time
P4_ENV=P4CLIENT,P4PORT,P4USER,P4PASSWD,P4EDITOR,P4DEL_LINK_OPTIONS
#-----
#
#=====
#### [CVS] USER MODIFIES [OPTIONAL] ####
# Concurrent Versions System [cvs] specific environment variables are set here
#=====# Example: set
CVSROOT=:local:c:\dev\cvs\cvsrep
# [Default-do not change]
CVSROOT=$VCS_REPOSITORY_URL
# Example: Set the remote shell login when logging into a remote host
# [Default-do not change]
CVS_RSH=ssh
#
# CVS ENV tells the system which CVS environment variables need to be set at execution
time
CVS_ENV=CVSROOT,CVS_RSH
#-----
#
#=====
#### [TFS] USER MODIFIES [OPTIONAL] ####
# Team Foundation Server [tfs] specific environment variables are set here
#=====
# Subversion editor for messages
# [Default-change if desired but must be in path (UNIX: vi, Windows: notepad) ]
TFS_EDITOR=notepad
#
# TFS_ENV tells PD Tool which TFS environment variables need to be set at execution time
TFS_ENV=TFS_EDITOR
#
# TFS Server URL. Use $$ to escape the required beginning $
TFS_SERVER_URL=$$/<Team_Project_Name>/cis_objects
#-----

```

Step 4.1: Initialize VCS Workspace

1. Understand the Background.

This local workspace is important for the scripts to perform check-in and check-out operations against the VCS server. It contains a snapshot of the data stored and managed on the VCS server. Creates the necessary workspace folders in the local file system and synchronizes with the VCS server to checkout the CIS files into the local workspace.

There is no script modification required as it is driven by the settings in the “studio.properties” file. The script contains the commands for linking to the workspace folder and checking out artifacts. The specific VCS commands have been implemented within the “vcsInitWorkspace” method for the supported VCS platforms (Subversion, Perforce and CVS).

Workspace Environment Variables:

This local workspace can be installed anywhere on your computer. The key variables that determine the location are shown below and were configured in the previous step.

A quick reference is provided here for the environment variables:

```
#-----
# VCS_WORKSPACE_HOME - This is the CIS VCS Workspace Home.
#   It is recommended to set the location to PDToolStudio home [e.g. $PROJECT_HOME].
#   The user does have the flexibility to place the VCS workspace in a location other
#   than PDToolStudio home. [e.g. $APPDATA]
VCS_WORKSPACE_HOME=$APPDATA
#
# VCS_WORKSPACE_DIR:: VCS Workspace Directory is a combination of the VCS_WORKSPACE_HOME
# and a workspace directory name.
#   Variables can be used to construct the name. Surround variables with 2 $ or 2 %
# signs when concatenating strings.
#   e.g. $VCS_TYPE$cisVcsWorkspace - $VCS_TYPE$ gets evaluated as a variable.
#   _cisVcsWorkspace is a string that gets concatenated. Result: svn_cisVcsWorkspace
VCS_WORKSPACE_DIR=$VCS_WORKSPACE_HOME/$VCS_TYPE$cisVcsWorkspace
#-----
```

Example Locations:

- Note: PROJECT_HOME is automatically set by ExecutePDToolStudio script upon execution. Project home may be used as an environment variable within the studio.properties for placement relative to the project.
- Note: Use of operating system environment variables such as %APPDATA% are permitted and will be resolved at the time of execution. This allows the user to place the workspace in the users working directory. This is useful if multiple users

login to the same machine. The following shows an example where the user is “user1”.

Assumptions

- VCS_WORKSPACE_HOME= c:/Users/user1/AppData/Roaming
- Subversion
 - Workspace: c:/Users/user1/AppData/Roaming/PDToolStudio/[svnsw](#)
- Perforce
 - Workspace: c:/Users/user1/AppData/Roaming/PDToolStudio/[p4sw](#)
- CVS
 - Workspace c:/Users/user1/AppData/Roaming/PDToolStudio/[cvssw](#)
- TFS
 - Workspace c:/Users/user1/AppData/Roaming/PDToolStudio/[tfs2013sw](#)

2. Create the Workspace

The scripts are located in PDToolStudio/bin directory. Note: The default behavior is set to -vcsinit, therefore the user can simply double-click on ExecutePDToolStudio.bat to execute it.

Command Line Execution:

```
ExecutePDToolStudio.bat -vcsinit [-vcsuser user] [-vcspassword password]
```

Passwords:

As shown above, VCS username and passwords can be passed on the command-line for both command-line and Ant execution. Additionally, the password may be stored encrypted in studio.properties in the properties VCS_USERNAME and VCS_PASSWORD. Use the following commands to encrypt the password

```
ExecutePDToolStudio.bat -encrypt ../resources/config/studio.properties
```

Authentication with Subversion:

- Typically, the Subversion client saves the password somewhere on the local computer, so it will not prompt for it every time it is needed. At a client's site, this feature may be disabled. When you try to check in a file via Studio, it just hangs indefinitely with no errors thrown or logged. The remedy is to set the Subversion credential for VCS_USERNAME and VCS_PASSWORD. These credentials have been integrated into execution of the subversion commands via the VCS_OPTIONS environment variable. VCS_OPTIONS is automatically set during the command line execution: **--username user1 --password password**

3. Sample Results.

The example below is from an execution run using Subversion. Note...if there were no artifacts checked in, the checkout will not add any files to the workspace.

```
D:\dev\Workspaces\DeployToolWorkspace\PDTool\bin>ExecutePDToolStudio.bat -vcsinit

ExecutePDToolStudio::Tue 08/02/2011- 0:09:34.56::***** BEGIN COMMAND: ExecutePDToolStudio *****

-- COMMAND: vcsInitWorkspace -----

"C:\Program Files\Java\jdk1.6.0_25\bin\java" -cp
"D:\dev\Workspaces\DeployToolWorkspace\PDTool\dist\*;D:\dev\Workspaces\DeployToolWorkspace\PDTool\lib\*" -Dcom
.compositesw.ps.configroot="D:\dev\Workspaces\DeployToolWorkspace\PDTool\resources\config" -
Dlog4j.configuration="file:D:\dev\Workspaces\DeployToolWorkspace\PDTool\resources\config\log4j.
properties" ... vcsInitWorkspace " " "*****"

-- BEGIN OUTPUT -----

... [...] - <Config root D:\dev\Workspaces\DeployToolWorkspace\PDTool\resources\config>
... [...] - <Loading Sping Config File
D:\dev\Workspaces\DeployToolWorkspace\PDTool\resources\config\applicationContextList.xml>
... [...] - <Refreshing ... @4277158a: display name [... @4277158a]; startup date [Tue Aug 02
00:09:35 EDT 2011]; root of contexthierarchy>
... [...] - <Loading XML bean definitions from URL
[file:D:/dev/Workspaces/DeployToolWorkspace/PDTool/resources/config/applicationContextList.xml]
>

... [...] - <vcsInitWorkspace::----->
... [...] - <vcsInitWorkspace::Resolved Property Variables:>
... [...] - <vcsInitWorkspace::----->
... [...] - <vcsInitWorkspace::PROJECT_HOME= D:/dev/Workspaces/DeployToolWorkspace/PDTool/>
... [...] - <vcsInitWorkspace::CONFIG_HOME=
D:/dev/Workspaces/DeployToolWorkspace/PDTool/resources/config>
... [...] - <vcsInitWorkspace::PROPERTY_HOME=
D:/dev/Workspaces/DeployToolWorkspace/PDTool/resources/properties>
... [...] - <vcsInitWorkspace::MODULE_HOME=
D:/dev/Workspaces/DeployToolWorkspace/PDTool/resources/modules>
... [...] -
<vcsInitWorkspace::SCHEMA LOCATION=D:/dev/Workspaces/DeployToolWorkspace/PDTool/resources/schem
a/deployToolModules.xsd>
... [...] - <vcsInitWorkspace::DEBUG1= false>
... [...] - <vcsInitWorkspace::DEBUG2= false>
... [...] - <vcsInitWorkspace::DEBUG3= false>
... [...] - <vcsInitWorkspace::***** BEGIN COMMAND: vcsInitWorkspace *****>
... [...] - <vcsInitWorkspace:>
... [...] - <Config root D:\dev\Workspaces\DeployToolWorkspace\PDTool\resources\config>
... [...] - <Loading Sping Config File
D:\dev\Workspaces\DeployToolWorkspace\PDTool\resources\config\applicationContextList.xml>

... [...] - <initVCSWorkspace::----->
->
... [...] - <initVCSWorkspace::***** BEGIN VCS WORKSPACE INITIALIZATION *****>
... [...] - <initVCSWorkspace::----->
->
... [...] - <initVCSWorkspace::----->
... [...] - <initVCSWorkspace::Initialize workspace for VCS TYPE=svn>
... [...] - <initVCSWorkspace::----->
... [...] - <initVCSWorkspace:>
... [...] - <initVCSWorkspace::---VCS Input Variables from studio.properties file: >
... [...] - <initVCSWorkspace:: VCS_TYPE= svn>
... [...] - <initVCSWorkspace:: VCS_HOME= D:/dev/vcs/csvn/bin>
... [...] - <initVCSWorkspace:: VCS_COMMAND= svn>
... [...] - <initVCSWorkspace:: VCS_EXEC_FULL_PATH= true>
... [...] - <initVCSWorkspace:: VCS_OPTIONS= --non-interactive --no-auth-cache
--trust-server-cert --username mtinius --password *****>
... [...] - <initVCSWorkspace:: VCS_USER= mtinius>
... [...] - <initVCSWorkspace:: VCS_PASSWORD= *****>
```



```

... [...] - <initVCSWorkspace::      VCS_REPOSITORY_URL=
http://kauai.composite.com/svn/sandbox>
... [...] - <initVCSWorkspace::      VCS_PROJECT_ROOT=      cis objects>
... [...] - <initVCSWorkspace::      VCS_WORKSPACE_HOME=
D:/dev/Workspaces/DeployToolWorkspace/PDTool/>
... [...] - <initVCSWorkspace::      VCS_IGNORE_MESSAGES=    No files to submit>
... [...] - <initVCSWorkspace::>
... [...] - <initVCSWorkspace::---VCS Derived Variables:      >
... [...] - <initVCSWorkspace::      VCS_EXEC_COMMAND=      D:/dev/vcs/csvn/bin/svn>
... [...] - <initVCSWorkspace::      VCS Environment=      SVN EDITOR=notepad>
... [...] - <initVCSWorkspace::      VCS_WORKSPACE=
D:/dev/Workspaces/DeployToolWorkspace/PDTool/vcs_svn/cisVcsWorkspace>
... [...] - <initVCSWorkspace::      VCS_TEMP=      null
... [...] - <initVCSWorkspace::      VCS_WORKSPACE_PROJECT=
D:/dev/Workspaces/DeployToolWorkspace/PDTool/vcs_svn/cisVcsWorkspace/cis_objects>
... [...] - <initVCSWorkspace::      VCS LifecycleListener=
com.compositesw.cmdline.vcs.spi.svn.SVNLifecycleListener>
... [...] - <initVCSWorkspace::>
... [...] - <initVCSWorkspace::---VCS Static Variables:      >
... [...] - <initVCSWorkspace::      VCS_WORKSPACE_NAME=    cisVcsWorkspace>
... [...] - <initVCSWorkspace::>
... [...] - <initVCSWorkspace::      Linking local workspace to VCS Repository...>
... [...] - <initVCSWorkspace::      VCS Execute Command=D:/dev/vcs/csvn/bin/svn import .
http://kauai.composite.com/svn/sandbox/cis_objects --message
Linking workspace to VCS repository --non-interactive --no-auth-cache --trust-server-cert --
username mtinius --password *****>
... [...] - <initVCSWorkspace::      VCS Execute
Directory=D:/dev/Workspaces/DeployToolWorkspace/PDTool/vcs_svn/cisVcsWorkspace>
... [...] - <ScriptExecutor::----->
... [...] - <ScriptExecutor::Command: D:/dev/vcs/csvn/bin/svn import .
http://kauai.composite.com/svn/sandbox/cis_objects --message
Linking workspace to VCS repository --non-interactive --no-auth-cache --trust-server-cert --
username mtinius --password *****>
... [...] - <ScriptExecutor::Exec Dir:
D:/dev/Workspaces/DeployToolWorkspace/PDTool/vcs_svn/cisVcsWorkspace>
... [...] - <ScriptExecutor::Env Var: SVN_EDITOR=notepad>
... [...] - <ScriptExecutor::----->
... [...] - <Successfully executed command=D:/dev/vcs/csvn/bin/svn Output=>
... [...] - <initVCSWorkspace::      Checking out CIS objects from...>
... [...] - <initVCSWorkspace::      VCS Execute Command=D:/dev/vcs/csvn/bin/svn co
http://kauai.composite.com/svn/sandbox/cis_objects --non-interactive --no-auth-cache --trust-
server-cert --username mtinius --password *****>
... [...] - <initVCSWorkspace::      VCS Execute
Directory=D:/dev/Workspaces/DeployToolWorkspace/PDTool/vcs_svn/cisVcsWorkspace>
... [...] - <ScriptExecutor::----->
... [...] - <ScriptExecutor::Command: D:/dev/vcs/csvn/bin/svn co
http://kauai.composite.com/svn/sandbox/cis_objects --non-interactive --no-auth-cache --trust-
server-cert --username mtinius --password *****>
... [...] - <ScriptExecutor::Exec Dir:
D:/dev/Workspaces/DeployToolWorkspace/PDTool/vcs_svn/cisVcsWorkspace>
... [...] - <ScriptExecutor::Env Var: SVN_EDITOR=notepad>
... [...] - <ScriptExecutor::----->
... [...] - <Successfully executed command=D:/dev/vcs/csvn/bin/svn Output=A
cis_objects\services
A    cis_objects\services\webservices

...code removed

A    cis_objects\shared\examples\productCatalog_Transformation_procedure.cmf
A    cis_objects\shared\examples\ViewSales table.cmf
A    cis_objects\cis_objects
Checked out revision 86.
>
... [...] - <initVCSWorkspace::----->
... [...] - <initVCSWorkspace::Successfully Initialized workspace for VCS_TYPE=svn>
... [...] - <initVCSWorkspace::----->
... [...] - <vcsInitWorkspace::Successfully completed vcsInitWorkspace.>
... [...] - <vcsInitWorkspace::>

```

```
ExecutePDToolStudio::Tue 08/02/2011- 0:09:49.82::----- SUCCESSFUL SCRIPT COMPLETION
[ExecutePDToolStudio -vcsinit] -----
ExecutePDToolStudio::Tue 08/02/2011- 0:09:49.84::End of script.

D:\dev\Workspaces\DeployToolWorkspace\PDTool\bin>
```

Step 4.2: Initialize VCS Base Folders (VCS Administrator)

1. Initialize the Base Folders.

Important: This step is only done once by an administrator under these circumstances:

- a. The first time a VCS repository is initialized. Not the workspace....but the actual VCS repository.
- b. Each time a tenant path is required to be placed under version control.
- c. Each intermediate path leading up to the actual folders needing to be placed under version control.

This section describes the process of initializing the VCS repository with the Composite repository base folders. This is very important as it takes the burden off of the PDTool Studio users to perform this function. It also affords the opportunity to address Composite multi-tenant environments where not all tenants will be using version control. Imagine a Composite shared server with 800,000 to 1 million resources in development which is not uncommon in a large installation. Trying to do an initial check-in of all these objects will take an inordinate amount of time. The following strategy will demonstrate a) show a break-down the base-level folders, b) show how to check-in the base level folders, and c) show how to add intermediate folders to the VCS repository.

a. Break-down of base-level folders

The base structure with no resources in composite is shown below. The folder “cis_objects” represents a container in the workspace and VCS in which to hold the multiple base-level folders. This is a good best practice. Note that each folder contains the name of the folder with a .cmf extension. A .cmf file is an XML file representation of a Composite resource (all resources). Also notice that root (/) is designated by root.cmf and is a file in the root (/) folder.

```

/cis_objects
  /policy
    /security
      /user
        user.cmf
      security.cmf
    policy.cmf
  /security
    /rowlevel
```

```

        /filters
            filters.cmf
            rowlevel.cmf
        security.cmf
    /services
        /databases
            databases.cmf
        /webservices
            webservices.cmf
        services.cmf
    /shared
        shared.cmf
    /system
        /connector
            connector.cmf
        system.cmf
    /users
        /composite
            /admin
                admin.cmf
            composite.cmf
        users.cmf
    root.cmf

```

b. How to check-in base-level folders

There is a new option for ExecutePDToolStudio.bat that is used to initialize the base folders. By invoking the “-vcsinitBaseFolders” option directly after the workspace initialization, it allows the administrator to establish all of the base-level folders in the VCS repository without actually checking in the entire Composite repository at one time.

Option 2 - Execute VCS Base Folder initialization:

```
ExecutePDToolStudio.bat [-nopause] -vcsinitBaseFolders [-customCisPathList
custom-CIS-path-list] [-vcsuser vcs-username] [-vcspassword vcs-password]
```

arg1:: [-nopause] is an optional parameter used to execute the batch file without pausing at the end of the script.

arg2:: -vcsinitBaseFolders is used to initialize the vcs repository with the Composite repository base folders and custom folders.

arg3:: [-customCisPathList custom-CIS-path-list] optional parameter. Custom, comma separated list of CIS paths to add to the VCS repository.

arg4:: [-vcsuser vcs-username] optional parameter

arg5:: [-vcspassword vcs-password] optional parameter

Command Line Execution:

```
ExecutePDToolStudio.bat - vcsinitBaseFolders [-customCisPathList  
"custom-CIS-path-list"] [-vcsuser user] [-vcspassword password]
```

c. How to add intermediate folders

The same option for ExecutePDToolStudio.bat referenced above is also used to add intermediate folders from the base-level folders. This allows the administrator to prime the VCS repository with the necessary Composite folders leading up the actual folders where the users are doing work.

This flexibility allows the administrator to prepare a shared environment quickly. The scenarios that come into play for this feature are multi-tenant environments where version control is only selectively used or single-tenant environments where only certain folders will be governed under version control.

Observe the base-level folders shown above and then envision this example where additional folders are to be created where there are 3 tenants but only 1 tenant will actually use version control.

SCENARIO: Given the following multi-tenant folder structure in Composite, the administrator wants to initialize the VCS repository for Tenant2 only and create the “intermediate” folders.

```
/services
  /databases
    /ORG1
      /T1CAT
      /T1SCH
      /T2CAT
      /T2SCH
    /ORG2
      /T3CAT
  /webservices
    /ORG1
      /Tenant1
      /Tenant2
    /ORG2
      /Tenant3
  /shared
    /ORG1
      /Tenant1
      /Tenant2
    /ORG2
      /Tenant3
```

Command Line Execution:

```
ExecutePDToolStudio.bat -vcsinitBaseFolders -customCisPathList  
"/shared/ORG1/Tenant2,/services/webservices/ORG1/Tenant2,/services/databas
```

```
es/ORG1/T2CAT<TYPE=CATALOG>/T2SCH<TYPE=SCHEMA>" -vcsuser user -vcspassword password
```

In the above example the comma separated list of folders would be as follows (all on a single line):

```
/shared/ORG1/Tenant2,  
/services/webservices/ORG1/Tenant2,  
/services/databases/ORG1/T2CAT<TYPE=CATALOG>/T2SCH<TYPE=SCHEMA>
```

The /services/databases/ORG1 path contains two mandatory designators which tell PDTool which path part is a catalog and which one is a schema. This is required because the position after the database name may be either a catalog or schema. The designator <TYPE=CATALOG> tells PDTool that the path part is a catalog. The designator <TYPE=SCHEMA> tells PDTool that the path part is a schema.

Step 5: Enable VCS in Studio

In this section, we enable Studio to use the modified backend scripts and the local workspace to communicate with the VCS server. There are two ways to accomplish this configuration: **manually** and **automated**.

Important-1!! Unlike previous instructions, this section is CIS instance specific. When Studio is enabled, it is only enabled for the CIS instance (DEV, TEST, etc.) to which it is currently connected. If Studio is closed and re-launched, but connected to a different CIS instance, it will not be enabled. You would have to enable it again, probably with different values. In practice, we typically only enable Studio for the DEV environment.

Important-2!! If a computer is to be used by multiple users logging in then both the PD Tool Studio scripts and workspace must be installed into the %APPDATA% directory where the user's data is kept. This will be explained in the section below.

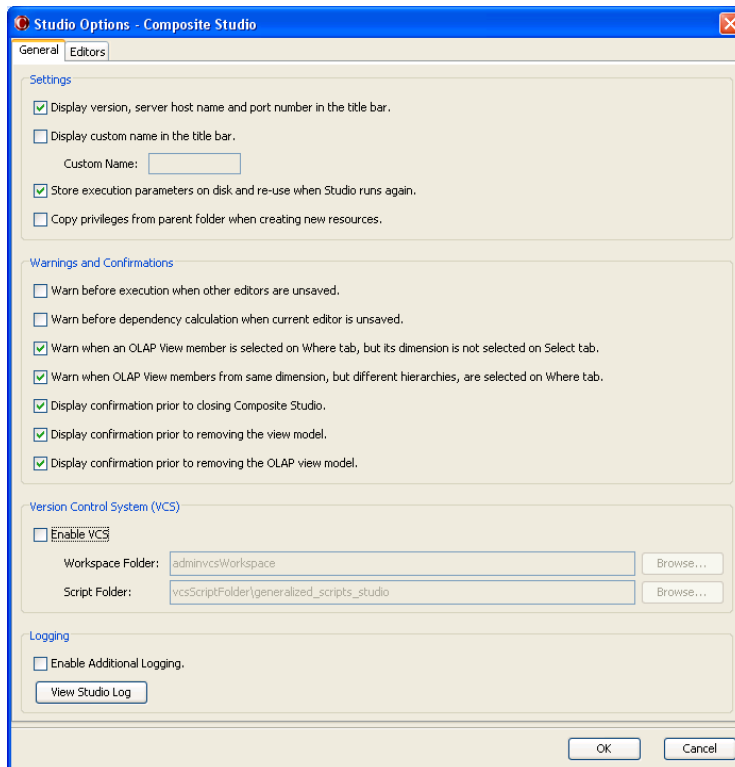
MANUAL CONFIGURATION TO ENABLE VCS IN STUDIO

1. Launch Composite Studio

- a. Connect to your Development Server

2. Select the menu option "Edit → Studio Options"

The following window will appear. The window displayed may look different for CIS 5.1



3. Check “**Enable VCS**” box and provide the following parameters:

These values are user specific, so be sure to use your own values.

- a. For **VCS Workspace Folder**, browse for and select the workspace folder created in Step 4 above. For example:

<CIS install directory>\conf\studio\PDToolStudio\<VCS_Type>sw \<VCS Project Root>

The Workspace folder can be in a separate folder from where the scripts are installed. Additionally, the %VCS_TYPE% may have been affixed to the workspace so as to provide the user with an easy to understand indicator of the type of VCS being used. Finally, the workspace folder contains the VCS project that was checked out during initialization and is indicated by VCS Project Root. By reviewing the studio.properties file, the user can resolve the VCS_TYPE and VCS_PROJECT_ROOT. The actual creation of the workspace was created and populated during the VCS initialization step performed earlier using the parameters provided in studio.properties. The following example will use Subversion (svn) and a project folder of cis_objects. It demonstrates the placement of the workspace within the PDToolStudio project folder:

<CIS install directory>\conf\studio\PDToolStudio\svns\cis_objects

However, if the computer will be used for multiple user logins, then both the Scripts and Workspace need to be placed in the %APPDATA% directory as defined

earlier. For example, users might want to store the workspace in %APPDATA% which will resolve to the user's AppData directory. For example, when using Subversion (svn) and a project folder of cis_objects, the Workspace folder would look like this:

C:\Users\<user>\AppData\Roaming\svn_cis\VcsWorkspace\cis_objects

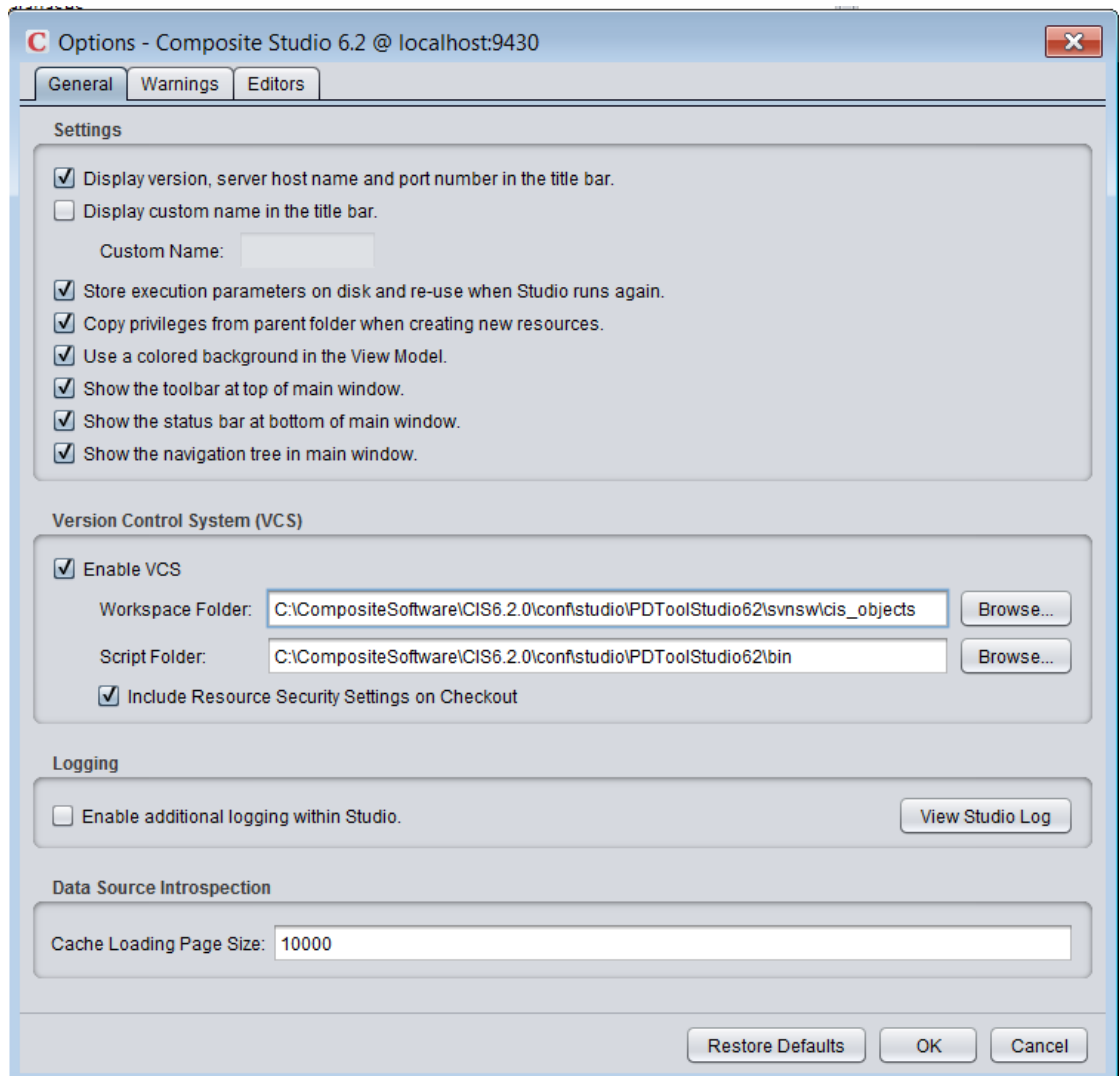
- b. For **Script Folder**, browse for and select the folder that contains the extracted and edited scripts for Composite Studio. It is perfectly acceptable to use any directory of your choosing for the scripts. As an example, the default is to use Composite Studio installation directory to manage version control, whereby you would specify:

<CIS install directory>\conf\studio\PDToolStudio\bin

However, if the computer will be used for multiple user logins, then both the Scripts and Workspace need to be placed in the %APPDATA% directory as defined earlier. During the installation step the scripts would have been installed in the AppData directory. For example, the PDToolStudio scripts for a user would be found at this location:

C:\Users\<user>\AppData\Roaming\PDToolStudio\bin

- c. For **Include Resource Security Settings on Checkout**, you can optionally check that box if you have MODIFY_ALL_RESOURCES and MODIFY_ALL_USERS rights. If selected, the privileges of the resource being checked out are included. If left clear, the privileges of the resource remain unchanged or have the default settings applied. If the resource existed in CIS then the privilege information is unchanged. If the resource did not already exist, default privilege settings are applied to it. If you receive a security violation exception, you do not have the necessary MODIFY_ALL_RESOURCES and MODIFY_ALL_USERS rights.



- d. **Click OK to save and exit.**

Studio has been enabled to communicate with the VCS Server.

AUTOMATED CONFIGURATION TO ENABLE VCS IN STUDIO

1. **Enable Studio VCS** (background)

The scripts are located in PDToolStudio/bin directory. Note: The ExecutePDToolStudio.bat script may be used to enable VCS within Composite Studio. Each time Composite Studio closes it writes the Composite Studio Enable VCS property file. The property file name is of the format <composite_user>.<domain>.<composite_server_host>.properties. In Windows 7 and 2008, the file is written to the user's home directory: C:\Users\<user>\.compositesw\<composite_user>.<domain>.<composite_server_host>.properties. For example, the user LDAP user mike1000 who is a member of corp_domain

logs into the corp_dev_host server would have the following file:

C:/Users/mike1000/.compositesw/mike1000.corp_domain.corp_dev_host.com.properties

The break-down of the file name is constructed as follows:

- <composite_user> – this is the user that is used to log into Composite. It can be a user for the composite domain or LDAP domain.
- <domain> – this is the domain that is specified in the login screen. The default domain is “composite”. If LDAP is configured, then it will be the name of LDAP configured in Composite.
- <composite_server_host> – this is the host name. This can be found in Composite Studio by accessing Administration → Composite Server → Configuration → General Information → Host Name. The value in the Host Name field will contain the base server name.
- .properties – the file extension

2. Close all instances of Studio

Composite Studio must be closed.

3. Run ExecutePDToolStudio.bat

Create the Composite Studio Enable VCS property file. The property file name is of the format <composite_user>.<domain>.<composite_server_host>.properties.

arg1:: **-nopause** specifies to not pause at the end of the script. Automated script would want to use this option.

arg2:: **-enablevcs** specifies the command

arg3:: **-user** is the user name that you login to Composite Studio with.

arg4:: **-domain** is the domain designated in the Composite Studio login.

arg5:: **-host** is the host name designated in the Composite Studio login.

arg6:: **-includeResourceSecurity** is true or false and designates whether to turn on resource security at checkin.

arg5:: [optional] **-vcsWorkspacePathOverride** is a way of overriding the VCS workspace path instead of allowing this method to construct from the studio.properties file properties: VCS_WORKSPACE_DIR+"/"+VCS_PROJECT_ROOT. It will use the substitute drive by default. This parameter is optional and therefore can be left out if you want to use the default settings in studio.properties. If a path is provided, use double quotes to surround the path.

Command Line Execution:

```
ExecutePDToolStudio.bat -nopause -enablevcs -user [studio_login_user]
-domain [composite_domain] -host [composite_host_server] -
includeResourceSecurity [true|false] -vcsWorkspacePathOverride "[path-
to-workspace-cis-objects]"
```

Command Line Example:

```
ExecutePDToolStudio.bat -nopause -enablevcs -user mike -domain  
corp_ldap -host corp_server.corp_domain.com -includeResourceSecurity  
true -vcsWorkspacePathOverride  
"C:/PDToolStudio62/svn_sworkspace/cis_objects"
```

Example of Enable VCS property file:

Generated by ExecutePDToolStudio.bat

2013-04-15 12:00:00.000

vcsIncludeResourceSecurity=true

enableVCS=true

vcsWorkspacePath=

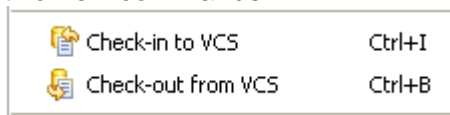
D:\\CompositeSoftware\\CIS6.2.0\\conf\\studio\\conf\\studio\\PDToolStudio62\\svns\\c
is_objects

vcsScriptFolder=D:\\CompositeSoftware\\CIS6.2.0\\conf\\studio\\PDToolStudio62\\bin

Step 6: Test VCS

In this section, we will test the Studio/VCS Server integration with some check-in and check-out operations.

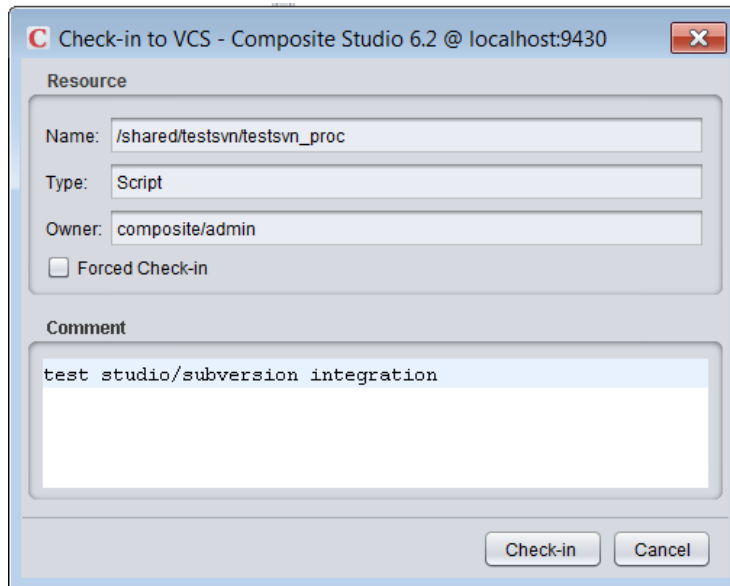
When VCS is enabled in Studio, new commands are added to the Studio interface. When you select a resource in the resource tree, the Resource menu and right click menu show two new commands:



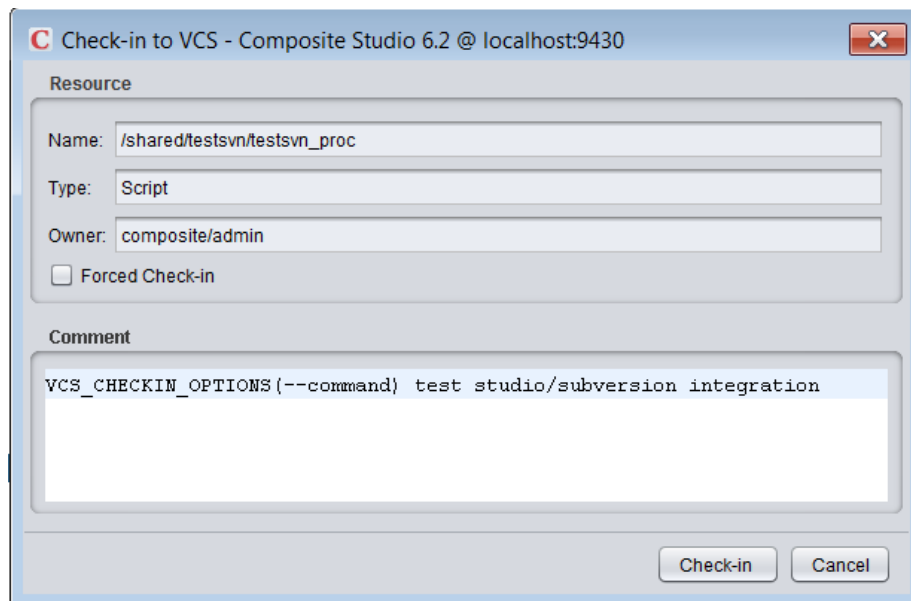
2. Check-in

- a. **Note:** If this is the first time that CIS objects have ever been checked into this VCS repository project, it is recommended that “/shared” be checked in first so that the repository is populated with the entire CIS structure. If there are a lot of objects and the tree is deep, this may take a while to complete. Be patient. For example, a tree containing approximately 9000 objects may take up to 20 minutes to check-in the first time.
- b. In Studio, under “/shared”, create a folder called “testvcs”. In this folder, create a SQL Script procedure called “testsvn_proc”. There is no need to put any code in it. Just save and close it.
- c. Right-click on “/shared/testsvn/testsvn_proc” and select “Check-in to VCS”. The following window will appear. Enter a comment to show this is a test object.

Option 1: don't inject a checkin command. Just put in a comment:

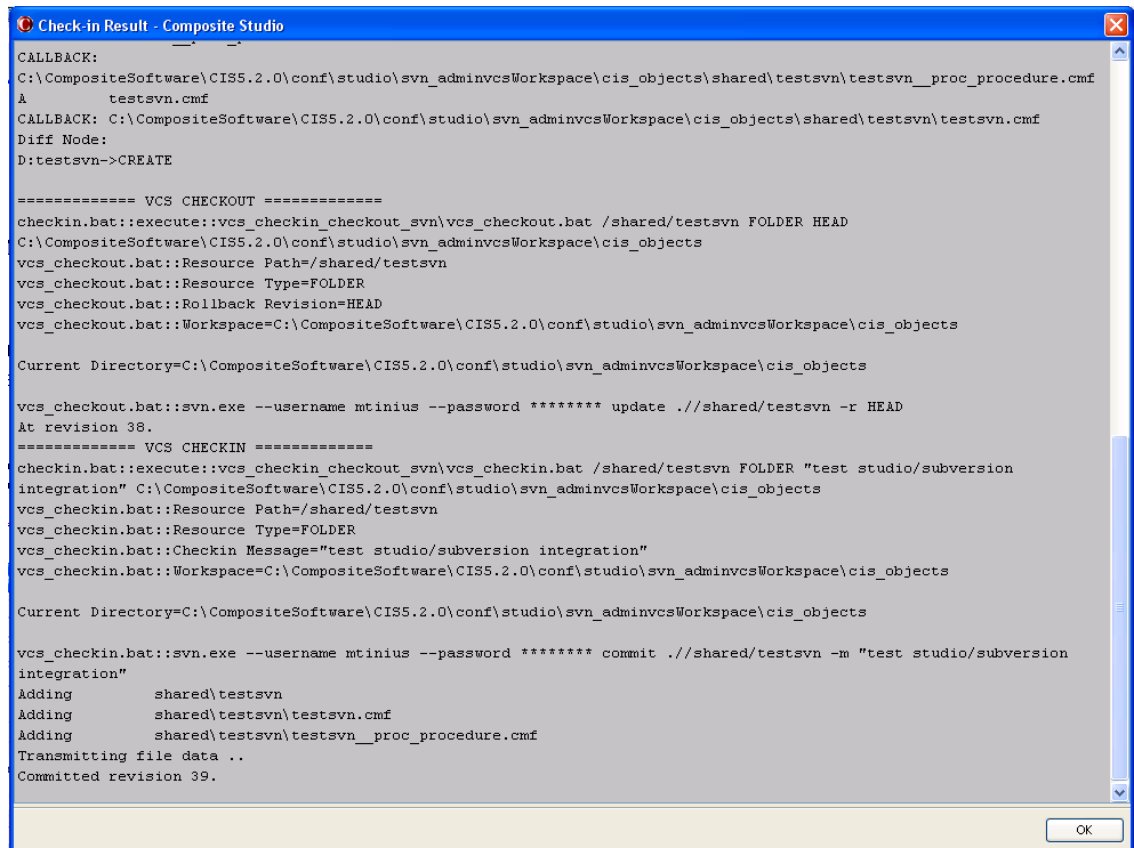


Option 2: inject a checkin command using the template
VCS_CHECKIN_OPTIONS(<commands>) my comment:



- d. Click "Check-in" to get the next window. It will show that the VCS Server has accepted the check-in. In VCS terminology, this change has been committed to the server.
 - i. If you chose to inject a VCS command line option then that will be included in the output. That command is not included in the comments that are passed to the VCS. If the command is not a valid command, the VCS will throw an error which will be displayed in the output below.

- ii. The studio.properties file also contains a property VCS_CHECKIN_OPTIONS_REQUIRED which is where you can require the developers to add particular commands to the comment message using the facility provided. The commands are comma separated so that you can require multiple commands. The commands should represent the base part of the command.



```
CALLBACK:
C:\CompositeSoftware\CISS.2.0\conf\studio\svn_adminvcsWorkspace\cis_objects\shared\testsvn\testsvn_proc_procedure.cmf
A
testsvn.cmf
CALLBACK: C:\CompositeSoftware\CISS.2.0\conf\studio\svn_adminvcsWorkspace\cis_objects\shared\testsvn\testsvn.cmf
Diff Node:
D:testsvn->CREATE

===== VCS CHECKOUT =====
checkin.bat::execute::vcs_checkout_checkout_svn\svn_checkout.bat /shared/testsvn FOLDER HEAD
C:\CompositeSoftware\CISS.2.0\conf\studio\svn_adminvcsWorkspace\cis_objects
vcs_checkout.bat::Resource Path=/shared/testsvn
vcs_checkout.bat::Resource Type=FOLDER
vcs_checkout.bat::Rollback Revision=HEAD
vcs_checkout.bat::Workspace=C:\CompositeSoftware\CISS.2.0\conf\studio\svn_adminvcsWorkspace\cis_objects

Current Directory=C:\CompositeSoftware\CISS.2.0\conf\studio\svn_adminvcsWorkspace\cis_objects

vcs_checkout.bat::svn.exe --username mtinius --password ***** update ../shared/testsvn -r HEAD
At revision 38.
===== VCS CHECKIN =====
checkin.bat::execute::vcs_checkout_checkout_svn\svn_checkout.bat /shared/testsvn FOLDER "test studio/subversion
integration" C:\CompositeSoftware\CISS.2.0\conf\studio\svn_adminvcsWorkspace\cis_objects
vcs_checkout.bat::Resource Path=/shared/testsvn
vcs_checkout.bat::Resource Type=FOLDER
vcs_checkout.bat::Checkin Message="test studio/subversion integration"
vcs_checkout.bat::Workspace=C:\CompositeSoftware\CISS.2.0\conf\studio\svn_adminvcsWorkspace\cis_objects

Current Directory=C:\CompositeSoftware\CISS.2.0\conf\studio\svn_adminvcsWorkspace\cis_objects

vcs_checkout.bat::svn.exe --username mtinius --password ***** commit ../shared/testsvn -m "test studio/subversion
integration"
Adding      shared\testsvn
Adding      shared\testsvn\testsvn.cmf
Adding      shared\testsvn\testsvn_proc_procedure.cmf
Transmitting file data ..
Committed revision 39.
```

- e. To verify the check-in operation, open the VCS Browser. This may be a URL or an installed viewer that allows the user to see the repository contents.

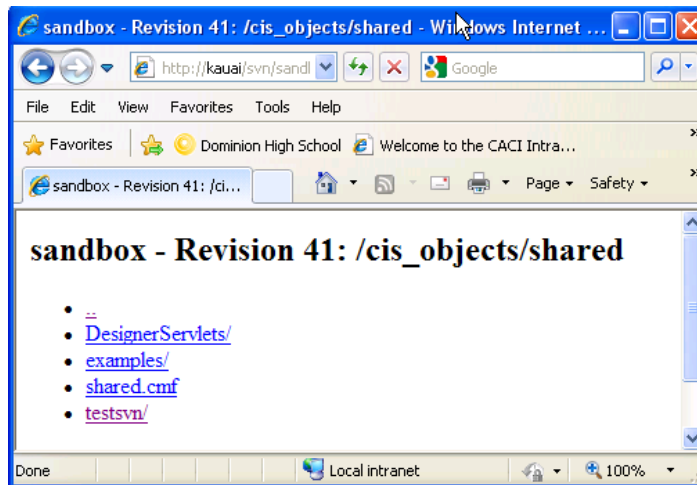
For example with Subversion, open a browser and go to the following URL.

http://<subversion_server>:<subversion_port>/svn/<repository>/<project>

For example...

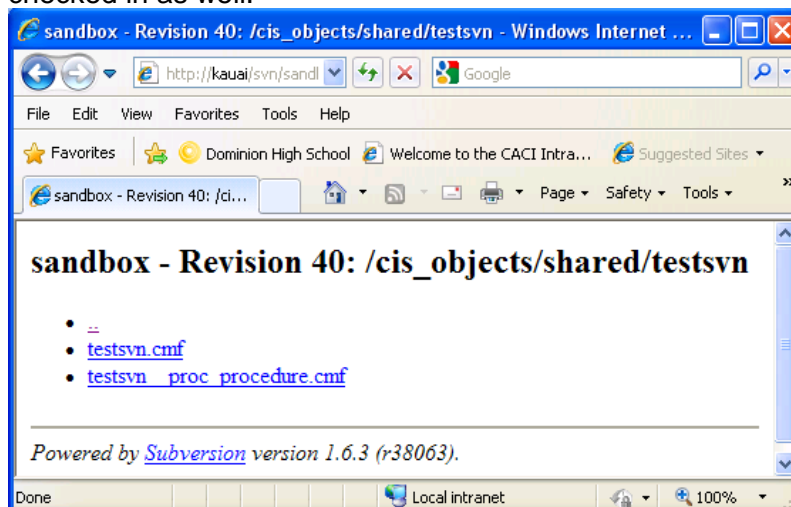
http://host.domain.com/svn/sandbox/cis_objects

This page should display at least three folders as shown below. It may show additional folders.



- f. Navigate down by clicking “shared” followed by “testsvn” and you should land on the following page.

Note: The file called “testsvn__proc_procedure.cmf”. This file contains the source codes from the procedure we just checked in. By seeing this procedure on the Subversion server, we have verified that it has been checked in successfully. Note that when the procedure object was checked in, its parent folder was automatically checked in as well.



You are not limited to checking in objects at the leaf level (no child objects). You can also check in a folder that contains many subfolders and objects. All of its child objects will be checked-in in one step.

3. Check-out

Before we check out the procedure that was just checked in, we will delete it from Studio to demonstrate that we can repopulate our Studio instance from the VCS Server.

- a. **CAUTION:** The process of checking out resources from the VCS repository will perform a diffmerger between what is currently in CIS and what is in the VCS Repository. The result of the diffmerger may result in folders being deleted in CIS that are not present in the VCS Repository. To illustrate this point, the following example will be used:

- i. **VCS Repository: .../sandbox/cis_objects**

- 1. /shared/project1
 - 2. /shared/testsvn (checked in earlier)

- ii. **CIS Repository:**

- 1. /shared/project2
 - 2. /shared/testsvn

In this example, if the user checks out at the top-level /shared folder level from the VCS Repository, the diffmerger will determine that /shared/project1 does not exist in CIS. It will determine that /shared/project2 is not in the VCS Repository and mark it for deletion. Finally, it will determine that /shared/testsvn exists in both the VCS Repository and the CIS Repository and is up to date. This process of updating the local workspace will result in a .CAR file being created that will upon import add /shared/project1 to the CIS Repository and delete /shared/project2. For those who are new to VCS, it may come as a surprise that /shared/project2 is deleted upon import. However, this is the correct behavior.

- b. **BEST PRACTICE:** Because of the points mentioned earlier regarding the possibility of deleting folders, the best practice for being able to check out folders structures from VCS into the user's CIS Repository will be illustrated below using an example. Notice that in this example the folder /shared/testsvn does not exist in the CIS Repository. However, the user would like to check out that folder structure. As we have learned previously, if the user were to check out at the /shared level, then they would lose their /shared/project2 which is not desirable. Let us assume that the user has the ability to browse the VCS Repository to find /shared/testsvn or has been instructed by a colleague that /shared/testsvn exists and they should check it out. Therefore, the best practice for the user is to manually create /shared/testsvn using Studio. Then right-mouse click on the folder testsvn and check out. This will allow the user to pin-point just the testsvn folder while preserving other project folders that are siblings to testsvn.

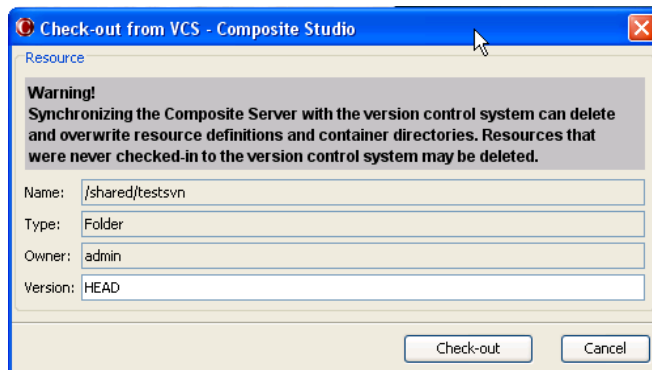
- i. **VCS Repository: .../sandbox/cis_objects**

- 1. /shared/project1
 - 2. /shared/testsvn (checked in earlier)

- ii. **CIS Repository:**

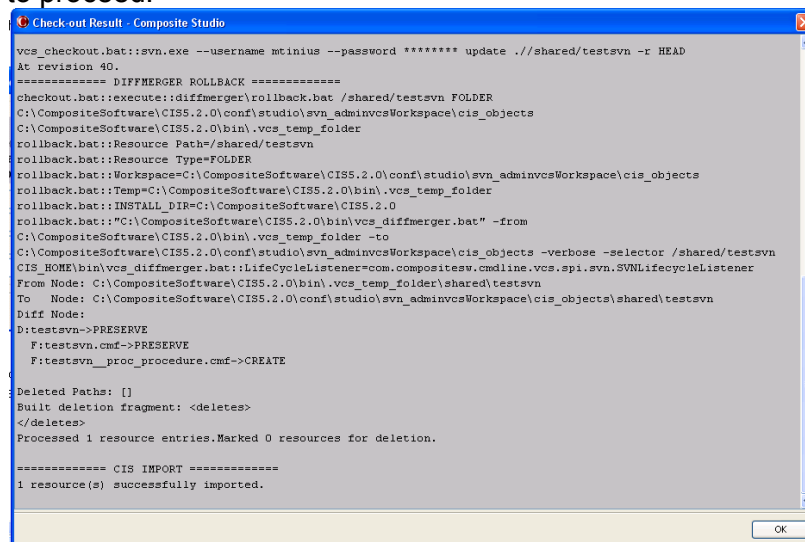
- 1. /shared/project2

- c. In Studio, right-click on "/shared/testsvn" and select "Check-out from VCS". The following window will appear.



- d. In the “Version” field, enter “HEAD” because we want to check out the latest version (called revision in Subversion).

Note: If you need to check out a previous version, enter its revision number such as “10”. The revision number is available via the VCS Server log. Click “Check-out” to proceed.



The check-out has been completed. The testsvn_proc procedure reappears within the Studio testsvn folder. The Studio/VCS Server integration has been completed.

4. Rolling Back to a Previous Version in Studio

To roll back to a previous version in Studio:

1. Check out a specific version as described above.
2. Check in using the Forced Check-in checkbox as described in the check in steps above.

EXCEPTIONS AND MESSAGES

The following are common exceptions and messages that may occur.

1. Repository moved temporarily to...:

D:/dev/vcs/csvn/bin/svn Execution Returned an Error=svn: Repository moved temporarily to 'http://kauai.composite.com/svn/sandbox/cis_objects/shared/'; please relocate
Resolution: Connect to the internet or vpn

2. Commit failed...not under version control:

Caused by: com.compositesw.ps.common.exception.CompositeException: /usr/bin/svn Execution Returned an Error=svn: Commit failed (details follow):
svn: '/u01/opt/DeployTool/PDToolStudio/vcs_svn/cisVcsWorkspace/cis_objects/testNN'" is not under version control
Resolution: run VCS workspace initialization (ExecutePDToolStudio -initvcs user password

3. Commit failed...file 'x' remains in conflict:

2011-07-29 08:02:21,330 main INFO [...] - <CisDeployTool::Abnormal Script Termination. Script will exit. ERROR=com.compositesw.ps.common.exception.CompositeException: /usr/bin/svn Execution Returned an Error=svn: Commit failed (details follow):
svn: Aborting commit:
'/u01/opt/DeployTool/CisDeployTool/vcs_svn/cisVcsWorkspace/cis_objects/shared/test00/ResourceCache/testCacheProc_procedure.cmf' remains in conflict
Background: This scenario can occur when you are running in a Multi-User Topology and each user has the ability to check-in resources to the shared VCS repository. This variant is called "Multi-User (Direct VCS Access). Insure that "setVCS.bat" has been configured for set VCS MULTI_USER_TOPOLOGY=true. Re-initialize the VCS workspace.
Resolution: run VCS workspace initialization (ExecutePDToolStudio -initvcs user password

4. Access is denied:

This may occur during "checkin" or "forced checkin" in a Windows 7 environment.
java.lang.RuntimeException: Encountered problem during vcs export:
C:\Program Files\Composite Software\CIS 5.2.0\bin\.vcs_temp_folder\vcs_export.zip (Access is denied)
Resolution: Try starting Studio "As Administrator" and run the checkin or forced checkin again.

VCS SPECIFIC INFORMATION

Subversion specific information

General Notes: This section contains general notes about Subversion and PDTTool.

1. Authentication with subversion server

This section provides background on how authentication is handled with Subversion. There is nothing the user needs to change with respect to PD Tool Studio. There are no scripts to modify.

Typically, the Subversion client saves the password somewhere on the local computer, so it will not prompt for it every time it is needed. At a client's site, this feature may be disabled. When you try to check in a file via Studio, it just hangs indefinitely with no errors thrown or logged. The remedy is to set the Subversion credential for VCS_USERNAME and VCS_PASSWORD in the studio.properties file and then encrypt the passwords. These credentials have been integrated into the backend scripts via the environment variable SVN_OPTIONS

The debug output will show the use of SVN_OPTIONS from studio.properties by incorporating the subversion syntax for the user name and password.

SVN_OPTIONS=--username user1 --password password.

For security purposes, the debug print-out will use "*****" for password display. Note that the argument name uses two dashes instead of one.

vcs_checkin:

```
:FOLDER
svn.exe commit ./%ResPath% -m %Message% %SVN_OPTIONS%
[code removed...]
:FILE
svn.exe commit ./%ResPath%_%ResType%.cmf -m %Message% %SVN_OPTIONS%
```

vcs_checkout:

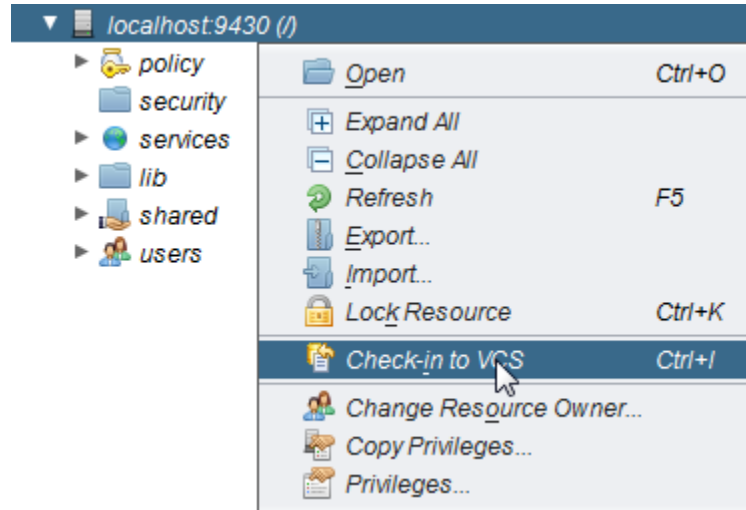
```
:FOLDER
svn.exe update ./%ResPath% -r %Revision% %SVN_OPTIONS%
[code removed...]
:FILE
svn.exe update ./%ResPath%_%ResType%.cmf -r %Revision% %SVN_OPTION
```

Perforce specific information

General Notes: This section contains general notes about Perforce and PDTTool.

1. Running PDTTool and PDTStudio on the same machine.
 - a. Make sure the workspace name "VCS_WORKSPACE_NAME" is configured with a different name such as p4_wworkspace for PDTTool and p4_sworkspace for PDTToolStudio.
2. When to initialize the local workspace directory.
 - a. There are a few reasons when it is necessary to initialize the local workspace directory.
 - i. When PDTTool or PDTStudio is first installed.
 - ii. When the workspace becomes out of sync and is throwing erroneous errors, it is a good idea to simply re-initialize the workspace to sync it up with the Perforce Depot.
 - iii. When moving between the use of "label" and "non-label" VCS commands.
 1. Simply put, the workspace for a label command is not compatible with the workspace required for non-label commands.
 2. When using a label command, the workspace is completely truncated at the root directory and only files associated with the label are brought down to the workspace.
 3. When using non-label commands, the entire depot as defined by the URL is brought down which represents the latest version of the depot. A non-label command is expecting all files in the depot to be present. If a VCS command using labels was used then the current set of files are not present and the command throws an error.
3. Checking out using Perforce Labels.
 - a. It is imperative that the first checkin from Studio if using PDTToolStudio or PDTTool be done from the Composite root directory. Be patient as this initial checkin may take a long time. It may be useful to pare down the CIS resources initially by exporting folders and deleting them to get to a bare bones folder structure. Then check in root.
 - i. *Why checkin at root.* Perforce must have root.cmf, shared.cmf, services.cmf, database.cmf and webservices.cmf as the baseline folders. If those are not there then PDTTool diffmerger marks folders for deletion when the car file is imported to the target CIS server.

- b. In studio, you would right-click on “localhost:<port> (/)” as shown in the diagram below:



- c. The following folders get checked in:

/	– root.cmf (readonly)
/policy	– policy.cmf (readonly)
/security	– security.cmf (readonly)
/services	– services.cmf (read/write)
/databases	– databases.cmf (read/write)
/webservices	– webservices.cmf (read/write)
/shared	– shared.cmf (read/write)
/system	– system.cmf (readonly)
/users	– user.cmf (readonly)

- d. Perform Best Practices for assigning CIS resources to Performce labels:

- i. Always assign the CIS base structures (root.cmf, databases.cmf, webservices.cmf, and shared.cmf)
- ii. The performce admin may assign sibling folders to a label such as /services/databases/TEST01, /services/webservices/WS01, /shared/tes01. All siblings will be synchronized with the workspace, zipped up into a single checkout.car file and then imported into the target CIS server.
- iii. **WARNING:** Any resources that are not assigned to a label and exist in the target CIS server are marked for deletion by PDTTool diffmerger. Subsequently, those resources are deleted from the target CIS server upon import of the checkout.car file.

- iv. **OBSERVATION:** During import of the car file, you will see the following deletions. These folders are read-only and do not actually get deleted by the import:

Deleted Paths: [/users_d, /security_d, /policy_d, /system_d]
Built deletion fragment:

```
<deletes>
  <delete path="/users" type="32001"/>
  <delete path="/security" type="32001"/>
  <delete path="/policy" type="32001"/>
  <delete path="/system" type="32001"/>
</deletes>
```

- e. PDTool Perforce Command for Labels performs the following steps
 - i. PDTool removes the VCS Workspace Project directory.
 - 1. For example if VCS_PROJECT_ROOT=cis_objects and VCS_WORKSPACE_DIR=\$VCS_WORKSPACE_HOME/\$VCS_WORKSPACE_NAME (v:\p4_wworkspace) then the workspace project directory is "v:\p4_wworkspace\cis_objects"
 - ii. PDTool creates the VCS Workspace Project directory
 - iii. PDTool checkout executes "p4 sync -f @label" which forces all files in the label to be brought down to the local workspace.
 - 1. At this point, only the files associated with the label are present in the workspace.
 - iv. PDTool performs a vcsDiffMergerCommand.
 - 1. Exports CIS target server into the VCS_TEMP_DIR
 - 2. Compares resources between VCS_WORKSPACE_DIR and VCS_TEMP_DIR and marks items for deletion in the target CIS server.
 - 3. Zips up the directory into a checkout.car
 - v. PDTool performs and import into the target CIS server with checkout.car
 - 1. Any resources marked for deletion are deleted upon import.

CVS specific information

General Notes: This section contains general notes about CVS and PDTool.

TFS specific information

General Notes: This section contains general notes about Team Foundation Server (TFS) and PDTool.

Use case: 2008 with Team Explorer Everywhere

- 1. General description

Team Explorer Everywhere is a multi-platform TFS client distributed by Microsoft. It currently consists of a command-line tool and an Eclipse plugin. TEE is a Java based app. More information about it is found here <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/team-explorer-everywhere>. The current version of TEE is 10 and it can be used as an in-place replacement for the standard Windows Team Explorer version 2010 command line client. TEE client supports all the options of the

standard Team Explorer command client and more. The main motivation to use this client was the ability to associate a checkin with one or more work item ids. This functionality is not available in the standard Team Explorer command client (It is available for GUI client). The use of TEE also gives us the ability to use TFS in an environment when PDTTool is hosted and runs on a non-Windows server. TEE can be downloaded from <http://www.microsoft.com/en-us/download/details.aspx?id=4240> . We only require TEE-CLC product from this download page. Before we use TEE in PDTTool we should accept the TEE EULA license by running "tf eula -accept".

2. What versions of TFS can it be used with

TEE can be used as an in-place replacement for Team Explorer 2010 command line client, which means we can use it as a client for TFS server 2005, 2008 and 2010.

3. Multi-platform – changes to / vs. – for commands

This is not specific to TEE. Both TEE and the regular Team Explorer command client accepts / and - for parameters on the command line. However / is valid only on the Windows environment. Using - in our code we can use the same codebase and use the standard TFS client or the TEE client and run on a Windows or non-Windows environment. Using - makes our code platform independent.

4. Strategies for shortening the length of the path when running in windows
 - a. How to get around the 259 limit – give examples of what the path is composed of.

Here is the scenario – VCS_WORKSPACE_HOME is set to \$APPDATA (In this example \$APPDATA is C:\Users\nqpe\AppData\Roaming)

VCS_WORKSPACE_DIR=\$VCS_WORKSPACE_HOME/\$VCS_TYPE\$_wks

VCS_TYPE=tfs2010

VCS_PROJECT_ROOT=TeamProject/DataFoundation/Development/Databases/CIS

VCS_REPOSITORY_URL=http://host:8080/DefaultCollection

TFS_SERVER_URL=\$\$/TeamProject/DataFoundation/Development/Databases/CIS

So the resultant workspace project folder is

C:\Users\nqpe\AppData\Roaming\tfs2010_wks\TeamProject\DataFoundation\Development\Databases\CIS

This ends up being 95 characters and we have only 164 characters for the path names in CIS.

This issue was overcome in two ways.

First, shorten VCS_WORKSPACE_DIR to C:\prj\tfs

Next, PDTTool's TFS implementation was enhanced to use the concept of workfold that is part of TFS. Workfold allows us to map an arbitrary TFS Server URL to a specific folder on the client's machine. For this TFS_SERVER_URL configuration variable was introduced (set to =TeamProject/DataFoundation/Development/Databases/CIS/) and it was mapped to the workspace project folder (C:\prj\tfs\CIS). So we reduced the path name from 95 characters to 15 characters.

- b. Collections and other strategies you use to shorten the path

Instead of having a project url of something like

\$/TeamProject/DataFoundation/Development/Databases/CIS/ TFS 2010 allows

you the ability of creating a collection at
http://host:8080/TEAMPROJECT/DataFoundation/Development/Databases
Then we can have the setting as follows
VCS_WORKSPACE_HOME=C:\prj
VCS_WORKSPACE_DIR=\$VCS_WORKSPACE_HOME/tfs
VCS_TYPE=tfs2010
VCS_PROJECT_ROOT=CIS
VCS_REPOSITORY_URL=http://host:8080/TeamProject/DataFoundation/Development/Databases/CIS/

This would result in the same path size, however creating additional collections on TFS has cost associated with it. There needs to be separate SQL Server database, the user and group permissions has to be maintained separately etc. CIS may have be part of a larger TFS project and it may not be feasible to have a separate collection as there are certain restrictions of what you can do if a project is spanned across collections. For more information about restrictions regarding collections please check <http://msdn.microsoft.com/en-us/library/dd236915.aspx>.

5. TFS deploy property file environment variables

a. What they do

Note: TFS_CHECKIN_OPTIONS has been depreceated. User VCS_CHECKIN_OPTIONS.

The TFS specific configuration variables that were added as part of the enhancement are

VCS_CHECKIN_OPTIONS and TFS_SERVER_URL

VCS_CHECKIN_OPTIONS are the options that passed to the TFS checkin command. This is opposed to VCS_OPTIONS where the VCS_OPTIONS are passed to all TFS commands. These specific checkin options are valid only on the checkin command and if passed to other commands like workspace creation or checkout it would result in error.

TFS_SERVER_URL was introduced for two reasons. One to use it as a TFS server URL path for workfold command and also because the TFS server path always begin with '\$' character, we should be processing this variable without calling CommonUtils.extractVariable method. Use \$\$ to escape the \$.

b. How the values are related to VCS_REPOSITORY_URL

The concept of collections was introduced in TFS 2010. In TFS 2005 and TFS 2008 there is only one default collection. The VCS_REPOSITORY_URL will always be <https://hostname:8080/>. The path to the project resources will be in TFS_SERVER_URL. In TFS 2010 if the CIS project is in a separate collection then the path to the project will be part of VCS_REPOSITORY_URL. E.g. <https://hostname:8080/path/to/the/project/> The TFS_SERVER_URL will then be set to \$

c. Provide examples of what is in TFS directory structure and how PD Tool references these TFS folders

- What should VCS_REPOSITORY_URL look like?

- What should VCS_PROJECT_ROOT point to?
- What should TFS_SERVER_URL point to and why is this an advantage (benefit)?

When the server's default collection is used (TFS 2005 and TFS 2008 has only one default collection), the configuration variables will look like this:

```
VCS_WORKSPACE_HOME=C:\prj
VCS_WORKSPACE_DIR=$VCS_WORKSPACE_HOME/tfs
VCS_TYPE=tfs2010
VCS_PROJECT_ROOT=CIS
VCS_REPOSITORY_URL=http://host:8080/
TFS_SERVER_URL=/$/TeamProject/DataFoundation/Development/Databases/CIS/
```

When TFS 2010 is used with a separate collection for CIS project and the CIS project is defined in the root of the collection

```
VCS_WORKSPACE_HOME=C:\prj
VCS_WORKSPACE_DIR=$VCS_WORKSPACE_HOME/tfs
VCS_TYPE=tfs2010
VCS_PROJECT_ROOT=CIS
VCS_REPOSITORY_URL=http://host:8080/TeamProject/DataFoundation/Development/Databases/CIS/
TFS_SERVER_URL=/$/
```

6. Associate a Work Item with a TFS Task

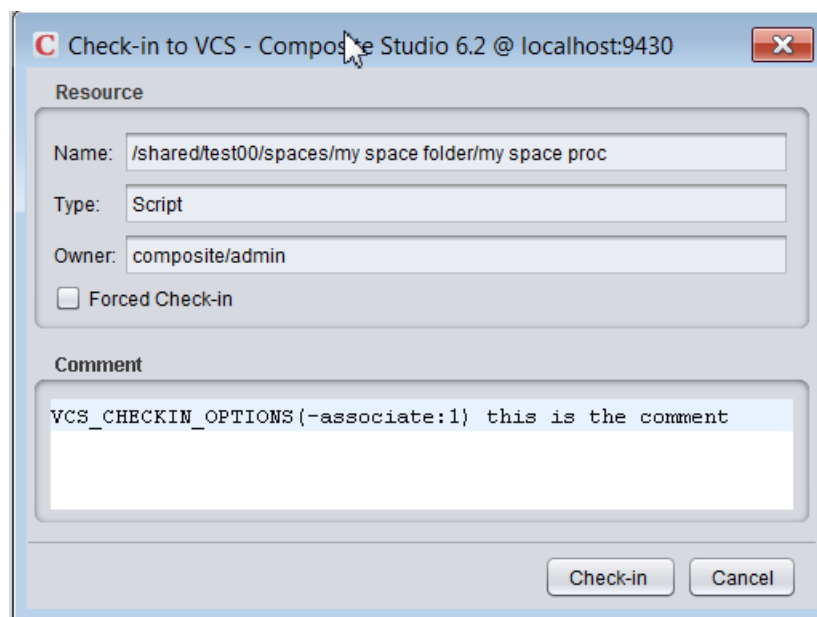
- This topic discusses how to associate work items with the checkin process. A work item is a way of associating an ID from TFS with a task. Typically tasks are associated with User Stories. Generally speaking, developers are working on tasks that make up the bigger picture of User Stories. Therefore, the task ID is found in TFS and is assigned to a developer. The developer does their work in Composite and then checks in code. From a PDTTool perspective, the command line option “-associate:<id>” is used to inject the association of the work item id with the checking in of code. There is a VCS_CHECKIN_OPTIONS property in the deploy.properties where the -associate:<id> command line option can be placed.
- From a Composite Studio perspective, the developer now has the ability to inject command line options via the Studio Comments. The studio comment section is a free-form section. What we have done with PDTToolStudio is provide a template by which to inject command line checkin options. The example below shows the use of the template VCS_CHECKIN_OPTIONS(<command>) and a free form comments.

- Some comment. VCS_CHECKIN_OPTIONS(-associate:1) Another comment.

The result of this command is that the command itself is stripped out of the comments and only the free form comments are passed to the VCS while the command is added to the command line during execution. The commands will be added to any existing commands specified in the deploy.properties file or the VCSModule.xml file for the property VCS_CHECKIN_OPTIONS.

- Command: -associate:1
- Comments: Some comment. Another comment.

The screen shot below demonstrates the usage within Composite Studio.



What PDTToolStudio executes:

2014-03-07 15:42:09,657 main INFO [com.cisco.dvbu.ps.common.util.ScriptExecutor] -
<Successfully executed command:

E:/dev/vcs/TEE-CLC-11.0.0/tf.cmd checkin

W:/TFSS/Composite_62/cis_objects/shared/test00/spaces/my_0020space_0020folder/my_0020space_0020proc_procedure.cmf -comment:@comment_2014_03_07_15_41_58_466.txt -noprompt /login:mtinius,*** -associate:1>**

2014-03-07 15:42:09,657 main INFO [com.cisco.dvbu.ps.common.util.ScriptExecutor] -
<Output:

shared\test00\spaces\my_0020space_0020folder:

Checking in lock, edit: my_0020space_0020proc_procedure.cmf

Changeset #20 checked in.

Associated work item 1.

>

2014-03-07 15:42:09,669 main INFO

[com.cisco.dvbu.ps.deploytool.services.VCSManagerImpl] -

<DEBUG2::vcsStudioCheckin::===== COMPLETED VCS [tfs2012] Studio Checkin

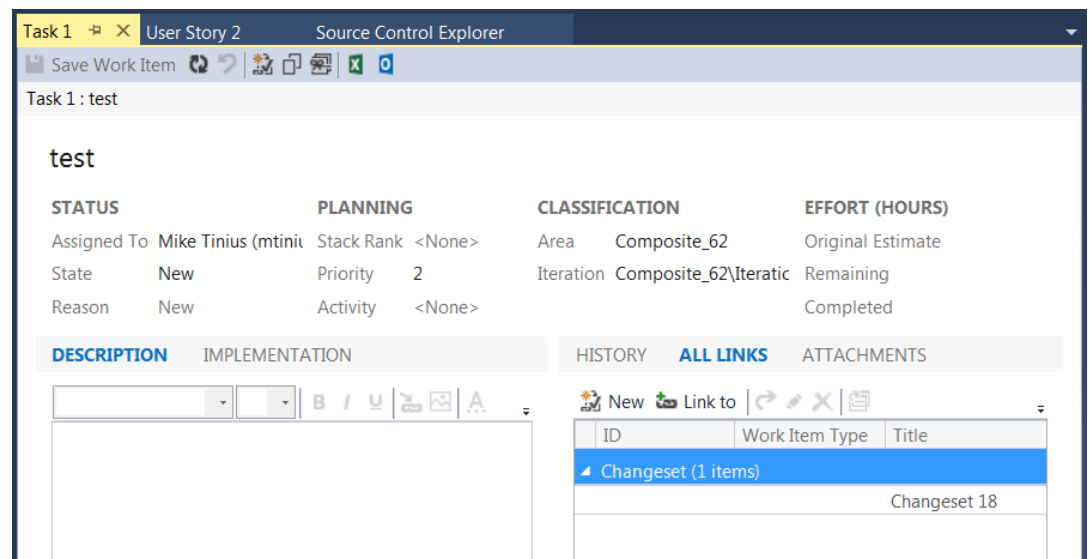
=====>


```

2014-03-07 15:42:09,669 main INFO
[com.cisco.dvbu.ps.deploytool.services.VCSManagerImpl] - <DEBUG2::vcsStudioCheckin::>
checkin.bat::execute::
checkin.bat::execute::----- SUCCESSFUL SCRIPT COMPLETION [ ] -----

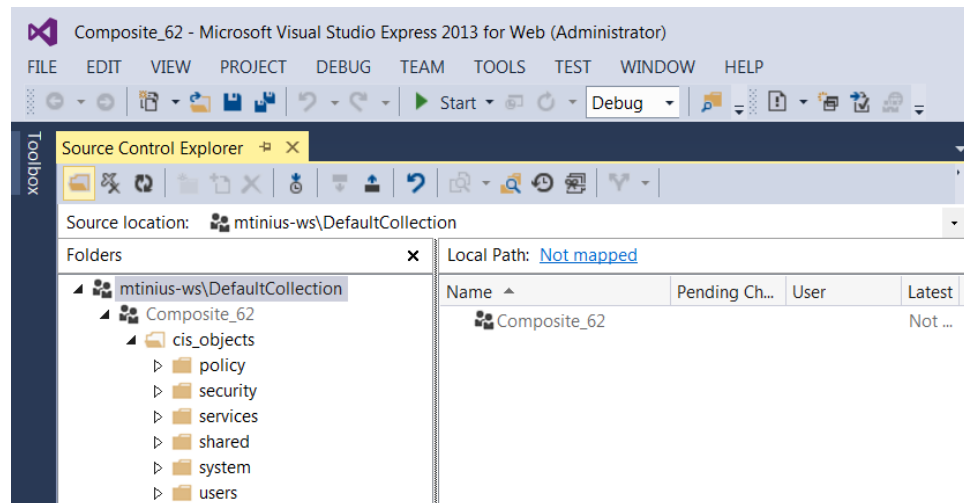
```

- c. The screen shot below shows the results of a task and the current change set for the composite code that was checked in. This example shows Task 1 which has an id=1. Composite used “-associate:1” on the command line to associate the checkin with this task.



TFS Preparation Checklist

- Install Team Explorer Everywhere (TEE) 11
- Install Visual Studio with Team Foundation Services 2012 or 2013
 - This can be used for browsing the TFS repo.
- Configure a repository/collection for Composite [TFS Admin]
 - E.g. DefaultCollection
 - Need to create “Team Project” for Composite [TFS Admin]
 - E.g. Composite_62
 - Define your folder structure
 - /cis_objects which will hold the Composite repository objects.
 - As shown in the diagram below, the Composite repository is checked in at the root level of composite and contains several sub-folders including: /policy, /security, /services, /shared, /system and /users.



- Need to create a composite group (AD Group) and assign permission
 - For PDTTool, this is a group that will be responsible for doing deployments.
- Needed to add an AD user to the group
 - For PDTTool, this is a user that will be responsible for doing deployments.
- Get TFS base server URL
 - E.g. `http://tfs_host:8080/tfs/DefaultCollection`
- Get TFS Composite project and folder structure that has been configured in TFS
 - This will be needed to configure the `TFS_SERVER_URL` and `VCS_PROJECT_ROOT`
- Configure PDTTool
 - Configure the `deploy.properties` or `VCSModule.xml`
 - For example, give the TFS URL= `http://tfs_host:8080/tfs/DefaultCollection` and the TFS folder structure of under the `DefaultCollection`
`Composite_62/cis_objects` the variables in the `deploy.properties` would be configured as follows:
 - `VCS_REPOSITORY_URL=http://tfs_host:8080/tfs/DefaultCollection`
 - `VCS_PROJECT_ROOT=Composite_62/cis_objects`
 - `TFS_SERVER_URL=$$/Composite_62/cis_objects`
 - When not set, errors occurred during initialization
 - `C:/PDTTool/TEE-CLC-11.0.0/tf.cmd` Execution Returned an Error=An argument error occurred: First free argument must be a server path.
 - 2 \$ to escape the \$
- Checking in a blank project of Composite
 - More times than not the Composite instance already has a large number of objects created. The initial check-in of a Composite repository that is loaded with resources may take a very-very long time.
 - Recommendation (strategy).
 - Create a new instance of Composite with no objects in it.
 - Configure PDTTool to point to that instance and the TFS repo.

- First, do a workspace initialization as per the PDTool documentation.
- Second, perform a checkin of the objects from the empty Composite repo. The base structure that was shown above will be checked in. The base structure with no resources in composite looks like this:

```

/cis_objects
  /policy
    /security
      /user
        user.cmf
      security.cmf
    policy.cmf
  /security
    /rowlevel
      /filters
        filters.cmf
      rowlevel.cmf
    security.cmf
  /services
    /databases
      databases.cmf
    /webservices
      webservices.cmf
    services.cmf
  /shared
    shared.cmf
  /system
    /connector
      connector.cmf
    system.cmf
  /users
    /composite
      /admin
        admin.cmf
      composite.cmf
    users.cmf
  root.cmf

```

- Third, Reconfigure PDTool to point to the real Composite repository.

CONCLUSION

Concluding Remarks

The PS Promotion and Deployment Tool for Studio VCS Integration is a set of pre-built modules intended to provide a turn-key experience for checking in and checking out resources from a VCS repository to the Composite Studio CIS Instance the user is connected with.

ABOUT COMPOSITE SOFTWARE

Composite Software, Inc. ® is the only company that focuses solely on data virtualization.

Global organizations faced with disparate, complex data environments, including ten of the top 20 banks, six of the top ten pharmaceutical companies, four of the top five energy firms, major media and technology organizations as well as government agencies, have chosen Composite's proven data virtualization platform to fulfill critical information needs, faster with fewer resources.

Scaling from project to enterprise, Composite's middleware enables data federation, data warehouse extension, enterprise data sharing, real-time and cloud computing data integration.

Founded in 2002, Composite Software is a privately held, venture-funded corporation based in Silicon Valley. For more information, please visit www.compositesw.com.



Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

Printed in USA

CXX-XXXXXX-XX 10/11

Composite Software is now part of Cisco
© 2013 Cisco and/or its affiliates. All rights reserved. This document is Cisco Public.

Page 68 of 68