PDTool Version Control System (VCS) Module User Guide

An Open Source Asset for use with TIBCO® Data Virtualization

Project Name	AS Assets PDTool (Promotion and Deployment Tool)
Document Location	This document is only valid on the day it was printed. The source of the document will be found in the PDTool and PDToolRelease folder (https://github.com/TIBCOSoftware)
Purpose	User's Guide

www.tibco.com

Global Headquarters
3303 Hillview Avenue
Palo Alto, CA 94304

Tel: +1 650-846-1000
+1 800-420-8450
Fax: +1 650-846-1005

Revision History

Version	Date	Author	Comments
1.0	08/01/2011	Mike Tinius	Initial revision for VCS Module User Guide
2.0	10/04/2011	Mike Tinius	Added Microsoft Team Foundation Server (TFS) support
2.1	01/26/2012	Mike Tinius	Added the ability to set VCS configuration properties using the VCSModule.xml instead of deploy.properties.
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	07/09/2012	Mike Tinius	Added labels for Perforce. Implemented SUBST command in windows for “file too long” problem.
2.4	08/15/2012	Mike Tinius	Added reference to TortoiseSVN 1.7.1 command line client usage
2.5	10/29/2012	Mike Tinius	Added documentation regarding resource types for internal and external display.
2.6	01/13/2013	Mike Tinius	Updated docs to accurately represent the VCS_REPOSITORY_URL=http:////host:port/folder
3.0	8/21/2013	Mike Tinius	Updated docs to Cisco format.
3.1	2/3/2014	Mike Tinius	Added support notes for TFS 2012.
3.2	2/18/2014	Mike Tinius	Prepare docs for open source.
3.3	3/6/2014	Mike Tinius	Add TFS specific documentation
3.4	3/24/2014	Mike Tinius	Changed references of XML namespace to www.dvbu.cisco.com
3.5	4/11/2014	Mike Tinius	Added vcsInitializeBaseFolderCheckin for vcs base folder initialization.
3.6	5/16/2014	Mike Tinius	Added vcsScanPathLength to check for folders exceeding 259 characters which includes the total length of VCS_WORKSPACE + VCS_PROJECT_ROOT + CIS_FOLDER.
3.7	5/28/2014	Mike Tinius	Improved documentation and added VCS_BASE_TYPE, VCS_CIS_IMPORT_OPTIONS and VCS_CIS_EXPORT_OPTIONS to deploy.properties and VCSModule.xml.
3.8	11/17/2014	Mike Tinius	Updated license.
3.9	12/1/2014	Mike Tinius	Added option -IGNORE_INIT_LINK to be used for read-only subversion access for VCS_WORKSPACE_INIT_LINK_OPTIONS in both deploy.properties and VCSModule.xml.
3.10	3/4/2015	Mike Tinius	Updated docs to Cisco format.
3.11	9/21/2015	Mike Tinius	Added TFS_USE_EXISTING_WORKSPACE=[true|false]
3.12	4/4/2016	Mike Tinius	Added SVN support for revision arguments: BASE, COMMITTED, PREV
4.0	12/14/2017	Mike Tinius	Initial revision with Tibco
4.1	06/11/2018	Mike Tinius	Fixed issues with PDTool GIT checkin and workspace initialization.
5.0	08/27/2020	Mike Tinius	Updated documentation

5.1	10/20/2020	Mike Tinius	Updated documentation

Related Documents

Name	Version
PDTool Installation Guide.pdf	Mike Tinius
PDTool User's Guide.pdf	Mike Tinius

Supported Versions

Name	Version
TIBCO® Data Virtualization	7.0.8 or later

Table of Contents

1 Introduction

Purpose

This document describes how to configure a command-line environment to communicate with a version control system (VCS). Once completed, the user will be able to check in and check out Data Virtualization (DV) objects from the command line.

These command line scripts support the “Promotion and Deployment Tool (PDTool)” for command-line and ANT-based resource deployment.

It is expected that a VCS server instance and a suitable repository are already available. 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(s) process works alongside the checkout/checkin code process, see the section “*Version Control and DV*”. This document replaces previous DV Tech notes including: VCS TechNote-5.1, VCS TechNote-5.2 and VCS TechNote-6.0.

Audience

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

- Architects
- Developers
- Administrators
- Operations personnel

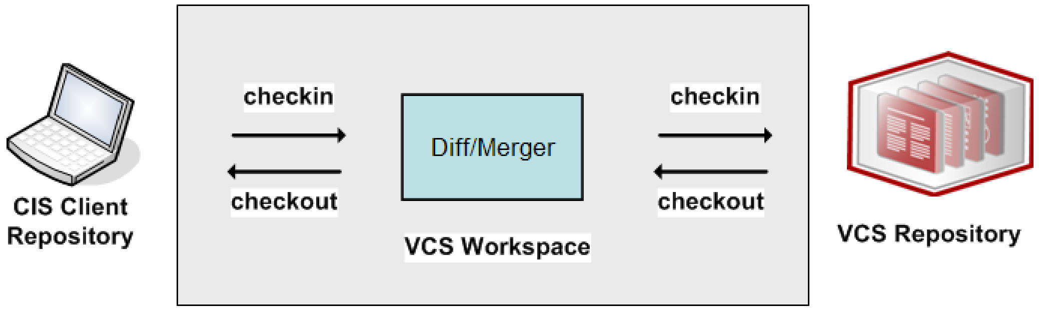
References

Product references are shown below. Any references to CIS or DV refer to the current TIBCO® Data Virtualization.

- TIBCO® Data Virtualization was formerly known as
    - Cisco Data Virtualization (DV)
    - Composite Information Server (CIS)

2 Version Control and Data Virtualization (DV)

Data Virtualization (DV) supports version control systems to maintain versions of DV resources. Using DV Studio or the command line interface and batch script files, DV users can maintain distinct revisions of DV resources generated during development using the version control system (VCS) of their preference. The DV 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.



Batch Script Driven

The extensibility of the VCS integration framework offered in Data Virtualization allows it to adapt to the special needs of the VCS used by a specific DV customer.

VCS Topologies

DV 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 Direct VCS Access and Multi-User Managed VCS Access.

Single-Node Topology

Single-Node refers to the scenario where a single Studio user is connected to their own DV development server and the Studio user has the ability to check-in and check-out their own DV 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.

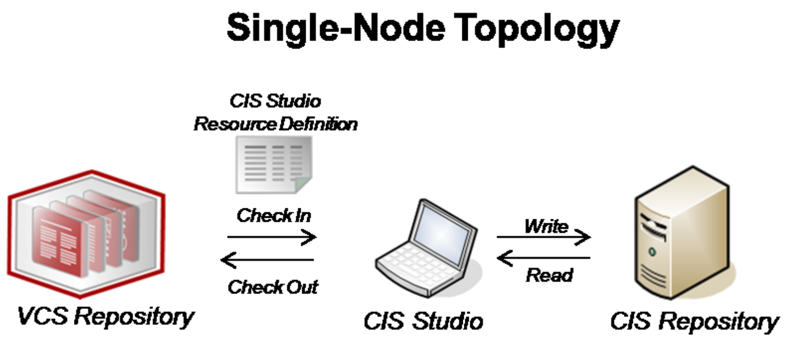


Diagram 1: Single-Node Topology

Multi-Node Topology

Multi-Node refers to the scenario where each Studio user is connected to their own DV development server and each Studio user has the ability to check-in and check-out their own DV 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.

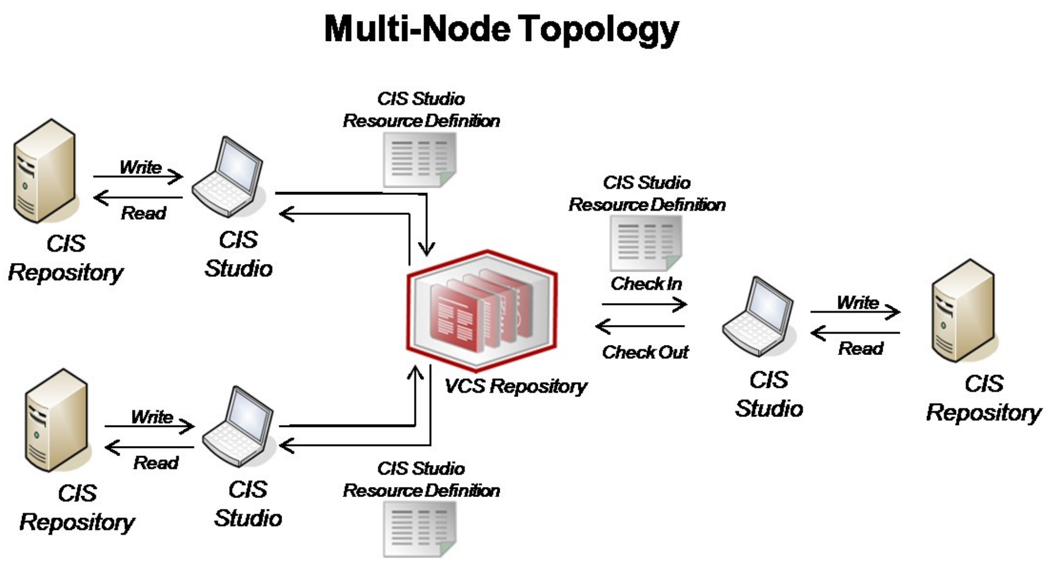


Diagram 2: Multi-Node Topology

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 DV development server and all Studio users have the ability to directly check-in or check-out DV 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 DV instance, special care must be taken so that 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 DV 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.

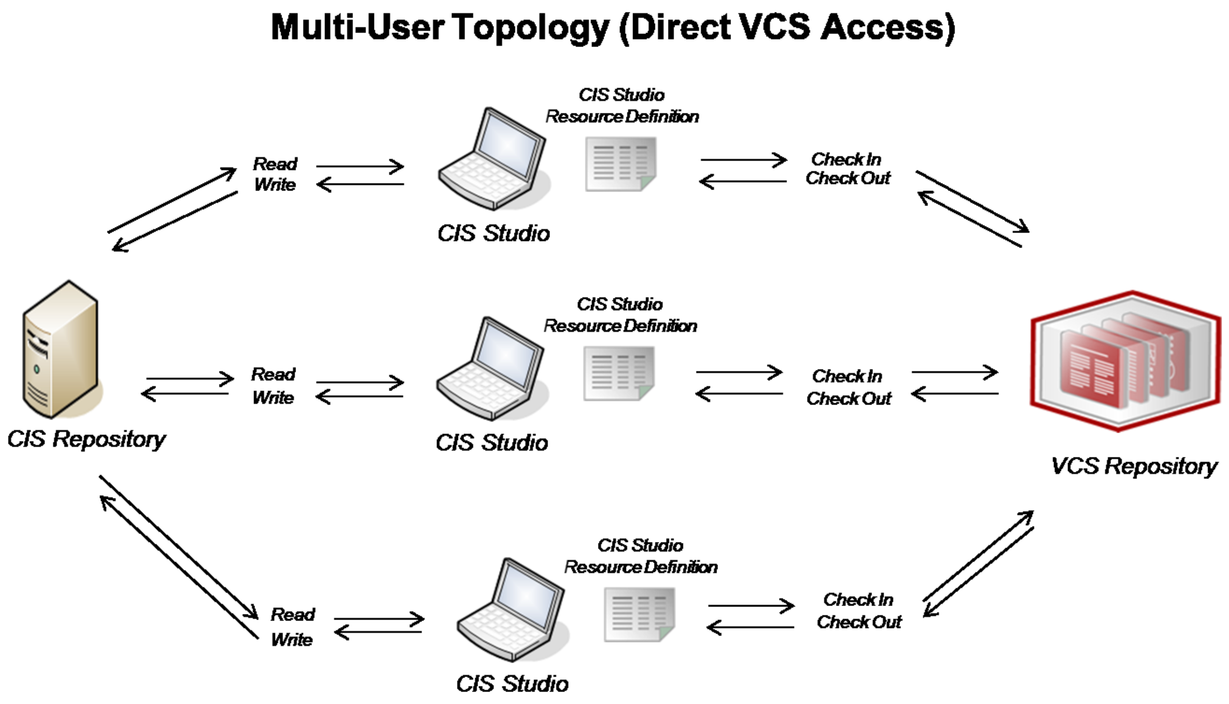


Diagram 3: Multi-User Topology (Direct VCS Access)

Multi-User Topology (Managed VCS Access)

Multi-User Managed VCS Access refers to the scenario where multiple Studio users are connected to a shared DV development server but only one "Managed" Studio user has the ability to check-in and check-out DV 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.

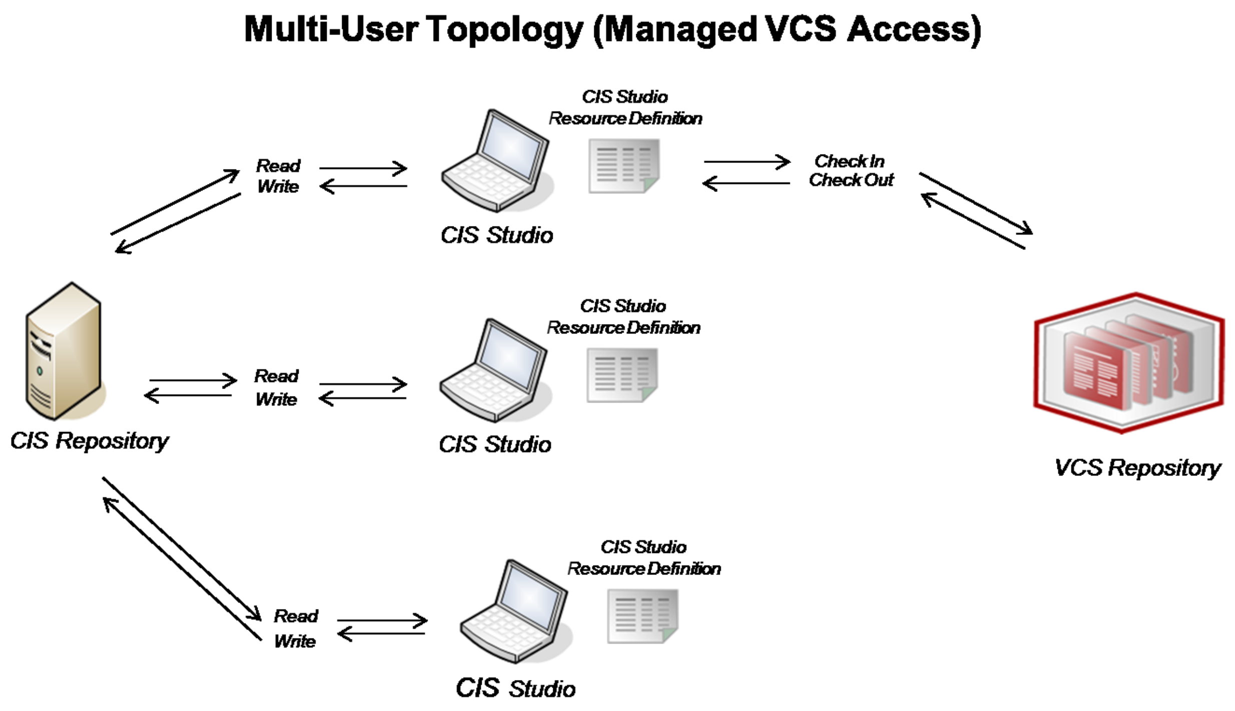

Multi-User Topology (Managed VCS Access)
CIS Studio Resource Definition
Read
Write
CIS Studio
Check In
Check Out
CIS Repository
CIS Studio Resource Definition
Read
Write
CIS Studio
VCS Repository
CIS Studio Resource Definition
Read
Write
CIS Studio

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

How Version Control Works in DV

VCS integration with DV 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 DV 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.

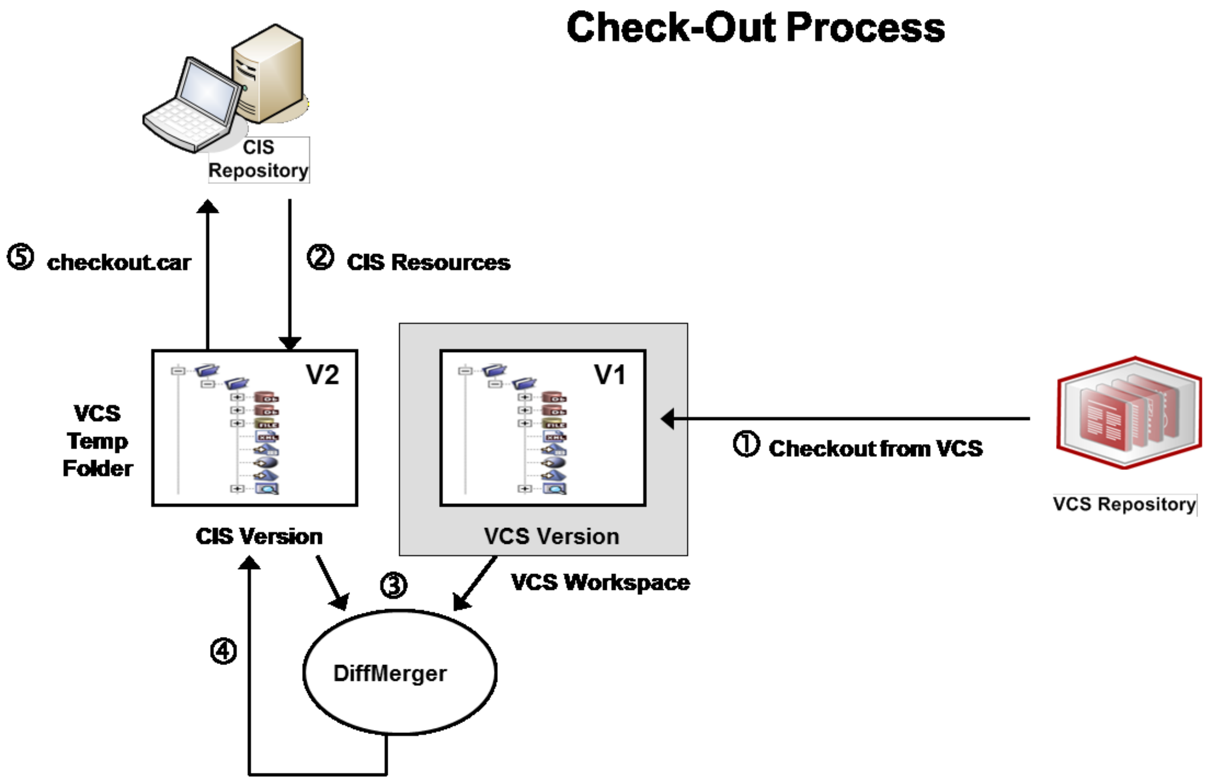
The Lifecycle Listener controls the VCS-specific lifecycle processes required to pre-process and post-process files that are checked in and checked out of the VCS. Sample listeners are provided for Subversion; otherwise a listener can be implemented in Java. Contact Professional Services for an engagement.

VCS Workspace

The Data Virtualization VCS implementation utilizes a workspace that acts as a bridge between the DV and VCS repositories. The VCS workspace is a user-defined folder in the file system that reflects the DV 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:

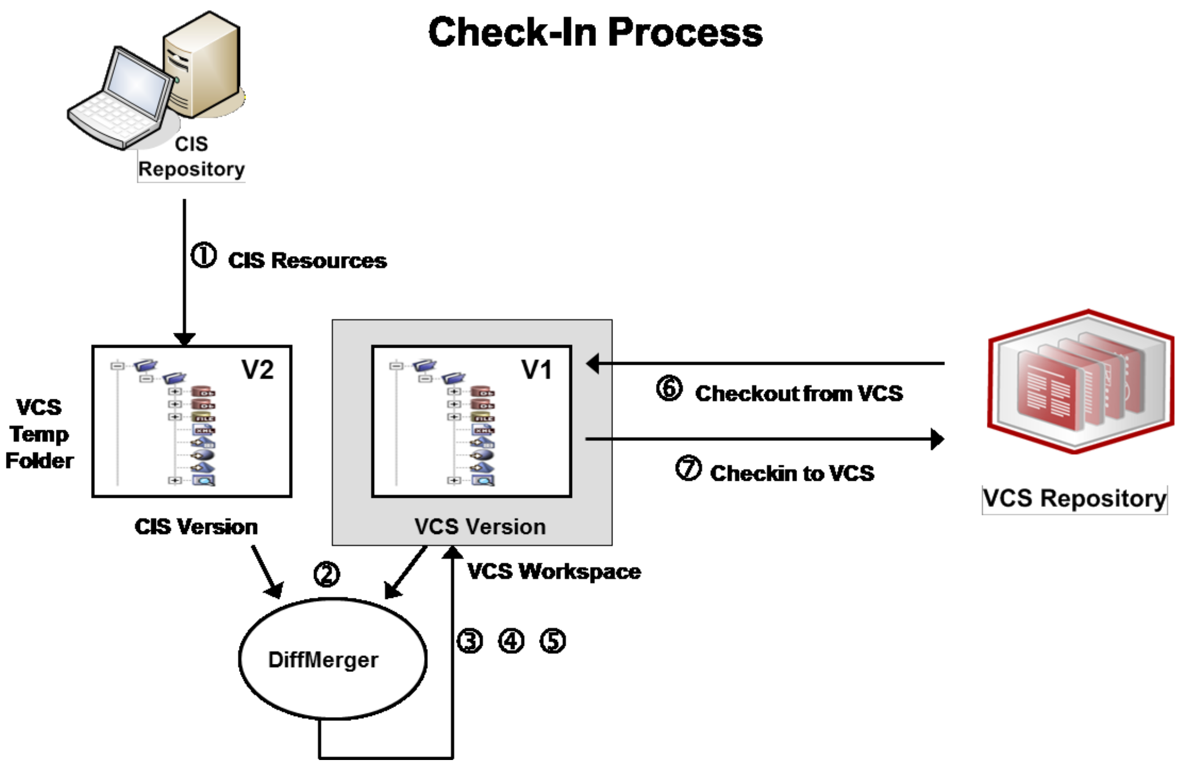
1. A current snapshot of the specified DV resources is obtained from the VCS workspace (V1).
2. The current version of the same resource subtree in DV is captured into the VCS temp folder (V2).
3. The DiffMerger batch script is applied against the DV and VCS versions to compute V1-V2. V1-V2 captures 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 DV 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 DV resources in a version control system is illustrated below.



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

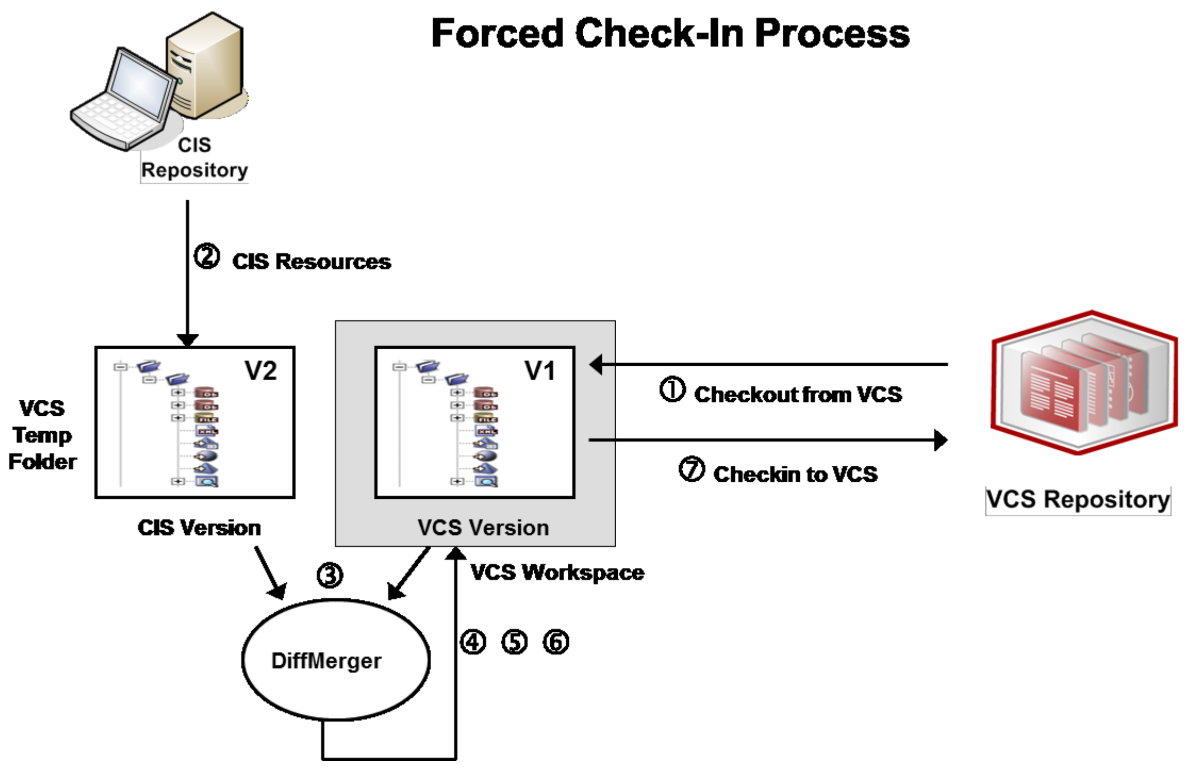
1. The current version of the same resource subtree in DV is captured into the VCS temp folder (V2).
2. The DiffMerger batch script is applied against the DV and VCS versions to compute V2-V1. V2-V1 captures the required modifications (that is resource creations, updates or deletions) required to convert 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 DV 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 DV 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 DV resources is obtained from the VCS workspace (V1).
2. The current version of the same resource subtree in DV is captured into the VCS temp folder (V2).
3. The DiffMerger batch script is applied against the DV 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 DV environment. The VCS check-in and check-out processes are DV security sensitive as follows:

- DV authentication is required.
- DV authorization rules are applied:
    - Check-in by a certain DV user applies only to those DV resources that are read-accessible by that user.
    - Check-out by a certain DV 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, DV resources need to be represented in the form of (.cmf) files that are stored on the file system of the DV user's workspace. Different file systems impose restrictions on the characters that are allowed in file names while DV does not impose any restrictions on resource names. To ensure that names are valid, DV 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

```
private static boolean isAccepted(char c) {

   return Character.isLetterOrDigit(c) || c == '_' || c == '-';
```

}

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".

The utility named "vcs_name_codec", located in the <DV_HOME install directory>\bin folder, can be used to determine the encoded/decoded form of resource/file names. However, this is done automatically by PDTool.

```
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 DV 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. Professional Services is available for engagements to enhance the Promotion and Deployment tool to work with other VCS systems besides what is supported out-of-the-box.

Version Control and DV Resources

The following DV 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')
- DV 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 DV 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.

3 Pre-Requisites for the Computer Hosting PDTool

Data Virtualization (DV)

The Promotion and Deployment Tool supports DV 7.x and DV 8.x.

Computer OS

The PDTool is supported on a number of different OS platforms that DV runs on including UNIX (Linux) or Windows (10).

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 the connection information to the VCS Server.

Github (GIT) Client Installation

The computer must have a VCS command line client tool installed. When installing the above client tool, use all default values. Version information for Git:

```
Type 'git --version' to see the program version
  git version 2.26.1.windows.1
```

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 http://www.perforce.com/manual.
   Server 2010.2/295040.
```

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).
```

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.

Version information for Team Explorer Everywhere:

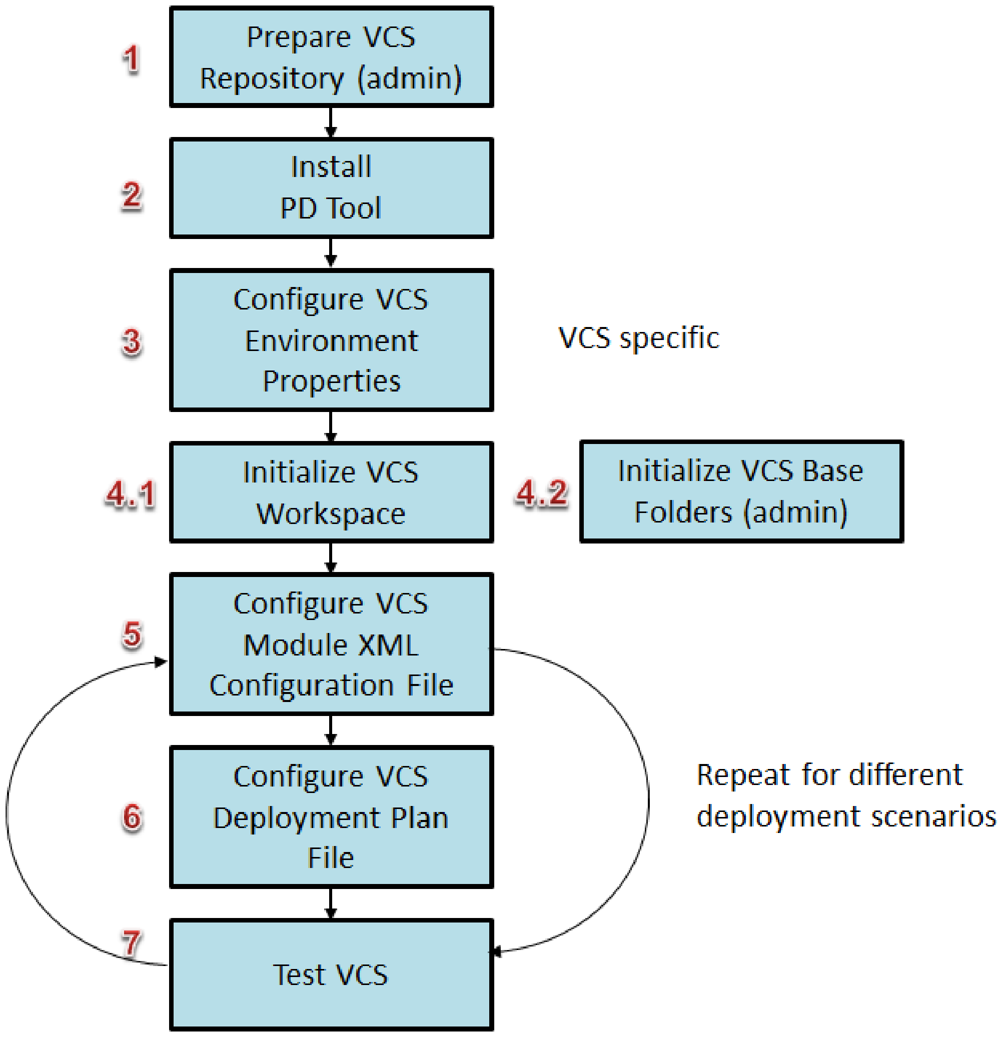
```
C:\temp>tf.cmd help
Team Explorer Everywhere Command Line Client (version 11.0.0.201306181526)
Type tf help <command name> for command line description.
```

4 Configuring Version Control for PDTool

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, and editing property files that customize the version control process for the specific VCS environment. Finally, for execution there is one script for windows and a corresponding script for UNIX to execute the VCS orchestration deployment scenario. A deployment scenario may also be referred to as a deployment plan.

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

CIS VCS Configuration Process for PDTool


1
Prepare VCS Repository (admin)
2
Install PD Tool
3
Configure VCS Environment Properties
VCS specific
4.1
Initialize VCS Workspace
4.2
Initialize VCS Base Folders (admin)
5
Configure VCS Module XML Configuration File
6
Configure VCS Deployment Plan File
Repeat for different deployment scenarios
7
Test VCS

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 may 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 DV repository objects.

   There are two important variables that come out of this section which are VCS_REPOSITORY_URL and VCS_PROJECT_ROOT. The VCS_REPOSITORY_URL is the point into the VCS repository including host, port and possibly other qualifying information. For example with subversion it may include some set of mandatory folders and for TFS it will include the collection name. The VCS_REPOSITORY_URL consists of the folders within the VCS repository that exist from the VCS_REPOSITORY_URL up to where the DV metadata folders will begin. The DV metadata folders begin with root.cmf, /shared, /services, /policy and etc.

2. *Step 2: Install the PS Promotion and Deployment Tool (PDTool)*

   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 DV 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: Configure VCS Module XML Configuration File*

   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.

6. *Step 6: Configure VCS Deployment Plan (.dp) File*

   This step is an iterative step used to configure the VCS deployment plan. The deployment plan tells PDTool what to methods to execute.

7. *Step 7: Test the Configuration (Script Execution)*

   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

[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.

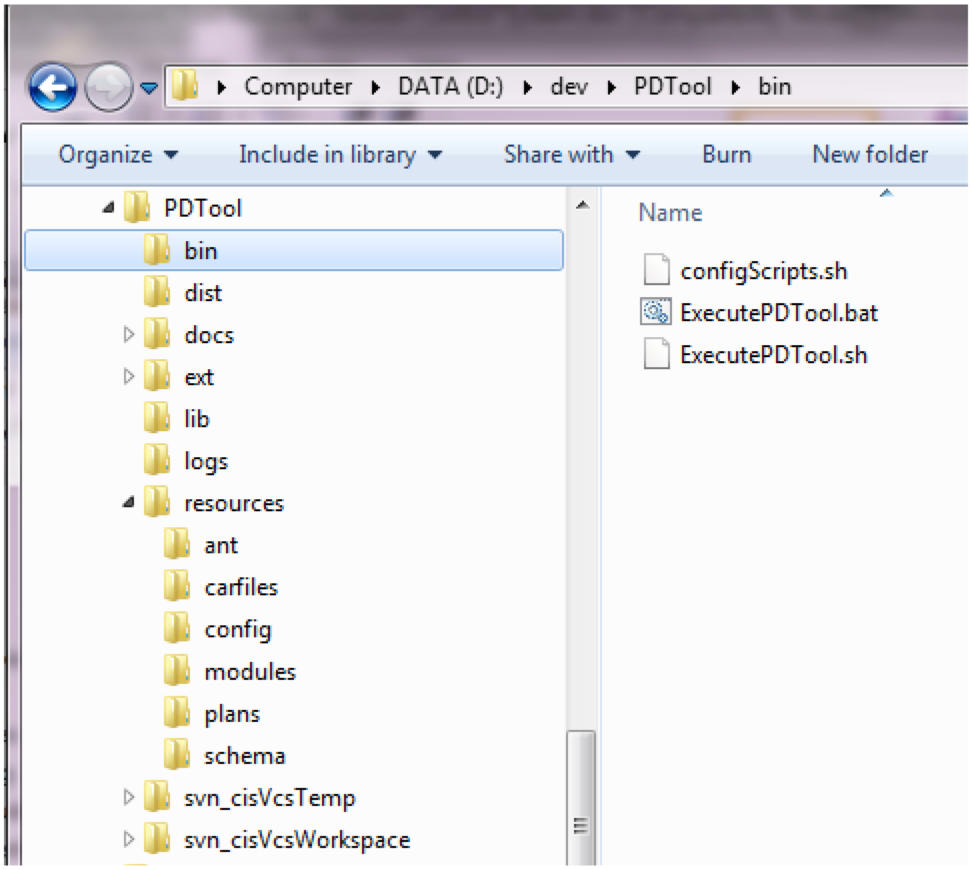
Step 2: Install the Promotion and Deployment Tool (PDTool)

Important note: This step is fully detailed in the "***PDTool Installation Guide.pdf***".

1. System Requirements:
   - **Java 7/8 -** JRE 1.8 is required to be present on the system for command line deployment. JDK 1.8 is required to be present on the system for Ant deployment.
   - **Read/Write Access** – The script must have read/write access to the file system where PDTool is installed. 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, TortoiseSVN 1.7.1 or higher with installed commad line client may be used. Prior versions were did not have 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 PDTool.

1. Follow the installation steps for the Promotion and Deployment Tool. After unzipping PDTool.zip in Windows, your directory would look something like this:



Step 3: Configure VCS Environment Properties

1. As of January 26, 2012 (V2.1), users have a choice of where to configure VCS Environment Properties.
   a. Option 1: (default) deploy.properties
      i. Definition – deploy.properties provides a system-wide VCS definition for all VCS-related activities. This option will become deprecated over time. For now, it will remain as the backward compatibility option.
   b. Option 2: (new) VCSModule.xml
      i. Definition – VCSModule.xml provides the ability to define multiple VCS configuration, environment properties. This gives the user the flexibility to designate which VCS connection should be used with a particular action in the

plan file. Ultimately, it allows the user to reference different URLs (branches/tags) within the VCS repository without having to change the system-wide deploy.properties file.

c. Property (VCS_MULTI_USER_TOPOLOGY) will remain in the deploy.properties file as a system-wide environment property. All other VCS properties may be defined in the VCSModule.xml

2. **Modify deploy.properties** – located in ../resources/config directory. There are also several example deploy.properties files that can be used when setting up VCS for Git, Subversion, Perforce or Microsoft Team Foundation Server.

   a. This step is applicable for both UNIX and Windows and Command-Line and Ant deployment.

   b. **General Instructions for deploy.properties**

      i. **PROJECT_HOME** is a variable referenced and is automatically set upon invocation of the ExecutePDTool script based on relative location of PDTool/bin.

      ii. Always use forward slashes for both Windows and Unix paths and URLs.

      iii. Variables may use $ or % notations. It is not operating system specific.

      iv. Variables may resolve to this property file, Java Environment (-DVAR=val) or the System Environment variables

      v. Surround variables with two $ or two % when concatenating strings (e.g. $VCS_TYPE$_cisVcsTemp)

   c. **<u>Section: VCS Environment Variables</u>**

      i. Set VCS_MULTI_USER_TOPOLOGY=[true|false] – This property will remain in the deploy.propeties file as a system-wide environment settings.

         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 PDTool user is connected to
their own CIS development server and the Studio user or PDTool 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 PDTool client is connected
to their own CIS development server and *EACH* Studio user or PDTool 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
PDTool clients are connected to a central CIS development server and only one *MANAGED*
Studio user or PDTool 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
PDTool clients are directly connected to a central CIS development server and all Studio
users or PDTool 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 PDTool 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
#
```

ii. 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, tfs2012 or tfs2013. 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, 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
#     Team Foundation Server=tfs2005 or [tfs2010, tfs2012, tfs2013]
VCS_TYPE=svn
#     Recommended options=[SVN, P4, or TFS].  Choose to coincide with VCS_TYPE.
VCS_BASE_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,tf.cmd]
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
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 $/DV_70/cis_objects
W:/wks/DV_70/cis_objects -workspace:wks -noprompt /login:user,********
#      SVN: svn import -m "linking workspace to the VCS repository" .
"${VCS_REPOSITORY_URL}/${VCS_PROJECT_ROOT}" ${SVN_OPTIONS} ${SVN_AUTH}
${VCS_WORKSPACE_INIT_LINK_OPTIONS}
#              -IGNORE_INIT_LINK - This option is used to ignore this step when users have
READ-only access and will only be doing check-out from subversion and not check-in.
#       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.]
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 $/DV_70/cis_objects -
noprompt /login:user,********
#       SVN: svn co "${VCS_REPOSITORY_URL}/${VCS_PROJECT_ROOT}" ${SVN_OPTIONS} ${SVN_AUTH}
${VCS_WORKSPACE_INIT_GET_OPTIONS}
#        P4: p4 sync ${VCS_WORKSPACE_INIT_GET_OPTIONS}
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/DV_70/cis_objects/shared/test00  $/DV_70/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}
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}
# 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}
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=

# 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=
# 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=

# VCS Resource Import Options:
#   Occurs during a vcsCheckout when the temporary checkout.car file is imported into the
target server.
#   The import is the final step in the process.  The backwards compatible option is to
include access: -includeaccess
#   Consult the Archive Module documentation for more options.
VCS_CIS_IMPORT_OPTIONS=-includeaccess

# VCS Resource Export Options:
#   Occurs during a vcsCheckin when the temporary checkout.car file is exported from the
target server.
#   The export from the target is used to compare resources against the VCS repository.
The backwards compatible option is leave this option blank.
#   Consult the Archive Module documentation for more options.
VCS_CIS_EXPORT_OPTIONS=
#--------------------

#

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

iii. 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 DV 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:

1. **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.company.com/svn/sandbox/PDTOOL/$CIS_VERSION
#
#     Perforce   - The Repository URL is the host and port in perforce -
#                  Command Format: [hostname:port]
#                         Example: myhost:1666
#     TFS - The base HTTP URL in Team Foundation Server
#                  Command Format: [http:////hostname.domain:8080/tfs/basename]
#                         Example: http:////myhost:8080/tfs/TeamCollection
#
VCS_REPOSITORY_URL=http:////myhost.company.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
#     TFS        - The project name / project folder name [TeamProject/cis_objects]
VCS_PROJECT_ROOT=cis_objects
#-------------------
#
#-------------------
# VCS_WORKSPACE_HOME - This is the CIS VCS Workspace Home.
#     It is recommended to set the location to PDTool home [e.g. $PROJECT_HOME].
#     The user does have the flexibility to place the VCS workspace in a location other
than PDTool 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 PDTool on the same machine as PDToolStudio 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. $VCS_TYPE$ww - $VCS_TYPE$ gets evaluated as a variable. "ww" is a string that
gets concatentated. Result: svnww
#    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_TEMP_DIR::       VCS Temp Dir is a combination of the VCS_WORKSPACE_HOME and a temp
dir name such as $VCS_TYPE$t.
VCS_WORKSPACE_NAME=$VCS_BASE_TYPE$ww
VCS_WORKSPACE_DIR=$VCS_WORKSPACE_HOME/$VCS_WORKSPACE_NAME
VCS_TEMP_DIR=$VCS_WORKSPACE_HOME/$VCS_BASE_TYPE$t
#-------------------
#
#-------------------
# 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=
#-------------------
#
#-------------------
```

```
# 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: ./ExecutePDTool.sh -encrypt ../resources/config/deploy.properties
#   Windows: ExecutePDTool.bat -encrypt ../resources/config/deploy.properties
VCS_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.
#    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 use “notepad” and for UNIX use “vi”.   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 PDTool 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 by
VCS_WORKSPACE_NAME which is also the directory at the end of VCS_WORKSPACE_DIR.
# example:  If [ VCS_WORKSPACE_DIR=$VCS_WORKSPACE_HOME/$VCS_WORKSPACE_NAME ] then
P4CLIENT=p4_wworkspace
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>
#
#
# P4_ENV tells PDTool which P4 environment variables need to be set at execution time
P4_ENV=P4CLIENT,P4PORT,P4USER,P4PASSWD,P4EDITOR
# 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=-f
#----------------------------------------------------------------------
#
#======================================================================
#### [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 PDTool 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
#------------------------------------------------------------------------------
```

3. **Modify VCSModule.xml** – located in ../resources/modules directory. The VCSModule.xml serves as the example. Users are welcome to create their own VCSModule.xml copy and point to from their plan file.
    a. This step is applicable for both UNIX and Windows and Command-Line and Ant deployment.
    b. In the VCSModule.xml example below, there are four sample VCS connections, one for each VCS type supported. The XML element name is the same name that is used by the deploy.properties file. For the sake of not duplicating text, please refer to the property types and explanations found in section 3.2 above for each corresponding XML element found below.

```
<?xml version="1.0"?>
<p1:VCSModule xmlns:p1="http://www.dvbu.cisco.com/ps/deploytool/modules">
  <vcsConnections>
    <vcsConnection>
        <id>svn01</id>
        <VCS_TYPE>svn</VCS_TYPE>
        <VCS_BASE_TYPE>SVN</VCS_BASE_TYPE>
        <VCS_HOME>D:/dev/vcs/csvn/bin</VCS_HOME>
        <VCS_COMMAND>svn</VCS_COMMAND>
        <VCS_EXEC_FULL_PATH>true</VCS_EXEC_FULL_PATH>
        <VCS_OPTIONS>--non-interactive --no-auth-cache --trust-server-cert</VCS_OPTIONS>
        <VCS_WORKSPACE_INIT_NEW_OPTIONS></VCS_WORKSPACE_INIT_NEW_OPTIONS>
        <VCS_WORKSPACE_INIT_LINK_OPTIONS></VCS_WORKSPACE_INIT_LINK_OPTIONS>
        <VCS_WORKSPACE_INIT_GET_OPTIONS></VCS_WORKSPACE_INIT_GET_OPTIONS>
       <VCS_BASE_FOLDER_INIT></VCS_BASE_FOLDER_INIT>
        <VCS_CHECKIN_OPTIONS></VCS_CHECKIN_OPTIONS>
        <VCS_CHECKIN_OPTIONS_REQUIRED></VCS_CHECKIN_OPTIONS_REQUIRED>
        <VCS_CHECKOUT_OPTIONS></VCS_CHECKOUT_OPTIONS>
        <VCS_CHECKOUT_OPTIONS_REQUIRED></VCS_CHECKOUT_OPTIONS_REQUIRED>
        <VCS_CIS_IMPORT_OPTIONS>-includeaccess</VCS_CIS_IMPORT_OPTIONS>
        <VCS_CIS_EXPORT_OPTIONS></VCS_CIS_EXPORT_OPTIONS>
      <VCS_REPOSITORY_URL>http:////myhost.company.com/svn/sandbox</VCS_REPOSITORY_URL>
        <VCS_PROJECT_ROOT>cis_objects</VCS_PROJECT_ROOT>
        <VCS_WORKSPACE_HOME>$PROJECT_HOME</VCS_WORKSPACE_HOME>
        <VCS_WORKSPACE_NAME>$VCS_BASE_TYPE$ww</VCS_WORKSPACE_NAME>
        <VCS_WORKSPACE_DIR>$VCS_WORKSPACE_HOME/$VCS_WORKSPACE_NAME</VCS_WORKSPACE_DIR>
        <VCS_TEMP_DIR>$VCS_WORKSPACE_HOME/$VCS_BASE_TYPE$t</VCS_TEMP_DIR>
        <VCS_USERNAME></VCS_USERNAME>
        <VCS_PASSWORD></VCS_PASSWORD>
        <VCS_IGNORE_MESSAGES></VCS_IGNORE_MESSAGES>
        <VCS_MESSAGE_PREPEND></VCS_MESSAGE_PREPEND>
        <!--Element vcsSpecificEnvVars is optional-->
        <vcsSpecificEnvVars>
            <!--Element envVar is optional, maxOccurs=unbounded-->
            <envVar>
                <envName>SVN_EDITOR</envName>
                <envValue>notepad</envValue>
            </envVar>
        </vcsSpecificEnvVars>
    </vcsConnection>
    <vcsConnection>
        <id>perforce01</id>
        <VCS_TYPE>p4</VCS_TYPE>
        <VCS_BASE_TYPE>P4</VCS_BASE_TYPE>
        <VCS_HOME>D:/dev/vcs/perforce</VCS_HOME>
        <VCS_COMMAND>p4</VCS_COMMAND>
        <VCS_EXEC_FULL_PATH>true</VCS_EXEC_FULL_PATH>
        <VCS_OPTIONS></VCS_OPTIONS>
        <VCS_WORKSPACE_INIT_NEW_OPTIONS></VCS_WORKSPACE_INIT_NEW_OPTIONS>
        <VCS_WORKSPACE_INIT_LINK_OPTIONS></VCS_WORKSPACE_INIT_LINK_OPTIONS>
        <VCS_WORKSPACE_INIT_GET_OPTIONS></VCS_WORKSPACE_INIT_GET_OPTIONS>
        <VCS_BASE_FOLDER_INIT></VCS_BASE_FOLDER_INIT>
        <VCS_CHECKIN_OPTIONS></VCS_CHECKIN_OPTIONS>
        <VCS_CHECKIN_OPTIONS_REQUIRED></VCS_CHECKIN_OPTIONS_REQUIRED>
        <VCS_CHECKOUT_OPTIONS></VCS_CHECKOUT_OPTIONS>
        <VCS_CHECKOUT_OPTIONS_REQUIRED></VCS_CHECKOUT_OPTIONS_REQUIRED>
        <VCS_CIS_IMPORT_OPTIONS>-includeaccess</VCS_CIS_IMPORT_OPTIONS>
        <VCS_CIS_EXPORT_OPTIONS></VCS_CIS_EXPORT_OPTIONS>
        <VCS_REPOSITORY_URL>myhost:1666</VCS_REPOSITORY_URL>
        <VCS_PROJECT_ROOT>cis_objects</VCS_PROJECT_ROOT>
        <VCS_WORKSPACE_HOME>$PROJECT_HOME</VCS_WORKSPACE_HOME>
        <VCS_WORKSPACE_NAME>$VCS_BASE_TYPE$ww</VCS_WORKSPACE_NAME>
        <VCS_WORKSPACE_DIR>$VCS_WORKSPACE_HOME/$VCS_WORKSPACE_NAME</VCS_WORKSPACE_DIR>
        <VCS_TEMP_DIR>$VCS_WORKSPACE_HOME/$VCS_BASE_TYPE$t</VCS_TEMP_DIR>
        <VCS_USERNAME></VCS_USERNAME>
        <VCS_PASSWORD></VCS_PASSWORD>
        <VCS_IGNORE_MESSAGES>No files to submit</VCS_IGNORE_MESSAGES>
        <VCS_MESSAGE_PREPEND>SCR:</VCS_MESSAGE_PREPEND>
        <!--Element vcsSpecificEnvVars is optional-->
        <vcsSpecificEnvVars>
            <!--Element envVar is optional, maxOccurs=unbounded-->
            <envVar>
                <envName>P4EDITOR</envName>
                <envValue>notepad</envValue>
            </envVar>
            <envVar>
                <envName>P4CLIENT</envName>
                <envValue>$VCS_WORKSPACE_NAME</envValue>
            </envVar>
            <envVar>
                <envName>P4PORT</envName>
                <envValue>$VCS_REPOSITORY_URL</envValue>
            </envVar>
            <envVar>
                <envName>P4USER</envName>
                <envValue>$VCS_USERNAME</envValue>
            </envVar>
            <envVar>
                <envName>P4PASSWD</envName>
                <envValue>$VCS_PASSWORD</envValue>
            </envVar>
            <envVar>
                <envName>P4DEL_LINK_OPTIONS</envName>
                <envValue>-f</envValue>
            </envVar>
        </vcsSpecificEnvVars>
    </vcsConnection>
    <vcsConnection>
        <id>tfs01</id>
        <VCS_TYPE>tfs2013</VCS_TYPE>
        <VCS_BASE_TYPE>TFS</VCS_BASE_TYPE>
        <VCS_HOME>E:\\dev\\vcs\\TEE-CLC-11.0.0</VCS_HOME>
        <VCS_COMMAND>tf.cmd</VCS_COMMAND>
        <VCS_EXEC_FULL_PATH>true</VCS_EXEC_FULL_PATH>
        <VCS_OPTIONS></VCS_OPTIONS>
       <VCS_WORKSPACE_INIT_NEW_OPTIONS>/location:server</VCS_WORKSPACE_INIT_NEW_OPTIONS>
        <VCS_WORKSPACE_INIT_LINK_OPTIONS></VCS_WORKSPACE_INIT_LINK_OPTIONS>
        <VCS_WORKSPACE_INIT_GET_OPTIONS></VCS_WORKSPACE_INIT_GET_OPTIONS>
        <VCS_BASE_FOLDER_INIT></VCS_BASE_FOLDER_INIT >
        <VCS_CHECKIN_OPTIONS>-associate:1</VCS_CHECKIN_OPTIONS>
        <VCS_CHECKIN_OPTIONS_REQUIRED>-associate</VCS_CHECKIN_OPTIONS_REQUIRED>
        <VCS_CHECKOUT_OPTIONS></VCS_CHECKOUT_OPTIONS>
        <VCS_CHECKOUT_OPTIONS_REQUIRED></VCS_CHECKOUT_OPTIONS_REQUIRED>
        <VCS_CIS_IMPORT_OPTIONS>-includeaccess</VCS_CIS_IMPORT_OPTIONS>
        <VCS_CIS_EXPORT_OPTIONS></VCS_CIS_EXPORT_OPTIONS>
<VCS_REPOSITORY_URL>http:////myhost:8080/tfs/DefaultCollection</VCS_REPOSITORY_URL>
        <VCS_PROJECT_ROOT>DV_70/cis_objects</VCS_PROJECT_ROOT>
        <VCS_WORKSPACE_HOME>$PROJECT_HOME</VCS_WORKSPACE_HOME>
        <VCS_WORKSPACE_NAME>$VCS_BASE_TYPE$ww</VCS_WORKSPACE_NAME>
        <VCS_WORKSPACE_DIR>$VCS_WORKSPACE_HOME/$VCS_WORKSPACE_NAME</VCS_WORKSPACE_DIR>
        <VCS_TEMP_DIR>$VCS_WORKSPACE_HOME/$VCS_BASE_TYPE$t</VCS_TEMP_DIR>
        <VCS_USERNAME></VCS_USERNAME>
        <VCS_PASSWORD></VCS_PASSWORD>
        <VCS_IGNORE_MESSAGES>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>
        <VCS_MESSAGE_PREPEND></VCS_MESSAGE_PREPEND>
        <!--Element vcsSpecificEnvVars is optional-->
        <vcsSpecificEnvVars>
            <!--Element envVar is optional, maxOccurs=unbounded-->
            <envVar>
                <envName>TFS_EDITOR</envName>
                <envValue>notepad</envValue>
            </envVar>
            <envVar>
                <envName>TFS_SERVER_URL</envName>
                <envValue>$$/DV_70/cis_objects</envValue>
            </envVar>
        </vcsSpecificEnvVars>
    </vcsConnection>
  </vcsConnections>


     <vcsResource>
        ...
    </vcsResource>
```

```
</p1:VCSModule>
```

Attributes of Interest for <vcsConnection>:

id – the unique identifier for the <vcsConnection> within the VCSModule.xml file.

VCS_TYPE – The type of VCS being used [git, svn, p4, tfs2005, tfs2010, tfs2012, tfs2013, etc.]. Github=git, Subversion=svn, Perforce=p4, and Team Foundation Server=tfs2005 or [tfs2010, tfs2012, tfs2013]

VCS_BASE_TYPE – Recommended options=[GIT, SVN, P4, or TFS]. Choose to coincide with VCS_TYPE.

VCS_HOME – The 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. Windows example: D:/dev/vcs/csvn/bin. UNIX example: /usr/bin. Use \\ for windows paths such as D:\\dev\\vcs\\csvn\\bin.

VCS_COMMAND – The actual command for the given VCS Type [git,svn,p4,tf.cmd]

VCS_EXEC_FULL_PATH – [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_OPTIONS – The 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_WORKSPACE_INIT_NEW_OPTIONS – Workspace Initialization. For example, VCS_WORKSPACE_INIT_NEW_OPTIONS=/location:server. 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

VCS_WORKSPACE_INIT_LINK_OPTIONS – 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 $/DV_70/cis_objects W:/wks/DV_70/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}

Option: **-IGNORE_INIT_LINK** - This option is used to ignore this step when users have READ-only access and will only be doing check-out from subversion and not check-in. It is a subversion-only option.

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.]

VCS_WORKSPACE_INIT_GET_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 $/DV_70/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}

VCS_BASE_FOLDER_INIT – 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}

.g. E:/dev/vcs/TEE-CLC-11.0.0/tf.cmd add P:/TFSww/DV_70/cis_objects/shared/test00 $/DV_70/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}

VCS_CHECKIN_OPTIONS – Resource Check-in. An example of a check-in option is "-associate:1". Check-in 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}

VCS_CHECKIN_OPTIONS_REQUIRED – A comma separated list of base-level commands that are required for check-in. 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_CHECKOUT_OPTOINS – 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}

VCS_CHECKOUT_OPTIONS_REQUIRED – A comma separated list of base-level commands that are required for checkout. VCS_CHECKOUT_OPTIONS is validated against this list.

VCS_CIS_IMPORT_OPTIONS – VCS Resource Import Options: Occurs during a vcsCheckout when the temporary checkout.car file is imported into the target server. The import is the final step in the process. The backwards compatible option is to include access: -includeaccess. Consult the Archive Module documentation for more options.

VCS_CIS_EXPORT_OPTIONS – VCS Resource Export Options: Occurs during a vcsCheckin when the temporary checkout.car file is exported from the target server. The export from the target is used to compare resources against the VCS repository. The backwards compatible option is leave this option blank. Consult the Archive Module documentation for more options.

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 check-in and checkout in the VCS. The VCS_PROJECT_ROOT also gets used in the folder structure of the local workspace. Use 4 forward slashes http://// to represent the URL so that a single / is escaped. This will result in http:// when resolved.

subversion - The base HTTP URL in subversion

Command Format: [http:////hostname.domain/svn/basename]

Example: http:////myhost.company.com/svn/sandbox/PDTOOL/$CIS_VERSION

perforce - The Repository URL is the host and port in perforce -

Command Format: [hostname:port]

Example: myhost:1666

TFS - The base HTTP URL in Team Foundation Server

Command Format: [http:////hostname.domain:8080/tfs/CollectionName]

Example: http:////myhost:8080/tfs/TeamCollection

VCS_PROJECT_ROOT – VCS_PROJECT_ROOT - This is root name of the project on the VCS Server

subversion - The project name

perforce - The depot name

TFS - The Team Project + folders leading up to where DV metadata folders begin.

VCS_WORKSPACE_HOME – This is the CIS VCS Workspace Home. It is recommended to set the location to PDTool home [e.g. $PROJECT_HOME]. The user does have the flexibility to place the VCS workspace in a location other than PDTool home. [e.g. $APPDATA]

VCS_WORKSPACE_NAME – The name of the workspace folder. This is not a directory but simply a name. The shorter the better. If running PDTool on the same machine as PDToolStudio 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_TYPEww - VCS_TYPE gets evaluated as a variable. "ww" is a string that gets concatentated. Result: svnww

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_BASE_TYPEww]. Use u for UNIX: [VCS_TYPEuw]. Use s for studio: [VCS_BASE_TYPEsw].

VCS_WORKSPACE_DIR – VCS Workspace Dir is a combination of the VCS_WORKSPACE_HOME and a workspace directory name "VCS_WORKSPACE_NAME".

VCS_TEMP_DIR – VCS Temp Dir is a combination of the VCS_WORKSPACE_HOME and a temp dir name such as VCS_BASE_TYPEt.

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_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: ./ExecutePDTool.sh -encrypt ../resources/config/deploy.properties

Windows: ExecutePDTool.bat -encrypt ../resources/config/deploy.properties

VCS_IGNORE_MESSAGES – A comma separated list of messages for the VCS Module to ignore upon execution.

Perforce:	No files to submit
Subversion:	No files to submit,Could not add all targets because some targets are already versioned,Skipping argument: '.svn' ends in a reserved name
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,No arguments matched any files to add

VCS_MESSAGE_PREPEND – A static message that gets prepended onto all check-in or forced check-in messages

vcsSpecificEnvVars – A list of custom environment variables

envVar – envelop for the name value pair.

envName – the name of the environment variable.

envValue – the value of then environment variable. This may contain variables. Use $$ to escape a dollar sign that is not a variable. For example, for the TFS_SERVER_URL use $$/tfs_server_url

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 "deploy.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 (Git, Subversion, TFS, Perforce).

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:

```
deploy.properties
#-------------------
# VCS_WORKSPACE_HOME - This is the CIS VCS Workspace Home.
#     It is recommended to set the location to PDTool home [e.g. $PROJECT_HOME].
#     The user does have the flexibility to place the VCS workspace in a location other than PDTool 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.
#    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 concatentated. Result: svnsw
#    For perforce, make sure all instances of 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=$VCS_BASE_TYPE$ww
VCS_WORKSPACE_DIR=$VCS_WORKSPACE_HOME/$VCS_WORKSPACE_NAME
VCS_TEMP_DIR=$VCS_WORKSPACE_HOME/$VCS_BASE_TYPE$t
```

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

VCSModule.xml
    </vcsConnection>
        <VCS_WORKSPACE_HOME>$PROJECT_HOME</VCS_WORKSPACE_HOME>
        <VCS_WORKSPACE_NAME>$VCS_TYPEww</VCS_WORKSPACE_NAME>
        <VCS_WORKSPACE_DIR>$VCS_WORKSPACE_HOME/$VCS_WORKSPACE_NAME</VCS_WORKSPACE_DIR>
        <VCS_TEMP_DIR>$VCS_WORKSPACE_HOME/$VCS_BASE_TYPE$t</VCS_TEMP_DIR>
    <vcsConnection>
```

Example Locations:

Note: PROJECT_HOME is automatically set by ExecutePDTool script upon execution.

Assumptions

- VCS_WORKSPACE_HOME=c:/deployment/PDTool
- Subversion
  - Workspace: c:/deployment/PDTool/svnww
  - Temp: c:/deployment/PDTool/svnt
- Perforce
  - Workspace: c:/deployment/PDTool/p4ww
  - Temp: c:/deployment/PDTool/p4t
- TFS
  - Workspace: c:/deployment/PDTool/tfsww
  - Temp: c:/deployment/PDTool/tfst

2. Create the Workspace

Within the PDTool, the user has the option of Command Line or Ant execution. The commands are the same. The scripts are located in PDTool/bin directory.

Command Line and Ant Execution:

Windows: *ExecutePDTool.bat -vcsinit -vcsuser user -vcspassword password*

UNIX: *./ExecutePDTool.sh -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 deploy.properties in the properties VCS_USERNAME and VCS_PASSWORD. Use the following commands to encrypt the password

Windows: *ExecutePDTool.bat -encrypt ../resources/config/deploy.properties*

UNIX: *./ExecutePDTool.sh -encrypt ../resources/config/deploy.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>ExecutePDTool.bat -vcsinit

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

 -- COMMAND: vcsInitWorkspace ----------------------

 "C:\Program Files\Java\jdk1.6.0_25\bin\java" -cp
"D:\dev\Workspaces\DeployToolWorkspace\PDTool\dist\*;D:\dev\Workspaces\DeployToolWorkspace\PDTool\l
ib\*" -Dcom
.tibco.ps.configroot="D:\dev\Workspaces\DeployToolWorkspace\PDTool\resources\config" -
Dlog4j.configuration="file:D:\dev\Workspaces\DeployToolWorkspace\CisDeployTo
ol\resources\config\log4j.dp"  com.tibco.ps.deploytool.DeployManagerUtil 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
org.springframework.context.support.FileSystemXmlApplicationContext@4277158a: display name
[org.springframework.context.support.FileSystemXmlApplicationContext@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/plans>
... [...] - <vcsInitWorkspace::MODULE_HOME=
D:/dev/Workspaces/DeployToolWorkspace/PDTool/resources/modules>
... [...] -
<vcsInitWorkspace::SCHEMA_LOCATION=D:/dev/Workspaces/DeployToolWorkspace/PDTool/resources/schema/PD
ToolModules.xsd>
```

```
... [...] - <vcsInitWorkspace::DEBUG1=         false>
... [...] - <vcsInitWorkspace::DEBUG2=         false>
... [...] - <vcsInitWorkspace::DEBUG3=         false>
... [...] - <vcsInitWorkspace::***** BEGIN COMMAND: vcsInitWorkspace *****>
... [...] - <vcsInitWorkspace::>
... [com.tibco.ps.deploytool.DeployManagerUtil] - <Config root
D:\dev\Workspaces\DeployToolWorkspace\PDTool\resources\config>
... [com.tibco.ps.deploytool.DeployManagerUtil] - <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 deploy.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.company.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=
D:/dev/Workspaces/DeployToolWorkspace/PDTool/vcs_svn/cisVcsTemp>
... [...] - <initVCSWorkspace::      VCS_WORKSPACE_PROJECT=
D:/dev/Workspaces/DeployToolWorkspace/PDTool/vcs_svn/cisVcsWorkspace/cis_objects>
... [...] - <initVCSWorkspace::      VCS LifecycleListener=
com.tibco.cmdline.vcs.spi.svn.SVNLifecycleListener>
... [...] - <initVCSWorkspace::>
... [...] - <initVCSWorkspace::---VCS Static Variables:      >
... [...] - <initVCSWorkspace::      VCS_WORKSPACE_NAME=     cisVcsWorkspace>
... [...] - <initVCSWorkspace::      VCS_WORKSPACE_TEMP_NAME=cisVcsTemp>
... [...] - <initVCSWorkspace::>
... [...] - <initVCSWorkspace::    Linking local worksapce to VCS Repository...>
... [...] - <initVCSWorkspace::    VCS Execute Command=D:/dev/vcs/csvn/bin/svn import .
http://kauai.company.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.company.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.company.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.company.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::>

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

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

Step 4.2: Initialize VCS Base Folders

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 DV 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 DV multi-tenant environments where not all tenants will be using version control. Imagine a DV 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 DV 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 DV 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 method "vcsInitializeBaseFolderCheckin" or "vcsInitializeBaseFolderCheckin2" that is used to initialize the base folders. By invoking this method from a deployment plan file directly after the workspace initialization allows the administrator to establish all of the base-level folders in the VCS repository without actually checking in the entire DV repository at one time.

c. How to add intermediate folders

The same method “vcsInitializeBaseFolderCheckin” or “vcsInitializeBaseFolderCheckin2” as 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 DV 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 DV, 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
```

The following line would be configured in the VCS deployment plan:

```
PASS  TRUE  ExecuteAction  vcsInitializeBaseFolderCheckin $SERVERID
"/shared/ORG1/Tenant2,/services/webservices/ORG1/Tenant2,/services/databases/O
RG1/T2CAT[TYPE=CATALOG]/T2SCH[TYPE=SCHEMA]" "" ""
```

In the above example the comma separated list of paths are as follows:

/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: Configure VCS Module XML Configuration File

1. **Modify VCSModule.xml** – located in ../resources/modules directory.

This discussion is referring to the <vcsResource> element and not the <vcsConnections> which was discussed earlier.

The VCSModule XML provides a structure “vcsResource” for “checkin, checkout, forcedCheckin and prepareCheckin”. The global entry point node is called “VCSModule” and contains one or more “vcsResource” nodes. A full description of the PDToolModule XML Schema can be found by reviewing /docs/PDToolModules.xsd.html.

Below is a sample VCS Module XML. Generally speaking, the best practice is to use “vcsCheckout” to checkout high-level project directories from VCS and import into the target CIS server. To that end, you only need to configure 1 “vcsResource” entry for each project path being promoted out of the VCS repository to the target CIS server.

The example below shows 3 major related project areas that include the following:

- Published Data Services (e.g. /services/databases/TEST00)
- Published We Services (e.g. /services/webservices/testWebService00)
- Shared CIS Resources (e.g. /shared/test00)

```
<p1:VCSModule xmlns:p1="http://www.dvbu.cisco.com/ps/deploytool/modules">

    <vcsResource>
        <id>testDB</id>
        <resourcePath>/services/databases/TEST00</resourcePath>
        <resourceType>FOLDER</resourceType>
        <vcsLabel></vcsLabel>
        <revision>HEAD</revision>
        <message>checkin testDB</message>
    </vcsResource>

    <vcsResource>
        <id>testWS</id>
        <resourcePath>/services/webservices/testWebService00</resourcePath>
        <resourceType>FOLDER</resourceType>
        <vcsLabel></vcsLabel>
        <revision>HEAD</revision>
        <message>checkin testWS</message>
    </vcsResource>

    <vcsResource>
```

```
        <id>testNN</id>
        <resourcePath>/shared/test00</resourcePath>
        <resourceType>FOLDER</resourceType>
        <vcsLabel></vcsLabel>
        <revision>HEAD</revision>
        <message>checkin testNN</message>
    </vcsResource>

</p1:VCSModule>
```

Attributes of Interest for <vcsResource>:

id – the unique identifier for the <vcsResource> within the VCSModule.xml file.

resourcePath (optional) – the CIS path to a folder or resource to be used during a VCS operation. The resourcePath is only optional when using vcslabel otherwise it must be provided.

resourceType (optional) – the type of VCS resources pointed to by resourcePath include FOLDER, definitions, link, procedure, table, tree, trigger, data_source, model (6.0+), relationship (6.0+) and policy (6.0+). The resourceType required unless using vcslabel. Also, valid in lieu of FOLDER is container. For ease of use, users may use the “Display Resource Type” found on the info tab of each Studio resource. For example, the display resource type for a DV published web service is “Composite Web Service”. This actually gets translated to the VCS type of “data_source”. The following is a list of how the resource type categories mentioned above that relate to internal resource types and external display types. It should be noted that internal types may be the primary resource type (p) or a sub type (s):

FOLDER

Internal = “CONTAINER (p)”

External Display = "Folder, Web Service Service, Web Service Operations, Web Service Port";

definitions

Internal = “DEFINITION_SET (p), DEFINITIONS (s), SQL_DEFINITION_SET (s), XML_SCHEMA_DEFINITION_SET (s), WSDL_DEFINITION_SET (s), ABSTRACT_WSDL_DEFINITION_SET (s), SCDL_DEFINITION_SET (s)”

External Display = "XML Schema Definition Set, SQL Definition Set, Web Service Definitions ";

procedure

Internal = “PROCEDURE (p), SQL_SCRIPT_PROCEDURE (s), JAVA_PROCEDURE (s), EXTERNAL_SQL_PROCEDURE (s), DATABASE_PROCEDURE (s), BASIC_TRANSFORM_PROCEDURE (s), XSLT_TRANSFORM_PROCEDURE (s), STREAM_TRANSFORM_PROCEDURE (s), XQUERY_TRANSFORM_PROCEDURE (s), OPERATION_PROCEDURE (s)”

External Display = "Script, Basic Transformation, XSLT Transformation, Web Service Operation, Packaged Query, XQuery Transformation, Parameterized Query ";

<u>table (views/tables)</u>

Internal = “TABLE (p), VIEW, SQL_TABLE (s), DATABASE_TABLE (s), DELIMITED_FILE_TABLE (s), SYSTEM_TABLE (s)”

External Display = "View, Table ";

<u>link</u>

Internal = “LINK (p)”

External Display = "Link, Published Resource ";

<u>tree</u>

Internal = “TREE (p), XML_FILE_TREE (s)”

External Display = "Hierarchical ";

<u>trigger</u>

Internal = “TRIGGER (p)”

External Display = "Trigger";

<u>data_source</u>

Internal = “DATA_SOURCE (p)”

External Display = "Data Source, Composite Database, Composite Web Service, Legacy Composite Web Service";

<u>relationship</u>

Internal = “RELATIONSHIP (p)”

External Display = "Relationship";

<u>model</u>

Internal = “MODEL (p)”

External Display = "Model";

<u>policy</u>

Internal = “POLICY (p)”

External Display = "Policy";

Finally, let’s say that the user of PDTool discovers that an exception is being thrown regarding an unknown resource type when using a resource type on the Studio Info tab. The user can “teach” PDTool what VCS Resource Type the Studio Info tab Resource Type belongs to. This is accomplished by using the property “VCSModule_ExternalVcsResourceTypeList” in the deploy.properties file. This provides an externalized mechanism to teach PDTool about new Resource Types and how they are associated with the basic VCS Resource Types discussed previously.

Each Studio Resource contains an info tab with a resource path and a display type. Use the Resource path in the info tab as input into the following Studio Web Service API to discover the CIS Resource Type.
/services/webservices/system/admin/resource/operations/getResource()

The CIS Resource Type is mapped to one of the basic VCS Resource Types provided above. Finally, provide the name value pair in the form of "VCS Resource Type=Studio Display Resource Type".   The name may be repeated more than once if there are multiple resources types for the same resource type category. The following is a potential list of names that can be used to teach PDTool. These names are related to the valid resource type categories:

```
folder=<value>
definitions=<value>
link=<value>
procedure=<value>
table=<value>
tree=<value>
trigger=<value>
data_source=<value>
relationship=<value>
model=<value>
policy=<value>
```

Create a comma separate list of these name=value pair. For example:

```
VCSModule_ExternalVcsResourceTypeList=folder=container, data_source =Composite Database, procedure=Basic Transformation
```

vcslabel (optional) – provides the ability to perform a “checkout” only using a label or tag defined in the VCS.   Currently only the Perforce implementation supports labels. The work to associate resources to a label is performed within the VCS system using VCS commands to assign various resources to the label. PDTool does not assign resources to labels. Note: If using vcslabel then resourcePath and resourceType must be left blank or omitted altogether.

revision (optional) – this value is only used during “checkout”. Typically it is set to HEAD but may also be a revision number. For Team Foundation Server a “T” is used in place of “HEAD”. PDTool will convert “HEAD” to “T” automatically. Never place an “r” in front of a revision number. Always use the number by itself. The number comes from the VCS system and is assigned sequentially during the checkin process. Use a VCS client or VCS browser to browse the VCS repository to determine specific version numbers.

> Note: To promote a specific branch or trunk, determine the revision number associated with that branch or trunk.   Everything has a revision number.

For Subversion check-out the following revision arguments are supported:

NUMBER	revision number
HEAD	latest in repository
BASE	base rev of item's working copy such as the tags directory
COMMITTED	last commit at or before BASE
PREV	revision just before COMMITTED

An example using BASE is shown below:

```
    <vcsResource>
        <id>testNN</id>
        <resourcePath>/shared/test00</resourcePath>
        <resourceType>FOLDER</resourceType>
        <vcsLabel></vcsLabel>
        <revision>BASE</revision>
        <message></message>
    </vcsResource>
```

message (optional) – this string is used during vcsCheckin, vcsForcedCheckin and vcsPrepareCheckin.

Step 6: Configure VCS Deployment Plan (.dp) File

2. Determine if you are configuring ***Command-line plan file*** or ***Ant build file.***

Note: Please note the section on "Module ID Usage" below when preparing either the command-line property file or Ant build file. This section provides guidance on how to configure a line of execution to point to the desired VCS Module identifiers. By decoupling the configuration from the execution, it provides a level of abstraction for the user to be able to pick and choose what a deployment plan looks like without having to change the VCS Module configuration constantly. This should help with ease-of-use.

January 26, 2012 (V2.1) or greater utilizes a new set of methods that allow the user to reference a VCS Connection Id in the VCSModule.xml file. Essentially, the method name is the same with a 2 appended to the end. For example, vcsInitWorkspace becomes vcsInitWorkspace2 and so on. However, the method signature is different. Please refer to the "Module ID Usage" section for more information about each method.

a. Command-line Deployment Plan File

The following section describes how to setup a property file for both command line and Ant and execute the script. This script will use the VCSModule.xml that was described in the previous section.

Properties File (UnitTest-VCS.dp):

Plan File Rules:

```
# ----------------------------
# UnitTest-VCS.dp
# ----------------------------
#   1. All parameters are space separated.  Commas are not used.
#         a. Any number of spaces may occur before or after any parameter and are
trimmed.
#
#   2. Parameters should always be enclosed in double quotes according to these
rules:
#         a. when the parameter value contains a comma separated list:
#                               ANSWER: "ds1,ds2,ds3"
#
#         b. when the parameter value contain spaces or contains a dynamic
variable that will resolve to spaces
#            i.   There is no distinguishing between Windows and Unix variables.
Both UNIX style variables ($VAR) and
#                 and Windows style variables (%VAR%) are valid and will be
parsed accordingly.
#            ii.  All parameters that need to be grouped together that contain
spaces are enclosed in double quotes.
#            iii. All paths that contain or will resolve to a space must be
enclosed in double quotes.
#                 An environment variable (e.g. $MODULE_HOME) gets resolved on
invocation PDTool.
#                       Paths containing spaces must be enclosed in double
quotes:
#                              ANSWER: "$MODULE_HOME/LabVCSModule.xml"
#                       Given that MODULE_HOME=C:/dev/Cis Deploy
Tool/resources/modules, PDTool automatically resolves the variable to
#                       "C:/dev/Cis Deploy
Tool/resources/modules/LabVCSModule.xml".
#
#         c. when the parameter value is complex and the inner value contains
spaces
#                    i. In this example $PROJECT_HOME will resolve to a path that
contains spaces such as C:/dev/Cis Deploy Tool
#                       For example take the parameter -pkgfile
$PROJECT_HOME$/bin/carfiles/testout.car.
#                       Since the entire command contains a space it must be
enclosed in double quotes:
#                              ANSWER: "-pkgfile
$PROJECT_HOME/bin/carfiles/testout.car"
#
#   3. A comment is designated by a # sign preceding any other text.
#         a. Comments may occur on any line and will not be processed.
#
#   4. Blank lines are not processed
#         a. Blank lines are counted as lines for display purposes
#         b. If the last line of the file is blank, it is not counted for display
purposes.
#
```

Plan File Parameters:

```
# ----------------------------
# Parameter Specification:
# ----------------------------
# Param1=[PASS or FAIL]  :: Expected Regression Behavior.  Informs the script
whether you expect the action to pass or fail.  Can be used for regression
testing.
# Param2=[TRUE or FALSE] :: Exit Orchestration script on error
# Param3=Module Batch/Shell Script name to execute (no extension).  Extension is
added by script.
# Param4=Module Action to execute
# Param5-ParamN=Specific space separated parameters for the action.  See Property
Rules below.
```

Plan File Example:

```
# -----------------------------------------
# Begin task definition list:
# -----------------------------------------
# Execute VCS Workspace Initialization
PASS   TRUE   ExecuteAction       vcsInitWorkspace

# ------------------
# Check-out
# ------------------
#
# Reference Resource and Type directly
# No reference to VCSModule.xml
# -----------------------------------------
# Check-out a DV Database Folder
PASS   TRUE   ExecuteAction       vcsCheckout                $SERVERID
/services/databases/TEST00 "Data Source" HEAD "$MODULE_HOME/servers.xml"
# Check-out a web Service Folder
PASS   TRUE   ExecuteAction       vcsCheckout                $SERVERID
/services/webservices/testWebService00 "Composite Web Service" HEAD
       "$MODULE_HOME/servers.xml"
# Check-out a shared Folder
PASS   TRUE   ExecuteAction       vcsCheckout                $SERVERID
/shared/test00 "Folder" HEAD     "$MODULE_HOME/servers.xml"

# Reference VCS Configuration File
# Reference to VCSModule.xml
# ---------------------------------------
# Check-out a DV Database, Web Service and Shared Folder
PASS   TRUE   ExecuteAction       vcsCheckouts               $SERVERID
"testDB,testWS,testNN"
       "$MODULE_HOME/VCSModule.xml" "$MODULE_HOME/servers.xml"
# Check-out a DV Database Folder
PASS   TRUE   ExecuteAction       vcsCheckouts               $SERVERID "testDB"
       "$MODULE_HOME/VCSModule.xml"     "$MODULE_HOME/servers.xml"

# ------------------
# Check-in
# ------------------
#
# Reference Resource and Type directly
# No reference to VCSModule.xml
# -----------------------------------------
# Check-in a DV Database Folder
PASS   TRUE   ExecuteAction       vcsCheckin                $SERVERID
/services/databases/TEST00 "Data Source" "check in"
       "$MODULE_HOME/servers.xml"
# Check-in a DV Web Service Folder
PASS   TRUE   ExecuteAction       vcsCheckin                $SERVERID
/services/webservices/testWebService00 "Composite Web Service" "check in"
       "$MODULE_HOME/servers.xml"
# Check-in a DV Shared Folder
PASS   TRUE   ExecuteAction       vcsCheckin                $SERVERID
/shared/test00 "Folder" "check in"      "$MODULE_HOME/servers.xml"

# Reference VCS Configuration File
# Reference to VCSModule.xml
# ---------------------------------------
# Check-in a DV Database Folder
PASS   TRUE   ExecuteAction       vcsCheckins               $SERVERID "testNN"
       "$MODULE_HOME/VCSModule.xml"     "$MODULE_HOME/servers.xml"
# Check-in a DV Web Service Folder
PASS   TRUE   ExecuteAction       vcsCheckins               $SERVERID "testDB"
       "$MODULE_HOME/VCSModule.xml"     "$MODULE_HOME/servers.xml"
# Check-in a DV Shared Folder
PASS   TRUE   ExecuteAction       vcsCheckins               $SERVERID "testWS"
       "$MODULE_HOME/VCSModule.xml"     "$MODULE_HOME/servers.xml"

# ------------------
# Forced Check-in
# ------------------
#
# Reference Resource and Type directly
# No reference to VCSModule.xml
# -----------------------------------------
# Forced Check-in a DV Database Folder
PASS   TRUE   ExecuteAction       vcsForcedCheckin          $SERVERID
/services/databases/TEST00 "Data Source" "force check in"
       "$MODULE_HOME/servers.xml"
# Forced Check-in a DV Web Service Folder
PASS   TRUE   ExecuteAction       vcsForcedCheckin          $SERVERID
/services/webservices/testWebService00 "Composite Web Service" "force check in"
       "$MODULE_HOME/servers.xml"
# Forced Check-in a DV Shared Folder
PASS   TRUE   ExecuteAction       vcsForcedCheckin          $SERVERID
/shared/test00 "Folder" "force check in"      "$MODULE_HOME/servers.xml"

# Reference VCS Configuration File
# Reference to VCSModule.xml
# ---------------------------------------
# Forced Check-in a DV Database Folder
```

```
PASS   TRUE   ExecuteAction       vcsForcedCheckins          $SERVERID "testNN"
       "$MODULE_HOME/VCSModule.xml"     "$MODULE_HOME/servers.xml"
# Forced Check-in a DV web Service Folder
PASS   TRUE   ExecuteAction       vcsForcedCheckins          $SERVERID "testDB"
       "$MODULE_HOME/VCSModule.xml"     "$MODULE_HOME/servers.xml"
# Forced Check-in a DV Shared Folder
PASS   TRUE   ExecuteAction       vcsForcedCheckins          $SERVERID "testWS"
       "$MODULE_HOME/VCSModule.xml"     "$MODULE_HOME/servers.xml"

# ------------------
# Prepare Check-in
# ------------------
#
# Reference Resource and Type directly
# No reference to VCSModule.xml
# ----------------------------------------------
# Prepare Check-in for a DV Shared Folder
PASS   TRUE   ExecuteAction       vcsPrepareCheckin          $SERVERID
/shared/test00 "Folder"    "$MODULE_HOME/servers.xml"

# Reference VCS Configuration File
# Reference to VCSModule.xml
# ---------------------------------------------
# Prepare Check-in for a DV Shared Folder
PASS   TRUE   ExecuteAction       vcsPrepareCheckins         $SERVERID "testNN"
       "$MODULE_HOME/VCSModule.xml"     "$MODULE_HOME/servers.xml"
```

b. Ant Build File

<u>Build File (build-VCS.xml):</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="PDTool" default="default" basedir=".">

  <description>description</description>

  <!-- Default properties -->
  <property name="SERVERID"                 value="localhost"/>
  <property name="noarguments"              value="&quot;&quot;"/>

  <!-- Default Path properties -->
  <property name="RESOURCE_HOME"            value="${PROJECT_HOME}/resources"/>
  <property name="MODULE_HOME"              value="${RESOURCE_HOME}/modules"/>
  <property name="pathToServersXML"         value="${MODULE_HOME}/servers.xml"/>
  <property name="pathToArchiveXML"         value="${MODULE_HOME}/ArchiveModule.xml"/>
  <property name="pathToDataSourcesXML"     value="${MODULE_HOME}/DataSourceModule.xml"/>
  <property name="pathToGroupsXML"          value="${MODULE_HOME}/GroupModule.xml"/>
  <property name="pathToPrivilegeXML"       value="${MODULE_HOME}/PrivilegeModule.xml"/>
  <property name="pathToRebindXML"          value="${MODULE_HOME}/RebindModule.xml"/>
  <property name="pathToRegressionXML"      value="${MODULE_HOME}/RegressionModule.xml"/>
  <property name="pathToResourceXML"        value="${MODULE_HOME}/ResourceModule.xml"/>
  <property name="pathToResourceCacheXML"
        value="${MODULE_HOME}/ResourceCacheModule.xml"/>
  <property name="pathToServerAttributeXML"
        value="${MODULE_HOME}/ServerAttributeModule.xml"/>
  <property name="pathToTriggerXML"          value="${MODULE_HOME}/TriggerModule.xml"/>
  <property name="pathToUsersXML"            value="${MODULE_HOME}/UserModule.xml"/>
  <property name="pathToVCSModuleXML"        value="${MODULE_HOME}/VCSModule.xml"/>

  <!-- Custom properties -->
  <property name="vcsIds"                    value="testNN"/>

  <!-- Default Classpath [Do Not Change] -->
  <path id="project.class.path">
        <fileset dir="${PROJECT_HOME}/lib"><include name="**/*.jar"/></fileset>
        <fileset dir="${PROJECT_HOME}/dist"><include name="**/*.jar"/></fileset>
        <fileset dir="${PROJECT_HOME}/ext/ant/lib"><include name="**/*.jar"/></fileset>
  </path>

  <taskdef name="executeJavaAction" description="Execute Java Action"
classname="com.tibco.ps.deploytool.ant.CompositeAntTask"
classpathref="project.class.path"/>

  <!-- =================================
       target: default
    ================================= -->
  <target name="default" description="Update CIS with environment specific parameters">

     <!-- Execute Line Here -->

        <executeJavaAction action="vcsCheckout"
       arguments="${SERVERID}^${vcsIds}^${pathToVCSModuleXML}^${pathToServersXML}^${VCS_USE
RNAME}^${VCS_PASSWORD}"       endExecutionOnTaskFailure="TRUE"/>

     <!-- Windows or UNIX: Entire list of actions

       <!-- Initialize Workspace -->
        <executeJavaAction action="vcsInitWorkspace"
       arguments="${VCS_USERNAME}^${VCS_PASSWORD}"  endExecutionOnTaskFailure="TRUE"/>

       <!-- Check-out -->
               <executeJavaAction action="vcsCheckout"
       arguments="${SERVERID}^/services/databases/TEST00^Folder^HEAD^${pathToServersXML}^${
VCS_USERNAME}^${VCS_PASSWORD}"       endExecutionOnTaskFailure="TRUE"/>
               <executeJavaAction action="vcsCheckout"
       arguments="${SERVERID}^/services/webservices/testWebService00^Folder^HEAD^${pathToSe
rversXML}^${VCS_USERNAME}^${VCS_PASSWORD}"   endExecutionOnTaskFailure="TRUE"/>
               <executeJavaAction action="vcsCheckout"
       arguments="${SERVERID}^/shared/test00^Folder^HEAD^${pathToServersXML}^${VCS_USERNAME
}^${VCS_PASSWORD}"    endExecutionOnTaskFailure="TRUE"/>

               <executeJavaAction action="vcsCheckouts"
       arguments="${SERVERID}^${vcsIds}^${pathToVCSModuleXML}^${pathToServersXML}^${VCS_USE
RNAME}^${VCS_PASSWORD}"       endExecutionOnTaskFailure="TRUE"/>

       <!-- Check-in -->
```

```
               <executeJavaAction action="vcsCheckin"
        arguments="${SERVERID}^/services/databases/TEST00^Folder^check in
database^${pathToServersXML}^${VCS_USERNAME}^${VCS_PASSWORD}"
        endExecutionOnTaskFailure="TRUE"/>
               <executeJavaAction action="vcsCheckin"
        arguments="${SERVERID}^/services/webservices/testWebService00^Folder^check in web
service^${pathToServersXML}^${VCS_USERNAME}^${VCS_PASSWORD}"
        endExecutionOnTaskFailure="TRUE"/>
               <executeJavaAction action="vcsCheckin"
        arguments="${SERVERID}^/shared/test00^check in shared
test00^Folder^${pathToServersXML}^${VCS_USERNAME}^${VCS_PASSWORD}"
        endExecutionOnTaskFailure="TRUE"/>


        <executeJavaAction action="vcsCheckins"
        arguments="${SERVERID}^${vcsIds}^${pathToVCSModuleXML}^${pathToServersXML}^${VCS_USE
RNAME}^${VCS_PASSWORD}"       endExecutionOnTaskFailure="TRUE"/>

        <!-- Forced Check-in -->
               <executeJavaAction action="vcsForcedCheckin"
        arguments="${SERVERID}^/services/databases/TEST00^Folder^force check in
database^${pathToServersXML}^${VCS_USERNAME}^${VCS_PASSWORD}"
        endExecutionOnTaskFailure="TRUE"/>
               <executeJavaAction action="vcsForcedCheckin"
        arguments="${SERVERID}^/services/webservices/testWebService00Folder^^force check in
web service^${pathToServersXML}^${VCS_USERNAME}^${VCS_PASSWORD}"
        endExecutionOnTaskFailure="TRUE"/>
               <executeJavaAction action="vcsForcedCheckin"
        arguments="${SERVERID}^/shared/test00^Folder^force check in shared
test00^${pathToServersXML}^${VCS_USERNAME}^${VCS_PASSWORD}"
        endExecutionOnTaskFailure="TRUE"/>


        <executeJavaAction action="vcsForcedCheckins"
        arguments="${SERVERID}^${vcsIds}^${pathToVCSModuleXML}^${pathToServersXML}^${VCS_USE
RNAME}^${VCS_PASSWORD}"       endExecutionOnTaskFailure="TRUE"/>

        <!-- Prepare Check-in -->
               <executeJavaAction action="vcsPrepareCheckin"
        arguments="${SERVERID}^/shared/test00^Folder^${pathToServersXML}^${VCS_USERNAME}^${V
CS_PASSWORD}"  endExecutionOnTaskFailure="TRUE"/>
        <executeJavaAction action="vcsPrepareCheckins"
        arguments="${SERVERID}^${vcsIds}^${pathToVCSModuleXML}^${pathToServersXML}^${VCS_USE
RNAME}^${VCS_PASSWORD}"       endExecutionOnTaskFailure="TRUE"/>
    -->
        </target>
</project>
```

Module ID Usage:

The following explanation provides a general pattern for module identifiers. The module identifier for this module is “vcsIds”.

- Possible values for the module identifier:
- 1. ***Inclusion List*** - CSV string like “id1,id2”
    - PDTool will process only the passed in identifiers in the specified module XML file.

Example command-line property file

```
PASS       TRUE   ExecuteAction       vcsCheckouts        $SERVERID "vcs1,vcs2"
           "$MODULE_HOME/LabVCSModule.xml" "$MODULE_HOME/servers.xml"
```

Example Ant build file

```
        <executeJavaAction action="vcsCheckouts"
   arguments="${SERVERID}^vcs1,vcs2^${pathToVCSModuleXML}^${pathToServersXML}^
${VCS_USERNAME}^${VCS_PASSWORD}"  endExecutionOnTaskFailure="TRUE"/>
```

- 2. ***Process All*** - '*' or whatever is configured to indicate all resources
  - PDTool will process all resources in the specified module XML file.

Example command-line property file

```
PASS       TRUE   ExecuteAction       vcsCheckouts        $SERVERID "*"
   "$MODULE_HOME/LabVCSModule.xml" "$MODULE_HOME/servers.xml"
```

Example Ant build file

```
        <executeJavaAction action="vcsCheckouts"
   arguments="${SERVERID}^*^${pathToVCSModuleXML}^${pathToServersXML}^
${VCS_USERNAME}^${VCS_PASSWORD}"  endExecutionOnTaskFailure="TRUE"/>
```

- 3. ***Exclusion List*** - CSV string with '-' or whatever is configured to indicate exclude resources as prefix like “-id1,id2”
  - PDTool will ignore passed in resources and process the rest of the identifiers in the module XML file.

Example command-line property file

```
PASS       TRUE   ExecuteAction       vcsCheckouts        $SERVERID "-
vcs1,vcs2"              "$MODULE_HOME/LabVCSModule.xml"
"$MODULE_HOME/servers.xml"
```

Example Ant build file

```
        <executeJavaAction action="vcsCheckouts"
   arguments="${SERVERID}^-vcs1,vcs2^${pathToVCSModuleXML}^${pathToServersXML}^
${VCS_USERNAME}^${VCS_PASSWORD}"  endExecutionOnTaskFailure="TRUE"/>
```

Step 7: Test the Configuration (Script Execution)

The full details on property file setup and script execution can be found in the document “*PDTool User's Guide.pdf*”. The abridged version is as follows:

2. Determine if you are configuring ***Command-line property file*** or ***Ant build file.***

   a. Command-line script execution

   Windows: ExecutePDTool.bat -exec ../resources/plans/UnitTest-VCS.dp [-vcsuser user] [-vcspassword password]

   Unix: ./ExecutePDTool.sh -exec ../resources/plans/UnitTest-VCS.dp [-vcsuser user] [-vcspassword password]

   b. Ant script execution

   Windows: ExecutePDTool.bat -ant ../resources/ant/build-VCS.xml [-vcsuser user] [-vcspassword password]

Unix: ./ExecutePDTool.sh -ant ../resources/ant/build-VCS.xml [-vcsuser user] [-vcspassword password]

3. Explore VCS Module common scenarios:

Example Scenario 1 – Checkout using resourcePath:

Description:

Perform VCS Checkout from Subversion repository and import into the Target CIS server. For example, the DEV server was used to checkin to Subversion. The checkout will be used to promote from DEV to TEST using VCS.

XML Configuration Sample:

VCS Module XML provides three nodes whereby the three major parts of CIS will be checked out. /services/databases, /services/webservices and /shared.

```
<p1:VCSModule xmlns:p1="http://www.dvbu.cisco.com/ps/deploytool/modules">

    <vcsResource>
        <id>testDB</id>
        <resourcePath>/services/databases/TEST00</resourcePath>
        <resourceType>FOLDER</resourceType>
        <revision>HEAD</revision>
        <message>checkin testDB</message>
    </vcsResource>
    <vcsResource>
        <id>testWS</id>
        <resourcePath>/services/webservices/testWebService00</resourcePath>
        <resourceType>FOLDER</resourceType>
        <revision>HEAD</revision>
        <message>checkin testWS</message>
    </vcsResource>
    <vcsResource>
        <id>testNN</id>
        <resourcePath>/shared/test00</resourcePath>
        <resourceType>FOLDER</resourceType>
        <revision>HEAD</revision>
        <message>checkin testNN</message>
    </vcsResource>
</p1:VCSModule>
```

Execution Sample:

Unix: ./ExecutePDTool.sh -exec ../resources/plans/UnitTest-VCS.dp
Property file setup for UnitTest-VCS.dp:

```
# -----------------------------------------
# Begin task definition list for UNIX:
# -------------------------------------------
# VCS actions
# Initialize the workspace
#PASS  TRUE   ExecuteAction       vcsInitWorkspace

# checkout the following
# testDB=/ services/databases/TEST00
# testWS=/services/webservices/ testWebService00
# testNN=/shared/test00
PASS   TRUE   ExecuteAction  vcsCheckouts   $SERVERID "testDB,testWS,testNN"
       "$MODULE_HOME/LabVCSModule.xml" "$MODULE_HOME/servers.xml"

# NOTE:  THESE LINES ARE COMMENTED OUT:

# Checkin /shared/test00
#PASS  TRUE   ExecuteAction       vcsCheckins   $SERVERID "testNN"
              "$MODULE_HOME/LabVCSModule.xml" "$MODULE_HOME/servers.xml"
# Force checkin /shared/test00
#PASS  TRUE   ExecuteAction       vcsForcedCheckins $SERVERID "testNN"
              "$MODULE_HOME/LabVCSModule.xml" "$MODULE_HOME/servers.xml"
# Prepare checkin /shared/test00
#PASS  TRUE   ExecuteAction       vcsPrepareCheckins $SERVERID "testNN"
              "$MODULE_HOME/LabVCSModule.xml" "$MODULE_HOME/servers.xml"
```

Results Expected:

The directory structures identified by testDB, testWS and testNN are checked out and imported into the target CIS server.

5 Version Control Specific Information

Subversion specific information

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

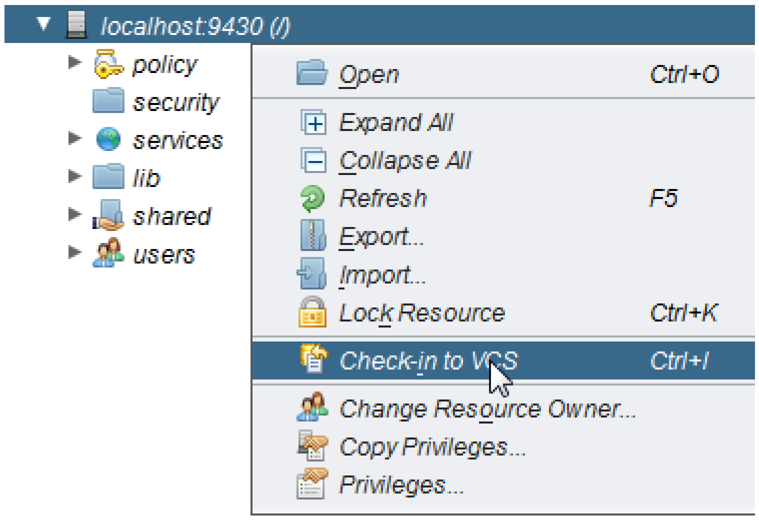
Perforce specific information

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

1. Running PDTool on the same machine.
    a. Make sure the workspace name "VCS_WORKSPACE_NAME" is configured with a different name such as p4_wworkspace for PDTool.
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 PDTool 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 PDTool be done from the DV 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 PDTool 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. Perforce Best Practices for assigning CIS resources to Perforce labels:

i. Always assign the CIS base structures (root.cmf, databases.cmf, webservices.cmf, and shared.cmf)

ii. The perforce 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 PDTool 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.

GIT specific information

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

For all forms of the method vcsCheckin including vcsCheckins, vcsCheckin2 and vcsCheckins2, they have been redirected to vcsForcedCheckin due to circumstances where a similar concept of check in to SVN could not be ascertained for GIT. Therefore, the most expedient resolution was to use the already working vcsForcedCheckin. The workflow for both vcsCheckin and vcsForcedChecking are shown below.

The main difference is that vcsForcedCheckin performs the check out from GIT first to synchronize the workspace and then performs the diff merger to add/delete resources with GIT followed by a check in to GIT.

Given the use case for PDTool, this was an acceptable resolution to this issue as PDTool is used for deployment and not typically for check in. If it is used for check in, then synchronizing the workspace first before check in is acceptable.

vcsCheckin: The check-in path performs the following steps:

1) Export from CIS into temp dir
2) Diffmerger to do a GIT add/delete
3) Check out [this was wiping out the add/delete]
4) Validate paths
5) Check in to GIT [commit and push]

vcsForcedCheckin: The vcs forced check-in performs the following steps:

1) Check out from GIT
2) Validate paths
3) Export from CIS into temp dir
4) Diffmerger to do a GIT add/delete
5) Check in to GIT [commit and push]

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 PDTool 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 PDTool 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, PDTool'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 deprecated. 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 and higher if the CIS project is in a separate collection then the collection is named in the server URL path and the project path will be part of VCS_REPOSITORY_URL. E.g. https:////hostname:8080/tfs/CollectionName. The TFS_SERVER_URL will then be set to $/path/to/the/project

   c. Provide examples of what is in TFS directory structure and how PDTool 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/DefaultCollection
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=TeamProject/DataFoundation/Development/Databases/CIS
VCS_REPOSITORY_URL=http:////host:8080/tfs/CollectionName
TFS_SERVER_URL=$$/TeamProject/DataFoundation/Development/Databases/CIS
```

d. Property that allows the user to control check-in from TFS Team Explorer. This is controlled via the deploy.properties file.

```
# Use an existing TFS workspace.  This will not create a separate workspace for
# VCS integration. If this is true, CIS will only copy to/from the workspace and
# not check into TFS. It will record add, edits and deletes in the workspace.
# This allows for tools like Visual Studio to perform the actual check-in to TFS.
# This environment variable can be set in deploy.properties.
TFS_USE_EXISTING_WORKSPACE=false
#
# TFS_ENV tells PDTool which TFS environment variables need to be set at execution
time
TFS_ENV=TFS_EDITOR,TFS_USE_EXISTING_WORKSPACE
```

6. VCS Module XML

a. Same topics as above as this is just a different mechanism to achieve the same goal.

In VCS Module the newly introduced variables TFS_SERVER_URL will be defined in “envVar” section as shown below.

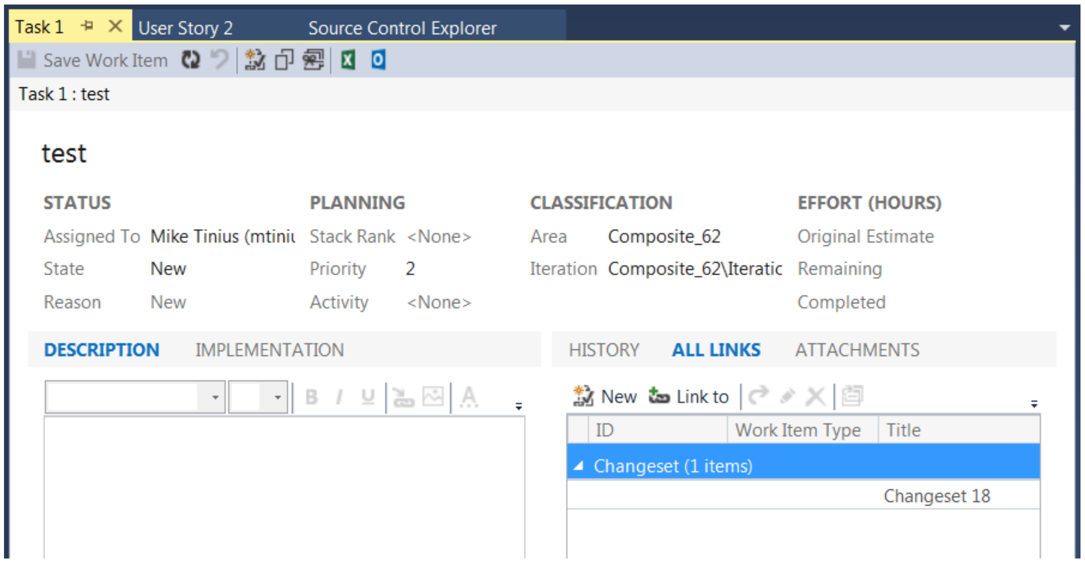
```
<vcsSpecificEnvVars>
 <!--Element envVar is optional, maxOccurs=unbounded -->
       <!-- Default-change if desired but must be in path (UNIX:vi, Windows: notepad) ] -->
 <envVar>
  <envName>TFS_EDITOR</envName>
  <envValue>notepad</envValue>
 </envVar>
      <!-- Use an existing TFS workspace.  This will not create a separate workspace for VCS integration. If this is
      true, CIS will only copy to/from the workspace and not check into TFS. It will record add, edits and deletes in
      the workspace. -->
 <envVar>
  <envName>TFS_USE_EXISTING_WORKSPACE</envName>
  <envValue>false</envValue>
 </envVar>
 <envVar>
  <envName>TFS_SERVER_URL</envName>
  <envValue>$$/$VCS_PROJECT_ROOT</envValue>
 </envVar>
      </vcsSpecificEnvVars>
```

Please note: When character '$' is part of the configuration variable as in the TFS Server URL then it should be escaped by repeating it so it that PDTool does not think that it is a variable reference and try to expand it. Therefore it should be $$.

7. Associate a Work Item with a TFS Task

    a. 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 DV and then checks in code. From a PDTool 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 and there is a corresponding <VCS_CHECKIN_OPTIONS> in the VCSModule.xml file where the -associate:<id> command line option can be placed.

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



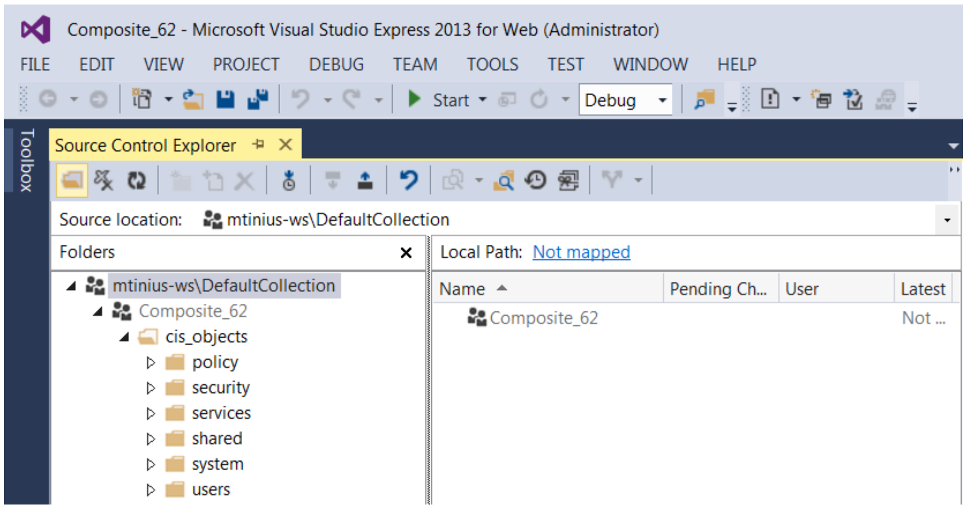
Assigned To	Mike Tinius (mtiniu	Stack Rank	<None>	Area	Composite_62	Original Estimate
State	New	Priority	2	Iteration	Composite_62\Iteratic	Remaining
Reason	New	Activity	<None>			Completed

ID	Work Item Type	Title
Changeset (1 items)		
		Changeset 18

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 DV [TFS Admin]
    - E.g. DefaultCollection
    - Need to create “Team Project” for DV [TFS Admin]
        - E.g. DV_70
    - Define your folder structure
        - /cis_objects which will hold the DV repository objects.
        - As shown in the diagram below, the DV repository is checked in at the root level of DV and contains several sub-folders including: /policy, /security, /services, /shared, /system and /users.


Composite_62 - Microsoft Visual Studio Express 2013 for Web (Administrator)
FILE EDIT VIEW PROJECT DEBUG TEAM TOOLS TEST WINDOW HELP
Start
Debug
Toolbox
Source Control Explorer
Source location: mtinius-ws\DefaultCollection
Folders
Local Path: Not mapped
mtinius-ws\DefaultCollection
Composite_62
cis_objects
policy
security
services
shared
system
users
Name
Pending Ch...
User
Latest
Composite_62
Not ...

- Need to create a DV group (AD Group) and assign permission
    - For PDTool, this is a group that will be responsible for doing deployments.
- Needed to add an AD user to the group
    - For PDTool, 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 DV 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 PDTool
    - 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 DV_70/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=DV_70/cis_objects
        - TFS_SERVER_URL=$$/DV_70/cis_objects
            - When not set, errors occurred during initialization
            - C:/PDTool/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 $
    - The VCSModule.xml would be configured as follows:
        - <VCS_REPOSITORY_URL>http:////tfs_host:8080/tfs/DefaultCollection</VCS_REPOSITORY_URL>
        - <VCS_PROJECT_ROOT>DV_70/cis_objects</VCS_PROJECT_ROOT>
        - <vcsSpecificEnvVars>

```
    <!--Element envVar is optional, maxOccurs=unbounded-->
    <envVar>
      <envName>TFS_EDITOR</envName>
      <envValue>notepad</envValue>
    </envVar>
    <envVar>
      <envName>TFS_SERVER_URL</envName>
      <envValue>$$/DV_70/cis_objects</envValue>
    </envVar>
</vcsSpecificEnvVars>
```

6 VCS Module Definition

Method Definitions and Signatures (Original maintain backward compatibility)

General Notes: The arguments pathToVCSXML and pathToServersXML will be located in PDTool/resources/modules. The value passed into the methods will be the fully qualified path. The paths get resolved when executing the property file and evaluating the $MODULE_HOME variable

1. ***vcsInitWorkspace***

   Initialize the VCS local workspace on the deployment server by linking a local folder with a VCS repository project and checking out all the resources from the VCS repository into the local workspace folder.

```
@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

public void vcsInitWorkspace(String vcsUser, String vcsPassword) throws
CompositeException
```

2. ***vcsInitializeBaseFolderCheckin***

   Initialize base folder check-in from the local workspace to the VCS repository. This provides an alternative way to establish the DV repository base folders into the VCS repository without checking in the entire DV repository. This can be useful for multi-tenant environments where only certain folders will be held under version control. The issue is that all the base-level folders must first be checked in into the VCS prior to any user-level folders being checked in. This method uses the deployment configuration property file "deploy.properties" for VCS connection properties.

   This method will perform a workspace initialization first followed by the base folder checkin.

```
@param customPathList - a comma separated list of paths that are added to
the base paths of /shared or /services/databases or
/services/webservices.  These paths and their corresponding .cmf file
will be created during initialization of the workspace and VCS
repository. This strategy provides a way to accommodate a series of
folders as part of the base-level check-in to insure that all
intermediate folders are checked in leading up the actual folders that
the user is interested in.   If these base-level, intermediate folders
are not checked in, the check-out process will not work properly.  There
must be a .cmf file for each DV resource including folders.  The
following line demonstrates the mandatory requirement for how a catalog
or schema is identified.  The reason for this is that the position
following the database name may be either a catalog or schema.  The
```

```
[TYPE=DATABASE] is optional since the position of the database always
follows /services/databases.

/services/databases/MyDb[TYPE=DATABASE]/MyCat[TYPE=CATALOG]/MySch[TYPE=SC
HEMA]"

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

public void vcsInitializeBaseFolderCheckin(String customPathList, String
vcsUser, String vcsPassword) throws CompositeException
```

3. **vcsCheckout**

Check out changes from the VCS repository and import the differences into the CIS server. If folders are present in CIS that are not present in the repository, those folders will be deleted in CIS. This method does not reference VCSModule.xml but contains all parameters in the call.

```
@param serverId - target server name

@param vcsResourcePath - the actual CIS resource path (not encoded)

@param vcsResourceType - the resource type

@param vcsRevision - the revision from the VCS (HEAD or an integer value
representing the revision number.)

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

public void vcsCheckout(String serverId, String vcsResourcePath, String
vcsResourceType, String vcsRevision , String pathToServersXML, String
vcsUser, String vcsPassword) throws CompositeException;
```

4. **vcsCheckout (overloaded version for vcsLabel)**

Check out changes from the VCS repository and import the differences into the CIS server using VCS labels. If folders are present in CIS that are not present in the repository, those folders will be deleted in CIS. This method does not reference VCSModule.xml but contains all parameters in the call.

```
@param serverId - target server name

@param vcsResourcePath - the actual CIS resource path (not encoded)

@param vcsResourceType - the resource type

@param vcsLabel - the VCS label that associates an entire release of CIS
resources together.  If not null, then vcsResourcePath and
vcsResourceType must be null otherwise vcsResourcePath takes precedence.
```

```
@param vcsRevision - the revision from the VCS (HEAD or an integer value
representing the revision number.)

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

public void vcsCheckout(String serverId, String vcsResourcePath, String
vcsResourceType, String vcsLabel, String vcsRevision , String
pathToServersXML, String vcsUser, String vcsPassword) throws
CompositeException;
```

5. **vcsCheckouts**

Check out changes from the repository and import the differences into the CIS server. If folders are present in CIS that are not present in the repository, those folders will be deleted in CIS.

```
@param serverId - target server name

@param vcsIds - comma separated list of VCS identifiers.

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

public void vcsCheckout(String serverId, String vcsIds, String
pathToVcsXML, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

6. **vcsCheckin**

Check in the changes from the local workspace to the VCS repository. This method does not reference VCSModule.xml but contains all parameters in the call.

Note: For GIT only, vcsCheckin is redirected to vcsForcedCheckin.

```
@param serverId - target server name

@param vcsResourcePath - the actual CIS resource path (not encoded)

@param vcsResourceType - the resource type

@param vcsMessage - the message that describes the checkin

@param pathToServersXML - path to the server values XML

@return void
```

```
@throws CompositeException

public void vcsCheckin(String serverId, String vcsResourcePath, String
vcsResourceType, String vcsMessage, String pathToServersXML, String
vcsUser, String vcsPassword) throws CompositeException;
```

7. **vcsCheckins**

   Check in the changes from the local workspace to the VCS repository.

   Note: For GIT only, vcsCheckins is redirected to vcsForcedCheckin.

```
@param serverId - target server name

@param vcsIds - comma separated list of VCS identifiers.

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@return void

@throws CompositeException

public void vcsCheckin(String serverId, String vcsIds, String
pathToVcsXML, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

8. **vcsForcedCheckin**

   Force check in based on what is in the local workspace for CIS vs. the repository. Overwrite differences in the VCS repository favor of what is in the CIS server. This method does not reference VCSModule.xml but contains all parameters in the call.

```
@param serverId - target server name

@param vcsResourcePath - the actual CIS resource path (not encoded)

@param vcsResourceType - the resource type

@param vcsMessage - the message that describes the checkin

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException


public void vcsForcedCheckin(String serverId, String vcsResourcePath,
String vcsResourceType, String vcsMessage, String pathToServersXML,
String vcsUser, String vcsPassword) throws CompositeException;
```

9. **vcsForcedCheckins**

   Force check in based on what is in the local workspace for CIS vs. the repository. Overwrite differences in the VCS repository favor of what is in the CIS server.

```
@param serverId - target server name

@param vcsIds - comma separated list of VCS identifiers.

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

public void vcsForcedCheckin(String serverId, String vcsIds, String
pathToVcsXML, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

***10.* vcsPrepareCheckin**

Prepare check in by updating the local workspace copy and comparing with VCS but don't check in. This method does not reference VCSModule.xml but contains all parameters in the call.

```
@param serverId - target server name

@param vcsResourcePath - the actual CIS resource path (not encoded)

@param vcsResourceType - the resource type

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

public void vcsPrepareCheckin(String serverId, String vcsResourcePath,
String vcsResourceType, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

***11.* vcsPrepareCheckins**

Prepare check in by updating the local workspace copy and comparing with VCS but don't check in.

```
@param serverId - target server name

@param vcsIds - comma separated list of VCS identifiers.

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line
```

```
@return void

@throws CompositeException

public void vcsPrepareCheckin(String serverId, String vcsIds, String
pathToVcsXML, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

12. vcsScanPathLength

This method handles scanning the DV path and searching for encoded paths that equal or exceed the windows 259 character limit. If found this routine reports those paths. The 259 character limit is only a limitation for windows-based implementations of VCS like TFS. Subversion does not have this issue. This is useful if the developers want to see if there will be any path issues prior to checking in the resources. The length of the folder is calculated by summing the length of VCS_WORKSPACE + VCS_PROJECT_ROOT + CIS_ENCODED_FOLDER.

The parameter “vcsMaxPathLength” can be used to specify what the max length to report on. If a developer wants to run this with say 230 then it will report on any encoded paths are longer than 230 characters. This way they can determine if they have any paths that are getting close to the absolute limit of 259 characters.

Example:

```
VCS_WORKSPACE=P:/TFSsw                                  Len=8
VCS_PROJECT_ROOT=/DV_70/cis_objects          Len=25
CIS_FOLDER=/shared/test00_longpath/SVN/PATH-this is a very long path with spaces in it and is
meant/FILE-this is a very long file name with spaces in it
CIS_ENCODED_FOLDER=/shared/test00__longpath/SVN/PATH-
this_0020is_0020a_0020very_0020long_0020path_0020with_0020spaces_0020in_0020it_0020and_
0020is_0020meant/FILE-
this_0020is_0020a_0020very_0020long_0020file_0020name_0020with_0020spaces_0020in_0020it
                                                        Len=230
Total Path Length=263 which exceeds 259 windows max.
```

Deployment Plan File Entry:

```
PASS     TRUE     ExecuteAction      vcsScanPathLength $SERVERID 259
"/shared/test00_longpath,/services/databases/TEST00" "$MODULE_HOME/servers.xml"
```

Sample output:

```
<---------------------------------------------->
<CONFIG_PROPERTY_FILE=deploy.properties>
<---------------------------------------------->
<vcsScanPathLength::***** BEGIN COMMAND *****>
<vcsScanPathLength::>
<vcsScanPathLength::>
<vcsScanPathLength::---VCS Module ID Arguments:>
<vcsScanPathLength::      vcsResourcePathList=           /shared/test00_longpath,/services/databases/TEST00>
<vcsScanPathLength::      VCS_WORKSPACE_PROJECT=         P:/TFSww/DV_70/cis_objects>
<vcsScanPathLength::      VCS_WORKSPACE_PROJECT Length=   33>
<vcsScanPathLength::      Max Windows Path Len=          259>
<vcsScanPathLength::>
<Resource exists? [true] /shared/test00_longpath on server localhost9430http>
<vcsScanPathLength::>
<vcsScanPathLength::CIS resource to scan:  type=CONTAINER  path=/shared/test00_longpath>
<vcsScanPathLength::      TotalPathLen=263   EncodedPathLen=230   ResourceType=PROCEDURE
EncodedPath=/shared/test00__longpath/SVN/PATH-
```

```
this_0020is_0020a_0020very_0020long_0020path_0020with_0020spaces_0020in_0020it_0020and_0020is_0020meant/FIL
E-this_0020is_0020a_0020very_0020long_0020file_0020name_0020with_0020spaces_0020in_0020it
*****OriginalPath*****=/shared/test00_longpath/SVN/PATH-this is a very long path with spaces in it and is meant/FILE-this is
a very long file name with spaces in it>
<vcsScanPathLength::      TotalPathLen=299    EncodedPathLen=266   ResourceType=PROCEDURE
EncodedPath=/shared/test00__longpath/SVN/PATH-
this_0020is_0020a_0020very_0020long_0020path_0020with_0020spaces_0020in_0020it_0020and_0020is_0020meant_00
20to_0020exceed_0020the_0020limit/FILE-
this_0020is_0020a_0020very_0020long_0020file_0020name_0020with_0020spaces_0020in_0020it
*****OriginalPath*****=/shared/test00_longpath/SVN/PATH-this is a very long path with spaces in it and is meant to exceed
the limit/FILE-this is a very long file name with spaces in it>
<vcsScanPathLength::>
<Resource exists? [true] /services/databases/TEST00 on server localhost9430http>
<vcsScanPathLength::>
<vcsScanPathLength::CIS resource to scan:  type=DATA_SOURCE  path=/services/databases/TEST00>
<vcsScanPathLength::      No paths found exceeding maximum.>
<vcsScanPathLength::>
<vcsScanPathLength::Final Scan Report:>
<vcsScanPathLength::      CIS Paths found >= 259 chars.  Total=2>
<vcsScanPathLength::>
<vcsScanPathLength::      Note: Subversion has no limitations with long path names.>
<vcsScanPathLength::      Note: TFS implementation on windows is affected by long path names.>
<CisDeployTool::Successfully completed vcsScanPathLength.>
```

```
@param serverId - target server name

@param vcsMaxPathLength - a positive integer length from which to compare
path lengths found in vcsResourcePathList.  When 0, use the default
CommonConstants.maxWindowsPathLen=259.

@param vcsResourcePathList- a comma separated list of CIS resource paths
to scan.  E.g.
"/shared/mypath,/services/databases/mydb,/services/webservices/mypath"

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

vcsScanPathLength(String serverId, int vcsMaxPathLength String
vcsResourcePathList, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

13. generateVCSXML

Generate a template VCSModule.xml file starting at a given path in CIS.

```
@param serverId target server id from servers config xml

@param startPath starting path of the resource e.g /shared

@param pathToDataSourceXML path including name to the VCS source xml
which needs to be created

@param pathToServersXML path to the server values xml

@throws CompositeException

public void generateVCSXML(String serverId, String startPath, String
pathToVCSXML, String pathToServersXML) throws CompositeException;
```

Method Definitions and Signatures (New VCS Connections via VCSModule.XML)

1. **vcsInitWorkspace2**

Initialize the VCS local workspace on the deployment server by linking a local folder with a VCS repository project and checking out all the resources from the VCS repository into the local workspace folder. This method uses VCSModule.xml for VCS connection properties.

```
@param vcsConnectionId - VCS Connection property information

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@throws CompositeException

public void vcsInitWorkspace2(String vcsConnectionId, String
pathToVcsXML, String vcsUser, String vcsPassword) throws
CompositeException;
```

2. **vcsInitializeBaseFolderCheckin2**

Initialize base folder check-in from the local workspace to the VCS repository. This provides an alternative way to establish the DV repository base folders into the VCS repository without checking in the entire DV repository. This can be useful for multi-tenant environments where only certain folders will be held under version control. The issue is that all the base-level folders must first be checked in into the VCS prior to any user-level folders being checked in. This method uses the deployment configuration property file "deploy.properties" for VCS connection properties.

This method will perform a workspace initialization first followed by the base folder checkin.

```
@param vcsConnectionId - VCS Connection property information

@param customPathList - a comma separated list of paths that are added to
the base paths of /shared or /services/databases or
/services/webservices.  These paths and their corresponding .cmf file
will be created during initialization of the workspace and VCS
repository. This strategy provides a way to accommodate a series of
folders as part of the base-level check-in to insure that all
intermediate folders are checked in leading up the actual folders that
the user is interested in.   If these base-level, intermediate folders
are not checked in, the check-out process will not work properly.  There
must be a .cmf file for each DV resource including folders.  The
following line demonstrates the mandatory requirement for how a catalog
or schema is identified.  The reason for this is that the position
following the database name may be either a catalog or schema.  The
[TYPE=DATABASE] is optional since the position of the database always
follows /services/databases.

/services/databases/MyDb[TYPE=DATABASE]/MyCat[TYPE=CATALOG]/MySch[TYPE=SC
HEMA]"
```

```
@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

public void vcsInitializeBaseFolderCheckin(String vcsConnectionId, String
customPathList, String pathToVcsXML, String vcsUser, String vcsPassword)
throws CompositeException
```

3. **vcsCheckout2**

   Check out changes from the VCS repository and import the differences into the CIS server. If folders are present in CIS that are not present in the repository, those folders will be deleted in CIS. This method does not reference VCSModule.xml resources but contains all parameters in the call. This method uses VCSModule.xml for VCS connection properties.

```
@param serverId - target server name

@param vcsConnectionId - VCS Connection property information

@param vcsResourcePath - the actual CIS resource path (not encoded)

@param vcsResourceType - the resource type

@param vcsRevision - the revision from the VCS (HEAD or an integer value
representing the revision number.)

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@throws CompositeException

void vcsCheckout2(String serverId, String vcsConnectionId, String
vcsResourcePath, String vcsResourceType, String vcsRevision, String
pathToVcsXML, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

4. **vcsCheckout2 (overloaded version for VCS labels)**

   Check out changes from the VCS repository and import the differences into the CIS server using VCS labels. If folders are present in CIS that are not present in the repository, those folders will be deleted in CIS. This method does not reference VCSModule.xml resources but contains all parameters in the call. This method uses VCSModule.xml for VCS connection properties.

```
@param serverId - target server name

@param vcsConnectionId - VCS Connection property information
```

```
@param vcsResourcePath - the actual CIS resource path (not encoded)

@param vcsResourceType - the resource type

@param vcsLabel - the VCS label that associates an entire release of CIS
resources together.  If not null, then vcsResourcePath and
vcsResourceType must be null otherwise vcsResourcePath takes precedence.

@param vcsRevision - the revision from the VCS (HEAD or an integer value
representing the revision number.)

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@throws CompositeException

void vcsCheckout2(String serverId, String vcsConnectionId, String
vcsResourcePath, String vcsResourceType, String vcsLabel, String
vcsRevision, String pathToVcsXML, String pathToServersXML, String
vcsUser, String vcsPassword) throws CompositeException;
```

5. **vcsCheckouts2**

Check out changes from the repository and import the differences into the CIS server. If folders are present in CIS that are not present in the repository, those folders will be deleted in CIS. This method uses VCSModule.xml for VCS connection properties.

```
@param serverId - target server name

@param vcsConnectionId - VCS Connection property information

@param vcsIds - comma separated list of VCS identifiers.

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@throws CompositeException

void vcsCheckouts2(String serverId, String vcsConnectionId, String
vcsIds, String pathToVcsXML, String pathToServersXML, String vcsUser,
String vcsPassword) throws CompositeException;
```

6. **vcsCheckin2**

Check in the changes from the local workspace to the VCS repository.   This method does not reference VCSModule.xml resources but contains all parameters in the call. This method uses VCSModule.xml for VCS connection properties.

Note: For GIT only, vcsCheckin2 is redirected to vcsForcedCheckin.

```
@param serverId - target server name
```

```
@param vcsConnectionId - VCS Connection property information
@param vcsResourcePath - the actual CIS resource path (not encoded)
@param vcsResourceType - the resource type
@param vcsMessage - the message that describes the checkin
@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against
@param pathToServersXML - path to the server values XML
@param vcsUser - the VCS user passed in from the command line
@param vcsPassword - the VCS user passed in from the command line
@throws CompositeException
void vcsCheckin2(String serverId, String vcsConnectionId, String
vcsResourcePath, String vcsResourceType, String vcsMessage, String
pathToVcsXML, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

7. **vcsCheckins2**

   References multiple vcsIds in the VCSModule.xml to check in the changes from the local workspace to the VCS repository. This method uses VCSModule.xml for VCS connection properties.

   Note: For GIT only, vcsCheckins2 is redirected to vcsForcedCheckin.

```
@param serverId - target server name
@param vcsConnectionId - VCS Connection property information
@param vcsIds - comma separated list of VCS identifiers.
@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against
@param pathToServersXML - path to the server values XML
@param vcsUser - the VCS user passed in from the command line
@param vcsPassword - the VCS user passed in from the command line
@throws CompositeException
void vcsCheckins2(String serverId, String vcsConnectionId, String vcsIds,
String pathToVcsXML, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

8. **vcsForcedCheckin2**

   Force check in is based on what is in the local workspace for CIS vs. the repository. Overwrite differences in the VCS repository favor of what is in the CIS server. This method does not reference VCSModule.xml for resources but contains all parameters in the call. This method uses VCSModule.xml for VCS connection properties.

```
@param serverId - target server name
@param vcsConnectionId - VCS Connection property information
```

```
@param vcsResourcePath - the actual CIS resource path (not encoded)

@param vcsResourceType - the resource type

@param vcsMessage - the message that describes the checkin

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@throws CompositeException

void vcsForcedCheckin2(String serverId, String vcsConnectionId, String
vcsResourcePath, String vcsResourceType, String vcsMessage, String
pathToVcsXML, String pathToServersXML, String vcsUser, String
vcsPassword) throws CompositeException;
```

9. vcsForcedCheckins2

References multiple vcsIds in the VCSModule.xml to force check in based on what is in the local workspace for CIS vs. the repository. Overwrite differences in the VCS repository favor of what is in the CIS server. This method uses VCSModule.xml for VCS connection properties.

```
@param serverId - target server name

@param vcsConnectionId - VCS Connection property information

@param vcsIds - comma separated list of VCS identifiers.

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@throws CompositeException

void vcsForcedCheckins2(String serverId, String vcsConnectionId, String
vcsIds, String pathToVcsXML, String pathToServersXML, String vcsUser,
String vcsPassword) throws CompositeException;
```

10. vcsPrepareCheckin2

Prepare check in by updating the local workspace copy and comparing with VCS but don't check in. This method does not reference VCSModule.xml but contains all parameters in the call. This method uses VCSModule.xml for VCS connection properties.

```
@param serverId - target server name

@param vcsConnectionId - VCS Connection property information

@param vcsResourcePath - comma separated list of VCS identifiers.

@param vcsResourceType - the resource type
```

```
@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@throws CompositeException

void vcsPrepareCheckin2(String serverId, String vcsConnectionId, String
vcsResourcePath, String vcsResourceType, String pathToVcsXML, String
pathToServersXML, String vcsUser, String vcsPassword) throws
CompositeException;
```

11. vcsPrepareCheckins2

References multiple vcsIds in the VCSModule.xml to prepare check in by updating the local workspace copy and comparing with VCS but don't check in. This method uses VCSModule.xml for VCS connection properties.

```
@param serverId - target server name

@param vcsConnectionId - VCS Connection property information

@param vcsIds - comma separated list of VCS identifiers.

@param pathToVcsXML - path including name to the VCS Module XML
containing a list of vcsIds to execute against

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@throws CompositeException

void vcsPrepareCheckins2(String serverId, String vcsConnectionId, String
vcsIds, String pathToVcsXML, String pathToServersXML, String vcsUser,
String vcsPassword) throws CompositeException;
```

14. vcsScanPathLength2

This method handles scanning the DV path and searching for encoded paths that equal or exceed the windows 259 character limit. If found this routine reports those paths. The 259 character limit is only a limitation for windows-based implementations of VCS like TFS. Subversion does not have this issue. This is useful if the developers want to see if there will be any path issues prior to checking in the resources. The length of the folder is calculated by summing the length of VCS_WORKSPACE + VCS_PROJECT_ROOT + CIS_ENCODED_FOLDER.

The parameter “vcsMaxPathLength” can be used to specify what the max length to report on. If a developer wants to run this with say 230 then it will report on any encoded paths are longer than 230 characters. This way they can determine if they have any paths that are getting close to the absolute limit of 259 characters.

Example:

VCS_WORKSPACE=P:/TFSsw	Len=8

VCS_PROJECT_ROOT=/DV_70/cis_objects Len=25
CIS_FOLDER=/shared/test00_longpath/SVN/PATH-this is a very long path with spaces in it and is meant/FILE-this is a very long file name with spaces in it
CIS_ENCODED_FOLDER=/shared/test00__longpath/SVN/PATH-this_0020is_0020a_0020very_0020long_0020path_0020with_0020spaces_0020in_0020it_0020and_0020is_0020meant/FILE-this_0020is_0020a_0020very_0020long_0020file_0020name_0020with_0020spaces_0020in_0020it
Len=230
Total Path Length=263 which exceeds 259 windows max.

Plan File Entry:

```
PASS	TRUE	ExecuteAction	vcsScanPathLength2 $SERVERID $VCONN 259 "/shared/test00_longpath,/services/databases/TEST00" "$MODULE_HOME/VCSModule.xml" "$MODULE_HOME/servers.xml"
```

Sample output:

```
<---------------------------------------------->
<CONFIG_PROPERTY_FILE=deploy.properties>
<---------------------------------------------->
<vcsScanPathLength::***** BEGIN COMMAND *****>
<vcsScanPathLength::>
<vcsScanPathLength::>
<vcsScanPathLength::---VCS Module ID Arguments:>
<vcsScanPathLength::	vcsResourcePathList=		/shared/test00_longpath,/services/databases/TEST00>
<vcsScanPathLength::	VCS_WORKSPACE_PROJECT=		P:/TFSww/DV_70/cis_objects>
<vcsScanPathLength::	VCS_WORKSPACE_PROJECT Length=	33>
<vcsScanPathLength::	Max Windows Path Len=		259>
<vcsScanPathLength::>
<Resource exists? [true] /shared/test00_longpath on server localhost9430http>
<vcsScanPathLength::>
<vcsScanPathLength::CIS resource to scan:  type=CONTAINER  path=/shared/test00_longpath>
<vcsScanPathLength::	TotalPathLen=263   EncodedPathLen=230   ResourceType=PROCEDURE
EncodedPath=/shared/test00__longpath/SVN/PATH-
this_0020is_0020a_0020very_0020long_0020path_0020with_0020spaces_0020in_0020it_0020and_0020is_0020meant/FIL
E-this_0020is_0020a_0020very_0020long_0020file_0020name_0020with_0020spaces_0020in_0020it
*****OriginalPath*****=/shared/test00_longpath/SVN/PATH-this is a very long path with spaces in it and is meant/FILE-this is
a very long file name with spaces in it>
<vcsScanPathLength::	TotalPathLen=299   EncodedPathLen=266   ResourceType=PROCEDURE
EncodedPath=/shared/test00__longpath/SVN/PATH-
this_0020is_0020a_0020very_0020long_0020path_0020with_0020spaces_0020in_0020it_0020and_0020is_0020meant_00
20to_0020exceed_0020the_0020limit/FILE-
this_0020is_0020a_0020very_0020long_0020file_0020name_0020with_0020spaces_0020in_0020it
*****OriginalPath*****=/shared/test00_longpath/SVN/PATH-this is a very long path with spaces in it and is meant to exceed
the limit/FILE-this is a very long file name with spaces in it>
<vcsScanPathLength::>
<Resource exists? [true] /services/databases/TEST00 on server localhost9430http>
<vcsScanPathLength::>
<vcsScanPathLength::CIS resource to scan:  type=DATA_SOURCE  path=/services/databases/TEST00>
<vcsScanPathLength::	No paths found exceeding maximum.>
<vcsScanPathLength::>
<vcsScanPathLength::Final Scan Report:>
<vcsScanPathLength::	CIS Paths found >= 259 chars.  Total=2>
<vcsScanPathLength::>
<vcsScanPathLength::	Note: Subversion has no limitations with long path names.>
<vcsScanPathLength::	Note: TFS implementation on windows is affected by long path names.>
<CisDeployTool::Successfully completed vcsScanPathLength2.>
```

```
@param serverId - target server name

@param vcsConnectionId - VCS Connection property information
```

```
@param vcsMaxPathLength - a positive integer length from which to compare
path lengths found in vcsResourcePathList.  When 0, use the default
CommonConstants.maxWindowsPathLen=259.

@param vcsResourcePathList - a comma separated list of CIS resource paths
to scan.  E.g.
"/shared/mypath,/services/databases/mydb,/services/webservices/mypath"

@param pathToVcsXML – the full path to the VCS Module XML which contains
the VCS connection information.

@param pathToServersXML - path to the server values XML

@param vcsUser - the VCS user passed in from the command line

@param vcsPassword - the VCS user passed in from the command line

@return void

@throws CompositeException

vcsScanPathLength(String serverId, String vcsConnectionId, int
vcsMaxPathLength, String vcsResourcePathList, String pathToVcsXML, String
pathToServersXML, String vcsUser, String vcsPassword) throws
CompositeException;
```

12. generateVCSXML2

Generate a template VCSModule.xml file starting at a given path in CIS.

```
@param serverId target server id from servers config xml

@param vcsConnectionId - VCS Connection property information

@param startPath starting path of the resource e.g /shared

@param pathToDataSourceXML path including name to the VCS source xml
which needs to be created

@param pathToServersXML path to the server values xml

@throws CompositeException

void generateVCSXML2(String serverId, String vcsConnectionId, String
startPath, String pathToVcsXML, String pathToServersXML) throws
CompositeException;
```

7 Exceptions and Messages

The following are common exceptions and messages that may occur.

1. Wrong Number of Arguments:

This may occur when you do not place double quotes around comma separated lists.

2. Repository moved temporarily to…:

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

3. Commit failed…not under version control:

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

4. Commit failed…file 'x' remains in conflict:

2011-07-29 08:02:21,330 main INFO [...] - <PDTool::Abnormal Script Termination. Script will exit. ERROR=com.tibco.ps.common.exception.CompositeException: /usr/bin/svn Execution Returned an Error=svn: Commit failed (details follow):
svn: Aborting commit:
'/u01/opt/DeployTool/PDTool/vcs_svn/cisVcsWorkspace/cis_objects/shared/test00/ResourceCache/testCacheProc_procedure.cmf' remains in conflict
Resolution: run vcs workspace initialization (ExecutePDTool -initvcs user password

5. CreateProcess error=193, %1 is not a valid:

2012-07-03 06:59:29,813 main ERROR [com.tibco.ps.common.util.CompositeLogger] - <Cannot run program "C:/Program Files/Perforce/p4" (in directory "."): CreateProcess error=193, %1 is not a valid
Win32 application
Cannot run program "C:/Program Files/Perforce/p4" (in directory "."): CreateProcess error=193, %1 is not a valid Win32 application>

Resolution: There are spaces in the command path for the VCS. This is a known problem and must be worked around by putting the VCS Path in the windows environment. Take VCS_HOME path and set the windows "Path" environment. Set VCS_EXEC_FULL_PATH=false.
VCS_HOME=C:/Program Files/Perforce
VCS_EXEC_FULL_PATH=false

8 Issues and Resolutions

The following are a list of issues and resolutions.

1. **Issue 1**: Cannot execute PDToolStudio and PDTool on the same machine.
    a. **VCS**: Perforce
    b. **Type**: Bug
    c. **Status**: Resolved
    d. **Release**: July 2012
    e. P4CLIENT must be different for PDTool and PDToolStudio and then it works. P4CLIENT is defined by VCS_WORKSPACE_NAME.
    f. Modified property files to force a difference by default using VCS_WORKSPACE_NAME property.
    g. Modified XML Schema to add VCS_WORKSPACE_NAME for PDTool
2. **Issue 2**: Filename is too long.
    a. **VCS**: All VCS
    b. **Type**: Tech Improvement
    c. **Status**: Resolved (within limits)
    d. **Release**: July 2012
    e. Modified batch files for PDTool and PDToolStudio to use a Substitution variable
    f. Reproduced issue with: Error=open for read: c:\Program Files\Composite Software\CIS 6.1.0\conf\studio\PDToolStudio\p4_sworkspace\cis_objects\shared\Utilities\repository\modelXSLT\modelGetResourceResponse\getResourceResponse__DATABASE__TABLE__w__Indexes\getBasicResourceXML__DATABASE__TABLE_002Exml_tree.cmf: The file name is too long.
    g. Set SUBST v: "c:\Program Files\Composite Software\CIS 6.1.0\conf\studio\PDToolStudio"
    h. Resolved path too long for all files and successfully checked in all Utilities
    i. At this time using NTFS style notation does not work for perforce and therefore this option was not pursued: "\\?\c:\Program Files\Composite Software\CIS 6.1.0\conf\studio\PDToolStudio"
3. **Issue 3**: On UNIX, vcsinit hangs as if it is trying to pop up the acknowledgement file.
    a. **VCS**: Perforce
    b. **Type**: Bug
    c. **Status**: Resolved
    d. **Release**: July 2012
    e. On windows, it will still pop up a window and the user will acknowledge the vcs initialization. On UNIX it cannot pop up a window in the process space

so standard out must be redirected to standard in so that the perforce request is automatically acknowledge. This is accomplished via "p4 client -o | p4 client –i". Therefore, the process completes without hanging.

4. **Issue 4:** Checkout using labels.
    a. **VCS**: Perforce
    b. **Type**: Enhancement
    c. **Status**: Resolved
    d. **Release**: July 2012
    e. Implemented p4 sync –f @label
    f. Remove workspace project directory first
    g. Execute p4 sync –f @label which forces all files in the label to be brought down to the local workspace.
    h. Caveat: User must checkin from studio at root "localhost:9400 (/)"
        i. Must have root.cmf, shared.cmf, services.cmf, database.cmf and webservices.cmf as the baseline folders. If those are not there then diffmerger marks folders for deletion when the car file is imported.
    i. Caveat: When checking in at root the following resources are checked in:
        i. / – root.cmf (readonly)
        ii. /policy – policy.cmf (readonly)
        iii. /security – security.cmf (readonly)
        iv. /services – services.cmf (read/write)
            1. /databases – databases.cmf (read/write)
            2. /webservices – webservices.cmf (read/write)
        v. /shared – shared.cmf (read/write)
        vi. /system – system.cmf (readonly)
        vii. /users – user.cmf (readonly)

    j. Info: p4 sync …@label will not work because it leaves empty folders and diffmerger gets confused. This approach was abandoned.
5. **Issue 5**: VCS Execution Command path with spaces fails
    a. **VCS**: All VCS
    b. **Type**: Bug
    c. **Status**: Workaround
    d. **Release**: July 2012
    e. Executing a VCS command for initialization where the path has spaces like "C:\Program Files\Perforce" results in an error.
    f. Workaround: Put the path in the Windows environment and set VCS_EXEC_FULL_PATH=false

9 Conclusion

Concluding Remarks

The Promotion and Deployment Tool is a set of pre-built modules intended to provide a turn-key experience for promoting DV resources from one DV instance to another. The user only requires system administration skills to operate and support. The code is transparent to operations engineers resulting in better supportability. It is easy for users to swap in different implementations of a module using the Spring framework and configuration files.

How you can help!

Build a module and donate the code back to Professional Services for the advancement of the "***Promotion and Deployment Tool***".