WHITE PAPER

COMPOSITE SOFTWARE

Composite Data Virtualization

Composite PS Promotion and Deployment Tool

User's Guide

Composite Professional Services

March 2014

TABLE OF CONTENTS

DOCUMENT CONTROL

Version History

Version	Date	Author	Description
1.0	03/31/2011	Mike Tinius	Initial revision
1.0.1	08/01/2011	Mike Tinius	Revision based on Architecture changes
1.1	3/9/2012	Mike Tinius	Added text regarding module ID wild card usage
3.0	8/21/2013	Mike Tinius	Updated docs to Cisco format.
3.1	1/31/2014	Mike Tinius	Added updateResourceCacheEnabled method to ResourceCache Module.
3.2	2/14/2014	Mike Tinius	Prepare docs for open source.
3.3	3/24/2014	Mike Tinius	Changed references of XML namespace to www.dvbu.cisco.com

Related Documents

Document	File Name	Author
Composite PS Promotion and Deployment Tool Installation Guide v1.0	Composite PS Promotion and Deployment Tool Installation Guide v1.0.pdf	Mike Tinius
LabPD-DeployTool-v1.0	LabPD-DeployTool-v1.0.pdf	Mike Tinius
PS Promotion and Deployment Tool v1.1	PS Promotion and Deployment Tool - v1.1.ppt	Mike Tinius
Composite PS Promotion and Deployment Tool Module - Archive	Composite PS Promotion and Deployment Tool Module - Archive.pdf	Mike Tinius
Composite PS Promotion and Deployment Tool Module - DataSource	Composite PS Promotion and Deployment Tool Module - DataSource.pdf	Mike Tinius
Composite PS Promotion and Deployment Tool Module - Group	Composite PS Promotion and Deployment Tool Module - Group.pdf	Mike Tinius
Composite PS Promotion and Deployment Tool Module - Privilege	Composite PS Promotion and Deployment Tool Module - Privilege.pdf	Mike Tinius
Composite PS Promotion and Deployment Tool Module - Rebind	Composite PS Promotion and Deployment Tool Module - Rebind.pdf	Jerry Joplin
Composite PS Promotion and Deployment Tool Module - Regression	Composite PS Promotion and Deployment Tool Module - Regression.pdf	*Sergei Sternin*
Composite PS Promotion and Deployment Tool Module - Resource Cache	Composite PS Promotion and Deployment Tool Module - Resource Cache.pdf	Mike Tinius
Composite PS Promotion and Deployment Tool Module - Resource	Composite PS Promotion and Deployment Tool Module - Resource.pdf	Mike Tinius
Composite PS Promotion and Deployment Tool Module - Server Attribute	Composite PS Promotion and Deployment Tool Module - Server Attribute.pdf	Mike Tinius
Composite PS Promotion and Deployment Tool Module - Server Manager	Composite PS Promotion and Deployment Tool Module - Server Manager.pdf	Gordon Rose
Composite PS Promotion and Deployment Tool Module - Trigger	Composite PS Promotion and Deployment Tool Module - Trigger.pdf	Kevin O'Brien
Composite PS Promotion and Deployment Tool Module - User	Composite PS Promotion and Deployment Tool Module - User.pdf	Mike Tinius
Composite PS Promotion and Deployment Tool Module - Version Control System.pdf	Composite PS Promotion and Deployment Tool Module - Version Control System.pdf	Mike Tinius

Composite PS Promotion and Deployment Developer’s Guide – Field Edition	Composite PS Promotion and Deployment Tool Developer's Guide - Field Edition.docx	Gordon Rose

Composite Products Referenced

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

INTRODUCTION

Purpose

The purpose of this document is to provide guidance on how to promote and deploy Composite Information Server (CIS) resources from one environment to another environment using automated scripts. An environment is defined as a CIS instance or CIS cluster which is expressly used for development (DEV), testing (TEST), unit acceptance testing (UAT), or production (PROD). Different customers use different names but the ones provided will give you a baseline understanding which you can extrapolate to your environment. Additionally, this User Guide will cover other environments such as the version control system (VCS) and deployment server (DS). Extensive work has been done by Composite Professional Services to build an out-of-the-box experience for performing CIS resource promotion and deployment. The culmination of the work is the "***PS Promotion and Deployment Tool***" ("PD Tool"). The framework's seeks to provide 90% of customers with an experience that merely involves setting configuration files for deployment. In those 10% situations, where the customer needs more than what is provided out-of-the-box, the framework provides extensibility to build new modules or invoke custom CIS procedures to interface with Composite's repository. The key to performing automated promotion and deployment is automatically affecting changes to CIS resources by using the Composite Web Service Repository API. The modules describe herein provide interfaces to various methods in the Composite Web Service Repository API.

This document provides:

1. **Problem Definition** – Describes why scripts are needed for deployment.
2. **Design Philosophy** – Describes the architecture and design principles.
3. **Distribution** – Describes the distribution files.
4. **Installation** – Describes the installation procedure.
5. **Execution** – Describes how to execute the scripts.
6. **Functional Modules** – Describes the functional modules implemented
7. **Conclusion** – Concluding thoughts.

Audience

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

- **Operations personnel** – provides guidance on how to execute promotion scripts.
- **Architects** – provides guidance on how CIS fits in with the version control and the deployment infrastructure.
- **Developers** – provides guidance on how to develop new Modules of functionality.

PROBLEM DEFINITION

What is promotion?

Every customer must promote CIS resources from one CIS environment to another. Without a methodology or scripts, it is a manual process. Version Control Systems (VCS) add another complexity to the problem definition as some customers want to be able to deploy CIS assets directly from a VCS such as subversion.

What is promotion?

Promotion is the task of moving a CIS resource such as a view or procedure and configuring that asset according to the environment that it is being moved to. Promotion encompasses the entire process and takes a holistic view of an environment.

1. **Requirements** – Some customers have rigorous and demanding deployment requirements and some have none.
2. **Variety** – There are a variety of environments supported by Composite including Windows and various flavors of UNIX.
3. **Paradigm** – CIS resources may be under source control and some may not. This affects the deployment paradigm.

Deployment

Deployment is the task of importing the CIS resources into the target Composite instance or cluster.

Configuration

Configuration is the task of modifying a CIS resource in the target Composite instance or cluster. One example of configuration is that data sources in development have a different hostname and password than data sources in test, UAT and production. It is necessary to tweak certain configuration parameters based on the environment that the CIS resources are being promoted into.

Version Control

Version Control Systems (VCS) provide a way to save different versions of the CIS resources. Many customers want to be able to deploy those code assets directly from the VCS to a target CIS server.

DESIGN PHILOSOPHY

Modularity

Modularity by itself is not enough. By creating a solid framework around the module allows the creator of that module to leverage common functions. Let's take the example of a carpenter who carries a tool belt filled with specialized tools. In this analogy, the tool belt is the framework that provides a common function of holding the tools and allowing the carpenter to access his tools with ease of use. The tools themselves represent the specialized modules. The carpenter uses the right tool for the job at hand. The carpenter can easily swap in different hammers for different jobs. This tool-kit approach allows the user of the Promotion and Deployment Tool to swap in different implementations of Modules with ease using the Apache Spring framework. This is nothing more than a configuration file from the User's perspective. All the heavy-lifting was handled by the implementation of the Promotion and Deployment Tool.

What makes up a Module

A Module is a functional grouping of actions. An action can be anything that affects a change to a CIS resource or the CIS environment. For example, the ArchiveModule contains actions for import and export. The DataSourceModule contains actions for re-introspect and update. The User is only concerned with what actions are available and how to affect a change on CIS. The way in which a user affects change to CIS is by configuring the XML property file associate with a Module. Each Module has its own Module XML definition. To summarize, a module is made up of the following items:

1. **Module Name** – The name of the Module.
2. **Module Action** – The action to be performed against a CIS instance.
3. **Module XML Property File** – The XML property file located in PDTool\resources\modules

Promotion Scenarios

The Promotion and Deployment Tool supports command line and Ant execution in both Windows and UNIX environments. It will also support local and remote deployments. It will support Composite CAR file and Version Control System (VCS) based deployments.

The following diagram depicts the three scenarios and will be described in more detail the ensuing sections. The diagram shows several individual developer workstations feeding changes into a central Composite Development Server. The process of getting changes into the Central Development Server is outside the scope of the Promotion and Deployment Tool. The scenarios pick at the point where a deployment is too occur starting with artifacts found in the Central Development Server. The CIS Target Promotion Server is where the artifacts will be moved to. A Target server is representative of CIS instance such as Test, Integration, UAT, and Production. Customers have different names for these CIS instances. The point is with promotion is that there is a source CIS instance and a target CIS instance. When

performing promotion with version control, there will also be a VCS server which is used to check-in and check-out artifacts from Composite. Finally, the entire promotion process may need to be executed from a remote server instead of being run on the Target server.

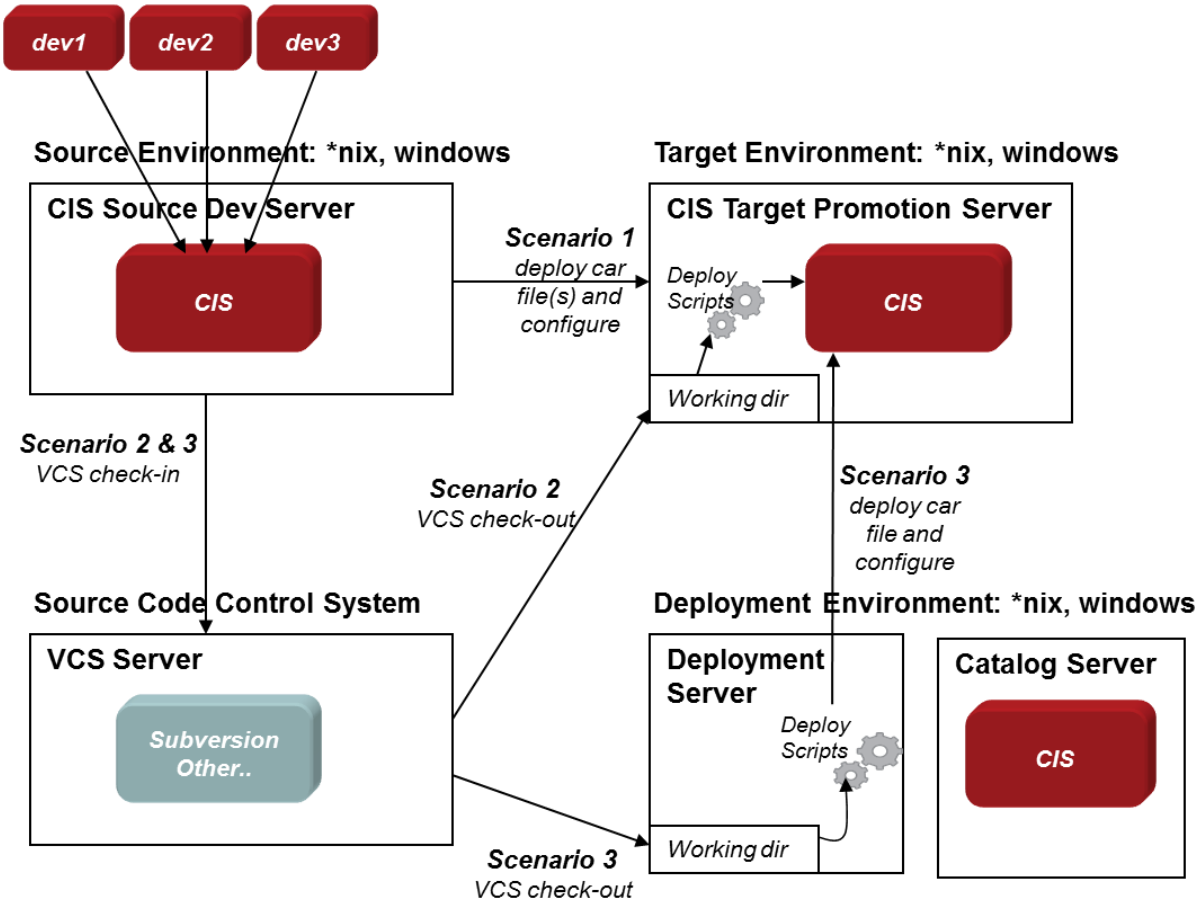


Figure 1 – Deployment Scenarios

Scenario 1 – Local CAR file based Deployment

In this scenario the scripts are executed locally on the Target Promotion Server. The PD Tool imports a CAR file into the target CIS instance and then executes various configuration actions on the target server.

Scenario 2 – Local VCS based Deployment

In this scenario, the Target Promotion Server is executing the promotion process. Instead of CAR files, the PD Tool is interfacing with a VCS server to check-out the specified artifacts, build a car file on the fly and then import into the Target Promotion Server. Additionally, the PD Tool will execute various configuration actions on the target CIS instance.

Scenario 3 – Remote VCS or CAR based Deployment

In this scenario, the Target Promotion Server is not involved in executing the PD Tool. Instead, there is a dedicated server that will execute the PD Tool. The remote promotion server would interface with VCS to check-out the specified CIS resources, build a car file on the fly and remotely import into the Target CIS Promotion instance. Similarly, if VCS was not involved, it could export specified CIS artifacts from the source CIS instance and import into the target instance. Finally, the Remote Promotion Server would remotely connect to the target CIS instance and execute various configuration actions. A CIS instance is not required on the deployment server for the PD Tool to function.

Technical Architecture

The design shown in Figure 2 below exemplifies the modular architecture used by the Promotion and Deployment Tool. In the description of each section below, it will be indicated what involvement by the user is required or possible. For any information regarding extending the framework, the user should consult the Developer's Guide. It is intended that 90% of the functionality of the Promotion and Deployment Tool is provided out of the box.

Shell/Batch Scripts

1. Driver Script – a single script used to invoke a java command "ExecutePDTool" that will orchestrate through a well-defined property file and execute tasks or actions.

    - Built by Composite PS. Generic. No need to extend.

Common Framework

Module Interfaces (Deployment Manager, Module Manager, WS API)

1. Deployment Manager, Module Manager, CIS WS Interface API - contains a single point of interface for both ANT and Command line utilities. It will provide logging, error handling, and XML property file parsing. It will provide a command line interface with the same parameters as ANT uses. It will provide looping through the XML property file, making iterative calls to the CIS WS API Interfaces which are largely based on single entry invocations and not lists. It will handle errors in a common way and log them.

    - Built by Composite PS. May be extended by PS or customer.

2. CIS JDBC Interface Jar – contains a single generic JDBC interface that can invoke any published CIS procedure

    - Built by Composite PS. No need to extend.

3. Common Jar – contains common framework functions that are required by the Interface jars and ant jars. Classpaths and packaging is separate from ANT so that an ant-less deploy can be realized. Contains common code for parsing XML and logging.

- Built by Composite PS. Generic. No need to extend.

Ant

1. Ant Jar – contains the necessary modules to execute ANT. Will be able to invoke Java-based interface wrapper modules as well as shell scripts. Its main job is for orchestrating task execution.

    - Built by Composite PS. Generic. No need to extend.

Configuration Files 

1. PDTool.dp – provides the sequence of task (actions) for the command line script to orchestrate through. This is known as the "Deployment Plan (dp)"

    - Configured by customer.

2. build.xml – provides the sequence of task (actions) for Ant to orchestrate through.

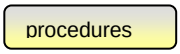
    - Configured by customer.

3. servers.xml – provides CIS server specific information

    - Configured by customer.

4. modules.xml – task module specific information.

    - Configured by customer.

CIS Published Procedures 

1. CIS Published Procedures – published to any virtual database for access via the JDBC interface. Focused on doing configuration.

    - Built by Composite PS or the customer.

2. CIS Procedures – These SQL Script Procedures are focused on performing configuration and provide an easy to use interface for a customer to execute logic within CIS. These procedures may invoke CIS WS Repository API or they may invoke pre-existing Utility components which in turn invoke the repository API.

    - Built by Composite PS or the customer.

3. Utilities – Composite Professional Services Asset procedures used to invoke the repository API's.

    - Built by Composite PS.

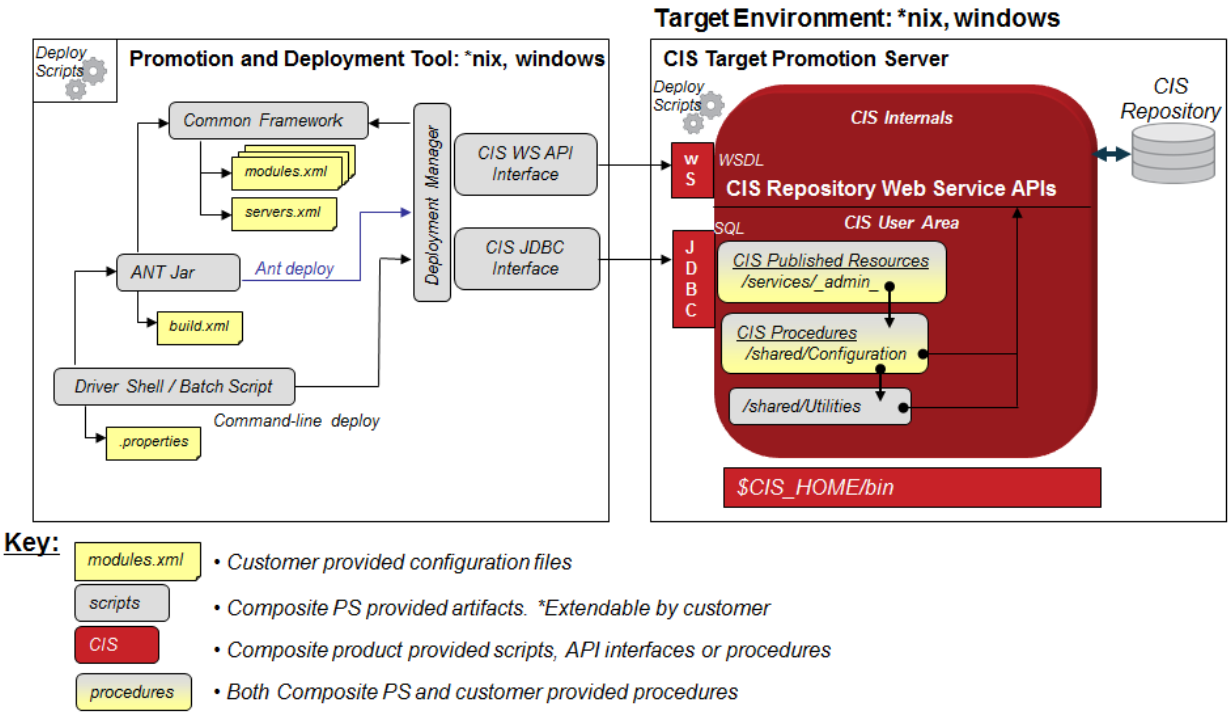


Figure 2 – Technical Architecture "A look under the covers"

Supportability

Composite provides a web service API to access its repository where all CIS artifacts are stored. The API allows several functions to be performed. This API has been externalized into several Java Modules as depicted in Figure 2. Additionally, a customer may choose to write their own SQL Procedures to interface with the repository API form within Composite. This uses the internal API interface. Ultimately, both invoke the same repository methods. CIS also provides general command line scripts for doing things like import, export, backup, restore, starting and stopping CIS. Lastly, the Promotion and Deployment Tool provide several log files for the user to be able to get information. The logging framework is consistent across Java, Ant and Command Line modules.

Phased Approach

Functionality will be delivered in phases.

Phase 1

Single CIS instance, local and remote deployments and core set of modules

Phase 2

Clustered CIS instances, additional modules.

EXECUTION

Command Line Script Execution

The objective of command line execution is to provide an easy to user interface for the user to orchestrate a Promotion and Deployment process. All of the scripts required for promotion are provided out of the box.

Preparing the Orchestration Deployment Plan File

The main orchestration plan file provides the user interface to script-based execution of Promotion and Deployment. The plan file provides a general definition for execution. Each line in the script represents an action to be taken during resource promotion. Actions are implemented by Modules. Modules provide a way of grouping related actions. The property file also dictates the order of execution. There is no distinction between Windows and UNIX. The variable replacement methodology can use both Windows style variables "%MODULE_HOME%" or UNIX style variables "$MODULE_HOME". The only recommendation is to be consistent with the use of a style. All examples will use UNIX style notation

1. **Property File Example** – Below is an example of a property file:

```
# Import
PASS TRUE ExecuteAction pkg_import qaserver import1
"$MODULE_HOME/ArchiveModule.xml" "$MODULE_HOME/servers.xml"

# Update Datasource
PASS TRUE ExecuteAction updateDataSources qaserver " datasource1, datasource2"
"$MODULE_HOME/DataSourceModule.xml" "$MODULE_HOME/servers.xml"

# Execute VCS Checkout
PASS TRUE ExecuteAction vcsCheckouts qaserver "test1" "$MODULE_HOME/VCSModule.xml"
"$MODULE_HOME/servers.xml"

# Execute Procedure
FAIL FALSE ExecuteAction executeProcedure qaserver testprocsimple TEST
"$MODULE_HOME/servers.xml" "'testprocsimple,'1'"
```

The following provides a detailed explanation of the specification for a property file and the rules to construct a property file.

2. **Plan File Specification** – How to construct a plan file. The name of the plan file can be anything the user wants. The default example file is PDTool.dp. A single line of execution represents an "action" and requires the following parameters unless optional is specified. Example of a line:

```
PASS TRUE ExecuteAction updateDataSources localhost "datasource1,datasource2"
"%MODULE_HOME%/DataSourceModule.xml" "%MODULE_HOME%/servers.xml"
```

2.1. **Editing the plan files** – When using notepad++ to edit plan files there is a certain configuration that can be set up so that notepad++ thinks it is a property file.

2.1.1. Select the menu option settings → Style Configurator → Select the Language "Property file" → Add "dp" to the User ext: → Save & Close

2.2. **Param1 (Expected Regression Behavior)** – Informs the script whether the user expects the action being executed is expected to pass or fail. This can be used as a regression test to test the outcome of an action. A summary log file is created in the logs directory which itemizes the execution and provides an overall success or failure based on the regression test. If all regression tests pass then the overall status is a pass. If one or more regression tests fail, then the overall status is a fail.

Value – PASS or FAIL (required parameter)

2.3. **Param2 (Exit on Error) –** Informs the script whether to exit the orchestration script or not. If set to TRUE and an error occur then the script immediately exits. If set to FALSE and an error occurs, then the orchestration script will continue processing. It is up to the user to decide the behavior. When doing a regression test, it is recommended to set all execution, action lines to FALSE so that the script will run to completion. Check the summary log file tor the outcome.

Value – TRUE or FALSE (required parameter)

2.4. **Param3 (Module Action Type) –** The type of action to execute. Currently the only action supported is "ExecuteAction" which correlates to a Java-implemented action.

2.4.1. **ExecuteAction –** Executes a Java Action implemented by the Promotion and Deployment Tool framework. The Java Action implements the CIS Repository Web Service API that which allows an action to affect a target server configuration.

2.4.2. In general, parameters follow the pattern below. However, there will be some Java actions that stray from this format. The safest approach is to consult the module documentation found in the /docs directory. Module documentation follows the format "Composite PS Promotion and Deployment Tool Module – <ModuleName>.pdf".

2.5. **Param4 thru N –** Specific space separated parameters for the script being executed.

2.5.1. General format for ExecuteAction:

action – the name of the action which equates to the Java method name.

serverId – the identifier of the CIS server in which to connect to in order to perform the specified action. The serverId is found in the servers.xml file.

moduleIds – one or more comma separated module identifiers which is used to identify the attributes for a particular module. The following rules apply to module Ids:

1. All wild card - " * " or whatever is configured to indicate all resources (PD Tool processes all resources in this case). Must have double quotes and a space surrounding the *.
2. Exclude list – coma-separated string with '-' or whatever is configured to indicate exclude resources as prefix like "-resource1, resource2" (PD Tool ignores passed in resources and processes the rest of the input xml
3. Exact list – comma-separated string like "resource1, resource2" (PD Tool processes only the resource names which are passed in)
4. Partial Wild card - prefix/postfix any label with a "*" or whatever is configured to indicate all resources.

*label - process all labels that end with "label"
label* - process all labels that start with "label"
label - process all labels that contain the string "label"
-*label - process all labels that DO NOT end with "label"
-label* - process all labels that DO NOT start with "label"
-*label* - process all labels that DO NOT contain the string "label"

modulePath – the path the module XML property file. For example the DataSourceModule.xml contains a list of data sources and their attributes.

serverPath – the path the servers.xml property file. The path may specify an environment variable (Windows: %MODULE_HOME% or UNIX: $MODULE_HOME).

moduleParams – Additional parameters that are specific to an action are placed at the end. *For example, the Java action* executeProcedure requires that d*ynamic arguments follow these rules* – single quoted, comma separated parameter list and enclosed in double quotes. Numbers and dates are also enclosed in single quotes. e.g. "'param1','0','2011-03-20'"

3. **Property File Rules** – How it works.

3.1. **Separators** – All parameters are space or tab separated. Commas are not used as parameter separators.

3.2. **Whitespace** – Any number of spaces or tabs may occur before or after a parameter and are trimmed.

3.3. **Parameters** – Parameters should always be enclosed in double quotes according to these rules:

3.3.1. when the parameter value contains a comma separated list:

EXAMPLE: "ds1,ds2,ds3"

3.3.2. when the parameter value contains spaces or contains a dynamic variable that will resolve to spaces

There is no distinguishing between Windows and Unix variables. Both UNIX style variables ($VAR) and Windows style variables (%VAR%) are valid and will be parsed accordingly.

All parameters that need to be grouped together that contain spaces are enclosed in double quotes.

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 of 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".

3.3.3. when the parameter value is complex and the inner value contains spaces

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"

3.3.4. when the parameter value that is being passed contains multiple single quoted parts

In this example a parameter list is being passed to the executeProcedure action.

EXAMPLE: "'myname','1','12.3','3.141592653589793','2000-02-01','23:59:01','1923-03-06 23:59:31','<node>text</node>','1'"

It is also worth noting that in this example, one of the parameters is an XML parameter. Since it is not possible to pass XML text such as < and > through scripts, the XML escape sequences are used.

Type	Ascii Character	Output Character in XML
quote	(")	&quot;
apostrophe	(')	&apos;
ampersand	(&)	&amp;
less than	(<)	&lt;
greater than	(>)	&gt;

3.4. **Comments** – A comment is designated by a hash (#) symbol preceding any text. Comments may occur on any line and will not be processed.

3.5. **Blank lines** – A blank line will not be processed. Blank lines are counted as lines for display purposes. If the last line is a blank line, it is not counted for display purposes.

Executing the Script

This section describes how to execute the script. The distribution zip file contains all of the necessary libraries, resources and scripts to execute batch or shell scripts. The root directory is called PDTool and the scripts are found in the bin directory. The user would change directories to <root-path>/PDTool/bin and then execute the following scripts depending on their environment. Therefore, the basic command syntax is as follows:

1. **Command Syntax**

   ExecutePDTool.bat | .sh -execType [path-to-property-file] [-vcsuser VCS_USERNAME] [-vcspassword VCS_PASSWORD] [-config deploy.properties]

2. **Windows execution**

   2.1. **Execute From:** Always execute from within a command-line window in PDTool/bin

   2.2. **Execute command-line property file with optional VCS user/password**

   ***Note**: If [**-vcsuser** vcsuser] and [**-vcspassword** vcspassword] are left off the command line, PD Tool expects to find them in the deploy.properties file. However, this is only when executing commands for the VCS Module. If no commands will be executed, then vcsuser and vcspassword do not have to be provided.*

   ***Note**: The **-config** option informs PDTool which /resources/config/deploy.properties should be used for a particular execution. The default is "deploy.properties".*

   ***Note**: The path to the property files for the -exec command may be a relative path which refers to a path outside of the PDTool directory. For example to reference a directory one level up from PDTool would look something like this: -exec ..\..\plans\PDTool.dp.*

   ExecutePDTool.bat -**exec** ../resources/plans/PDTool.dp [-vcsuser user] [-vcspassword password]

   2.3. **Execute VCS Workspace initialization with optional VCS user/password**

   ***Note**: If [vcsuser] and [vcspassword] are left off the command line, PD Tool expects to find them in the deploy.properties file.*

   ExecutePDTool.bat -**vcsinit** [**-vcsuser** user] [**-vcspassword** password]

   2.4. **Execute encrypt property file passwords in file**

   ExecutePDTool.bat - **encrypt** ../resources/plans/servers.xml

3. **UNIX execution**

   3.1. **Execute command-line property file with optional VCS user/password**

***Note**: If [vcsuser] and [vcspassword] are left off the command line, PD Tool expects to find them in the deploy.properties file. However, this is only when executing commands for the VCS Module. If no commands will be executed, then vcsuser and vcspassword do not have to be provided.*

***Note**: The path to the property files for the -exec command may be a relative path which refers to a path outside of the PDTool directory. For example to reference a directory one level up from PDTool would look something like this: -exec ../../plans/PDTool.dp.*

./ExecutePDTool.sh -**exec** ../resources/plans/PDTool.dp [-vcsuser user] [-vcspassword password]

3.2. **Execute VCS Workspace initialization with optional VCS user/password**

***Note**: If [vcsuser] and [vcspassword] are left off the command line, PD Tool expects to find them in the deploy.properties file.*

./ExecutePDTool.sh -**vcsinit** [**-vcsuser** user] [**-vcspassword** password]

3.3. **Execute encrypt property file passwords in file**

./ExecutePDTool.sh -**encrypt** ../resources/plans/servers.xml

Ant Execution

The objective of Ant execution is to provide an easy to user ant-based interface for the user to orchestrate a Promotion and Deployment process using a build.xml file. All of the scripts required for promotion are provided out of the box. For those familiar with build files, they will be very comfortable in this environment.

Preparing the Property Build File

The build.xml file is what drives the sequence of actions to be executed during the promotion process. The name of file can be anything the user wants. The same build file can be executed in either a Windows or UNIX environment.

1. **Build File Description** – The following items describe various parts of the build file. Project name and basedir should never change. These are defaults to for the build file.

   1.1. **Description** – The description (<description>) provides the user with a way to identify the overall objective of this build file.

   1.2. **Property name** – The property name (<property name="">) and value give the user a way to set variables and values much like environment variables do in scripts. The user will need to setup properties for each path to the Module XML file giving them meaningful names. The user may choose to setup variables for identifiers within the modules.

1.3. **File set** – The File set (<fileset dir="">) should not be changed. It provides a path to various jar files required for execution.

1.4. **Task definition name** – The task definition name (<taskdef name="">) identifies the two generic classes that are used for Java action and script execution.

1.5. **Target name** – The target name (target name="">) contains a sequence actions to execute. The actions refer to "executeJavaAction". All promotion capabilities are distilled down to one class for command execution.

1.6. **Execute Action** – The two choices are executeJavaAction or executeScriptAction.

1.6.1. **Argument separators** – The caret (^) symbol is used to separate arguments.

1.6.2. **executeJavaAction** – Takes an action and arguments. The number of arguments is dependent on the java method (action).

1.6.3. **Script behavior**

1.6.3.1. **endExecutionOnTaskFailure** – If the java action throws an error during execution then this property which is set to true or false will determine if the Ant execution should continue or stop.

2. **Build File Example** – Below is an example of an Ant build file:

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

        <description>description</description>

        <!-- Default properties -->
        <property name="SERVERID"                   value="localhost9400   "/>
        <property name="noarguments"                value="&amp;quot;&amp;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="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="dataSources" value="datasource1,datasource3"/>

        <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.cisco.dvbu.ps.deploytool.ant.CompositeAntTask"
classpathref="project.class.path"/>

        <!-- ================================
          target: default
```

```
        ================================= -->
       <target name="default" description="Update CIS with environment specific parameters">
           <executeJavaAction action="pkg_export"
arguments="${SERVERID}^export1^${pathToArchiveXML}^${pathToServersXML}"
endExecutionOnTaskFailure="TRUE"/>

        <executeJavaAction action="updateDataSources"
arguments="${SERVERID}^${dataSources}^${pathToDataSourcesXML}^${pathToServersXML}"
endExecutionOnTaskFailure="TRUE"/>

       </target>
</project>
```

Table 2: Ant build file example

Executing the Script

This section describes how to execute the Ant build file using a script. The distribution zip file contains all of the necessary libraries, resources and scripts to execute batch or shell scripts. The root directory is called PDTool and the scripts are found in the bin directory. The user would change directories to <root-path>/PDTool/bin and then execute the following scripts depending on their environment. Therefore, the basic command syntax is as follows:

1. **Command Syntax**

   ExecutePDTool.bat | .sh -execType [path-to-build-file] [[-vcsuser VCS_USERNAME] [-vcspassword VCS_PASSWORD] [-config deploy.properties]

2. **Windows execution**

   2.1. **Execute From:** Always execute from within a command-line window in PDTool/bin

   2.2. **Execute Ant build file with optional VCS user/password**

   ***Note**: If [vcsuser] and [vcspassword] are left off the command line, PD Tool expects to find them in the deploy.properties file. However, this is only when executing commands for the VCS Module. If no commands will be executed, then vcsuser and vcspassword do not have to be provided.*

   ***Note**: The -config option informs PDTool which /resources/config/deploy.properties should be used for a particular execution. The default is "deploy.properties".*

   ExecutePDTool.bat -**ant** ../resources/ant/build.xml [-vcsuser user] [-vcspassword password]

   2.3. **Execute VCS Workspace initialization with optional VCS user/password**

   ***Note**: If [vcsuser] and [vcspassword] are left off the command line, PD Tool expects to find them in the deploy.properties file.*

   ExecutePDTool.bat -**vcsinit** [-vcsuser user] [-vcspassword password]

   2.4. **Execute encrypt property file passwords in file**

ExecutePDTool.bat - **encrypt** ../resources/plans/servers.xml

3. **UNIX execution**

3.1. **Execute Ant build file with optional VCS user/password**

***Note**: If [vcsuser] and [vcspassword] are left off the command line, PD Tool expects to find them in the deploy.properties file. However, this is only when executing commands for the VCS Module. If no commands will be executed, then vcsuser and vcspassword do not have to be provided.*

./ExecutePDTool.sh -**ant** ../resources/ant/build.xml [-vcsuser user] [-vcspassword password]

3.2. **Execute VCS Workspace initialization with optional VCS user/password**

***Note**: If [vcsuser] and [vcspassword] are left off the command line, PD Tool expects to find them in the deploy.properties file.*

./ExecutePDTool.sh -**vcsinit** [-vcsuser user] [-vcspassword password]

3.3. **Execute encrypt property file passwords in file**

./ExecutePDTool.sh -**encrypt** ../resources/plans/servers.xml

PDTool Log Files

Both command line and Ant use the same log file configuration and output to the same log files. The /PDTool/resources/config/log4j.properties file defines where the log files are to be located.

Application Log File

1. **app.log** – application log file. The Java actions will output debug information to this log.

1.1. **location** – uses the location and file identified by log4j.appender.FileAppender.File=../logs/app.log

Summary Log File

2. **summary.log** – summary log file. Provides a quick summary for each executed line item action. Provides an overall status of execution.

2.1. **location** – uses the base directory identified by log4j.appender.FileAppender.File

```
================================================================================
Summary Status Log
================================================================================
Regression Status=Did the execution meet expectations.
Expected Status=Did the user expect this action to PASS or FAIL.
Actual Status=What really happened during execution of this action.
```

==========

Line #	Regression	Expected	Actual	Exit On Error	Action Type	Module Action
Line 38	PASS	PASS	PASS	TRUE	ExecuteAction	pkg_import
Line 41	PASS	PASS	PASS	TRUE	ExecuteAction	updateDataSources
Line 44	PASS	PASS	PASS	TRUE	ExecuteAction	pkg_export
Line 47	PASS	PASS	PASS	FALSE	ExecuteAction	executeConfiguredProcedures
Line 48	PASS	FAIL	FAIL	FALSE	ExecuteAction	executeConfiguredProcedures
Line 49	PASS	PASS	PASS	FALSE	ExecuteAction	executeConfiguredProcedures
Line 50	PASS	FAIL	FAIL	FALSE	ExecuteAction	executeConfiguredProcedures
Line 53	PASS	PASS	PASS	FALSE	ExecuteAction	executeProcedure
Line 57	PASS	PASS	PASS	FALSE	ExecuteAction	executeProcedure
Line 61	PASS	FAIL	FAIL	TRUE	ExecuteAction	updateDataSources

==========

Overall Regression Execution Status=PASS
Script was set to exit on error.
==========

Table 1: Summary Log

2.2. **Concept of Regression Testing** – By setting all of the Exit On Error flags to FALSE in the orchestration property file, the script will run to completion and execute all of the actions in the file. This will provide a complete summary of the actual regression for that action. The PDTool script will validate the Expected Status against the Actual status and determine the result of that line of execution. If the Expected status is Fail and the Actual status is Fail then the regression passed. In other words, when the Expected status = Actual status then Regression = PASS. When Expected status <> Actual status then Regression = FAIL. If any line contains a Regression status of FAIL, then the overall Regression status is FAIL, otherwise it is assigned a PASS status.

PDTOOL DEPLOYMENT STRATEGIES

Deployment Strategies

This section provides a description of various approaches or strategies for using PDTool. The approaches include the Basic Strategy and the Dynamic Strategy.

Why would a user choose one strategy over another?

- The **basic strategy** is just that…basic. For beginners it is the easiest to set up and not a lot of planning has to be put into it. If you want to get PDTool up and running quickly then you may want to start with this strategy especially if you are just trying to show the value of deployment for one environment.
- The **dynamic strategy** requires more planning up front to accommodate all of the environments that you want to deploy to. However, it can yield the greater benefits in terms of flexibility and reducing the maintenance on the number of configuration and plan files. With a little planning, a user can design a parameterized deployment strategy that can be used across all environments.

Basic Strategy

The basic strategy seeks to provide the baseline deployment strategy for PDTool.

Configuration Property File – uses the default deploy.properties files

Server List – the servers.xml file contains a listing of all the servers used in the environment.

Module XML files – uses a module.xml for each environment.

- DataSourceModule_TEST.xml
- DataSourceModule_PROD.xml

Deployment Plan File – uses a deployment plan for each environment which references a specific module.xml file for a given environment.

- Deploy_plan_TEST.dp
- Deploy_plan_PROD.dp

Dynamic Strategy

The dynamic strategy seeks to provide the maximum flexibility while reducing the number of configuration files to manage.

There are three main configuration files that a deployment specialist will need to create and manage. These files are described below:

- Deployment Configuration Property File
    - deploy.properties – provides a way to set standard and custom PDTool variables
- Deployment Plan – orchestrates the deployment
    - LabPD-Deploy.dp
- Module XML – provides a way to configure specific module properties
    - DataSourceModule.xml
    - VCSModule.xml
    - Etc.

The key to the “Dynamic Strategy” is variables. Variables help to parameterize the deployment and come up with the most flexible deployment strategy. Variables can be set in three different ways as described below:

- Operating System Environment Variables
    - Windows: SET MYVAR=myvalue
    - UNIX: export MYVAR=”myvalue”
- Java Environment Variables
    - -DPROJECT_HOME="%PROJECT_HOME%"
- PDTool Configuration Property File
    - SERVERID=localhost9440
    - DEPLOY_TYPE=DEV

Setting variables provides the deployment specialist a lot of flexibility. One option is to create a custom batch/shell script which gets invoked within the context of executing PDTool. Another option is to set the parameters the PDTool configuration property file. The user can execute PDTool using the -config option to specify the configuration file when executing PDTool. This can be especially useful when setting up a common variable with different values for the different deployment environments [DEV, TEST, UAT, PROD]. The parameterization concept will be explored further in the example provided below.

Example Objective:

- Utilize a deployment configuration property file for each environment where environment specific variables will be set.

- Maintain a single servers.xml file containing the list of Composite Servers.
- Maintain a single Module XML for each module vs. having multiple files per environment.
- Maintain a single plan file across all environments.

Configuration Property File – the configuration property file is established during installation. Typically, once this is setup it rarely changes. The PDTool installation guide discusses what properties to set. However, the user may set custom properties in this file. This gives the user the chance to set environment specific variables. The strategy is to setup one property file per environment. The property file can be referenced during execution as in the following:

ExecutePDTool.bat -exec ../resources/plans/myplan.dp -config deploy_TEST.properties

Some examples of deployment property files are shown below:

- deploy_TEST.properties ← custom variables section
  - **DEPLOY_TYPE=TEST** ← type of deployment used in module xml files
  - **SERVERID= TEST9400** ← servers.xml id used in plan file
  - VCONN=svn01 ← VCS connection id used in VCSModule.xml
- deploy_PROD.properties
  - DEPLOY_TYPE=PROD
  - SERVERID=PROD9410
  - VCONN=svn02

Server List – the servers.xml file contains a listing of all the servers used in the environment. The server "id" uniquely identifies the server instance. There are two strategies that can be taken with the server id. In both cases below, there is a different configuration property file for each environment.

(1) The "id" can be used as a parameter such as DEPLOY_TYPE=TEST. The variable DEPLOY_TYPE would be utilized for many different use cases within the deployment. This will be discussed in detail as this section progresses.

(2) A specific server variable can be set in the configuration property file such as SERVERID=TEST9400. This is shown in the servers.xml example below:

```
<servers>
  <server>
    <id>TEST9400</id>
    <hostname>testserver</hostname>
    <port>9400</port>
    <usage>Test</usage>
    <user>admin</user>
    <encryptedpassword>password</encryptedpassword>
    <domain>composite</domain>
    <cishome>/u01/composite/CIS6.2.0</cishome>
    <clustername>cluster2</clustername>
```

```
  <site>US East</site>
 </server>
 <server>
  <id>PROD9410</id>
  <hostname>prodserver</hostname>
  <port>9410</port>
  <usage>Production</usage>
  <user>admin</user>
  <encryptedpassword>password</encryptedpassword>
  <domain>composite</domain>
  <cishome>/u01/composite/CIS6.2.0</cishome>
  <clustername>cluster2</clustername>
  <site>US East</site>
 </server>
</servers>
```

Module XML files – each module has its own XML file because the definition of attributes is different for each module. There are two strategies that one can take with the XML files. One strategy is to have an XML file for each environment. The second and recommended strategy is to have a single module XML file that contains entries for each environment. There may be many different data sources. The idea is to identify the data source in a descriptive manner such as “orders” and then qualify it with the environment deployment type. So for TEST, the id would be “orders_TEST” and production would be “orders_PROD”. This strategy is shown in the example below with abbreviated XML:

```
<p1:DatasourceModule xmlns:p1="http://www.dvbu.cisco.com/ps/deploytool/modules">
  <datasource>  <datasource>
    <relationalDataSource>
      <id>orders_TEST</id>
      <resourcePath>/shared/test00/DataSources/ds_orders</resourcePath>
      <hostname>dbtesthost</hostname>
      <port>9408</port>
      <databaseName>orders</databaseName>
      <login>user1</login>
      <encryptedPassword>some-encrypted-password</encryptedPassword>
      <valQuery></valQuery>
        …
    </relationalDataSource>
  </datasource>
  <datasource>
    <relationalDataSource>
      <id>orders_PROD</id>
      <resourcePath>/shared/test00/DataSources/ds_orders</resourcePath>
      <hostname>dbprodhost</hostname>
      <port>9408</port>
      <databaseName>orders</databaseName>
      <login>user2</login>
      <encryptedPassword>a-different-encrypted-password</encryptedPassword>
      <valQuery></valQuery>
        …
    </relationalDataSource>
  </datasource>
</p1:DatasourceModule>
```

Deployment Plan File –the deployment plan file is used to orchestrate the execution. It contains a list of execution actions. Each action represents a stateless transaction to the Composite server to affect some change to the target server. The idea with the plan file is to construct the same plan across all environments (if possible). The plan file should use variables to reference things such as the server identifier “$SERVERID, the specific deployment environment “$DEPLOY_TYPE and VCS connections “$VCONN.

It is worth noting that there are times when the variable will need to be enclosed in the variable indicator such as $DEPLOY_TYPE$. PDTool looks for separator such as “/” to locate the end of a variable. If the plan references two variables next to each other, enclose them with a “$” at the beginning and end. For example when updating a multiple data sources within the same invocations as shown here: “orders_$DEPLOY_TYPE$, customers_$DEPLOY_TYPE$”. In

In this example the DataSourceModule.xml is being referenced. It shows the usage of $SERVERID to access the target Composite server and the $DEPLOY_TYPE$ variable to reference the correct orders data source information for TEST or PROD based on which deployment configuration property file was selected at execution time.

```
PASS    true ExecuteAction   updateDataSources  $SERVERID "orders_$DEPLOY_TYPE$"
        "$MODULE_HOME/DataSourceModule.xml" "$MODULE_HOME/servers.xml"
```

In this example the VCSModule.xml is being referenced which uses a VCS connection variable called $VCONN:

```
PASS    TRUE    ExecuteAction    vcsCheckout2       $SERVERID $VCONN /shared/test00 "Folder"
HEAD    "$MODULE_HOME/VCSModule.xml"  "$MODULE_HOME/servers.xml"
```

FUNCTIONAL MODULES

Functional Module Definitions for phase I

An explanation of each functional module and its actions are provided below.

Archive Module

The Archive Module performs various CIS archiving operations with CAR (Composite Archive) files such as backup, restore, import and export.

1. **Import (pkg_import)** – import a CAR file into a local or remote CIS instance with options.
2. **Export (pkg_export)** – export a CAR file from a local or remote CIS instance by designating a list of resources to export.
3. **Backup (backup_export)** – perform a complete backup of a local or remote CIS server instance.
4. **Restore (backup_import)** – perform a complete restore of a local or remote CIS server instance.

Data Source Module

The Data Source Module performs various actions with a CIS data source configuration such as updating data sources, enabling, and re-introspecting. The data source module can handle specific named relational data source attributes or generic name/type/value attributes for any type of data source. See Appendix A for a listing of Data Source Attributes.

1. **updateDataSources** – update a data source with either named relational attributes or generic attributes as defined by the DataSourceModule.xml.
2. **enableDataSources** – enable a data source.
3. **reIntrospectDataSources** – re-introspect a data source and add new resources such as tables to its schema container.
4. **generateDataSourcesXML** – given a starting CIS folder path, traverse the folder structure looking for DATA_SOURCE type resources and generate the list into the DataSourceModule.xml file. This is very useful for establishing the initial property file which can be easily tweaked for deploying into a different environment.

Group Module

The Group Module performs various actions with CIS group configuration such as creating, updating or deleting groups.

1. **createOrUpdateGroups** – create a group if it does not exist or update an existing group and apply any access right privileges for the group.

Right	Attribute	Description	Group or User
Read All Config	READ_ALL_CONFIG	View of CIS configuration settings, licenses	Administrator, Backup&Restore, Restore, Backup, Operations
Modify All Config	MODIFY_ALL_CONFIG	Change CIS configuration settings, licenses, and other server configuration	Administrator, Backup&Restore, Restore
Read All Resources	READ_ALL_RESOURCES	View of all resources, universal Read privilege, Full Server Backup, backup_import, Manager panels, execute any resource procedure, browse and edit resource services	Administrator, Backup&Restore, Restore, Backup
Modify All Resources	MODIFY_ALL_RESOURCES	Effective full privileges on all resources, change of privileges on any resource, change owner of a resource, import of privileges and copy/paste of resources with privileges (requires Modify All Users), restore/import (requires almost all other rights to	Administrator, Restore
Read All Status	READ_ALL_STATUS	View CIS current state, sessions, transactions, requests, caches, support diagnostics, query plan view, cluster status, view event, server, and storage logs, -profile option of the Server_Util, view of the following resource tables: SYS_CACHES, SYS_DATASO	Administrator, Backup&Restore, Backup, Operations, Developer
Modify All Status	MODIFY_ALL_STATUS	Manager, Server Overview actions (clear pool and test all data sources), view and clear of query plans and caches, terminate sessions, requests, transactions, view of the following resource tables: SYS_CACHES, SYS_DATASOURCE, SYS_STATISTICS, SYS_TRIGGERS,	Administrator
Read All Users	READ_ALL_USERS	Browse all lists of domains, groups, and users, (no domain or user passwords are exposed) using User Services or Composite Manager, full server backup (also requires Read All Resources and Read All Config), backup and restore (also requires Read All Resources)	Administrator, Backup&Restore, Restore, Backup
Modify All Users	MODIFY_ALL_USERS	Create/modify domains, groups, and users and their respective rights, change resource owner, import resources with users and their associated privileges (also requires Modify All Resources), paste preserving user privileges	Administrator, Backup&Restore, Restore
Unlock Resource	UNLOCK_RESOURCE	Enables unlocking of any locked resources regardless of the identity of the lock owner	Administrator

Table 3: Composite Group Rights

1. **addUsersToGroups** – add user(s) to an existing group.
2. **deleteGroups** – delete groups.
3. **deleteUsersFromGroups** – delete users from a group.
4. **generateGroupsXML** – generate the GroupModule.xml property file.

Privilege Module

The Privilege Module performs various actions with CIS resource configuration such as updating resource privileges.

1. **updateResourcePrivileges** – update the privileges on one more resources identified in the PrivilegeModule.xml property file.

Privilege	Description
READ	The Read privilege on a resource grants the ability to see that the resource exists.
WRITE	The Write privilege on a resource enables modification of the Composite resource definition that defines what and how the native resource may be used.
EXECUTE	Allow execution of a procedure.
SELECT	Allow submission of SQL selects to retrieve data.
UPDATE	Allow updating of data into the data source.
INSERT	Allow inserting of data into the data source.
DELETE	Allow deleting data in the data source.
GRANT	Allow the ability to grant privileges.

Table 4: Composite Access Privileges

1. **generatePrivilegesXML** – Generate the privileges XML file.

Rebind Module

The Rebind Module performs various actions with CIS such as rebind a specific resource, rebind all views in a folder, and rebind all procedures in a folder.

1. **rebindResources** – This procedure provides the capability to rebind the resources inside of the requested resource. For example, if a View points to a data source table, you may want to rebind to a different data source that has the same structure. This may be useful when redeploying from Dev to Test to Production or simply rebinding to a different development instance of the database. The RebindModule.xml provides the user the ability to define which resource to rebind, the type of resource and an array of target resources where the user defines the old path, old type, new path and new type for each target. One caveat with rebindResource is that both the old and new resources must exist for this procedure to execute properly.

2. **rebindFolders** – This procedure is used to rebind all of the resources (Views and/or Procedures) in a given starting source folder to a target rebind folder. For example, if all of the views in the Physical Views layer are pointing to a particular data source and you want to rebind them to point to a different data source folder then this procedure will accomplish that task. This procedure takes a starting resource folder and interrogates the folder to get a list of views or procedures in the folder. It then rebinds the source path to the target path. The source path resource does not have to exist for this method to execute properly.

3. **generateRebindXML** – Generate the RebindXML property file based on “rebindable” resources.

Regression Test Module

The Regression Test Module performs various actions with CIS regression testing configurations such as creating a regression test file and executing a regression test.

1. **executeRegressionTest** – Run a regression test for a given CIS Server. Before the regression is run, the regression text XML file is generated for that server. It contains all published views, procedures and web services for a given published datasource.
2. **createRegressionInputFile** – Create the regression test XML file for one published datasource on a given CIS server for a given user.

Resource Module

The Resource Module performs various actions with CIS resource configuration such as deleting, renaming and checking for existence.

1. **doResourcesExist** – given a list of resource paths in the ResourceModule.xml, validate that the resource exists in the target CIS instance. Throw an error if it does not.
2. **copyResources** – copy a resource in the target CIS instance.
3. **deleteResources** – delete a resource in the target CIS instance.
4. **renameResources** – rename a resource in the target CIS instance.
5. **moveResources** – move a resource in the target CIS instance.
6. **lockResources** – lock a resource in the target CIS instance.
7. **unlockResources** – unlock a resource in the target CIS instance.
8. **executeProcedure** – execute a CIS Data Service procedure.
9. **executeConfiguredProcedure** – execute a CIS Data Service procedure configured via the ResourceXML property file.
10. **createFolder –** Create all folders in the path associated with passed in resource path.
11. **createFolders** – Create all folders in the path associated with the passed in resource ids.

Resource Cache Module

The Resource Cache Module performs various actions with CIS resource cache configuration such as updating and refreshing the cache.

1. **clearResourceCache** – clear the resource cache.
2. **refreshResourceCache** – refresh the resource cache.
3. **updateResourceCache** – update the resource cache.
4. **updateResourceCacheEnabled** – enable/disable all caches in a starting folder.
5. **generateResourceCacheXML** – generate a list of resources that contain a cache.

Server Attribute Module

The Server Attribute Module performs various actions with CIS server attribute configuration such as updating server attributes. See Appendix B for a listing of Server Attributes.

1. **updateServerAttributes** – update the server attribute with a new value.
2. **generateServerAttributesXML** – generate the list of server attributes given a starting path.
3. **generateServerAttributeDefinitionsXML** – generate the list of server attribute defintions given a starting path.

Server Manager Module

The Server Module performs various actions with CIS server such as starting, stopping and restarting.

1. **startServer** – start the CIS instance.
2. **stopServer** – stop the CIS instance.
3. **restartServer** – restart the CIS instance.

Trigger Module

The Trigger Module performs various actions with CIS trigger configuration such as updating triggers or deleting triggers.

1. **enableTriggers** – enable a trigger.
2. **updateTriggers** – update the trigger configuration based on a trigger identifier list and the TriggerModule.xml. Trigger schedules are also maintained in the TriggerModule.xml.
3. **generateTriggersXML** – generate the list of Triggers and Trigger schedules that exist in a given path. Generate to the TriggerModule.xml file.

User Module

The User Module performs various actions with CIS group configuration such as creating, updating or deleting users.

1. **createOrUpdateUsers** – create a user if the do not exist otherwise update an existing user. The UserModule.xml provides a switch to force updating the password or not for a user.

2. **deleteUsers** – delete a user.

3. **generateUsersXML** – generate the list of users to the UserModule.xml file.

VCS Module

The VCS Module performs various version control actions using pre-installed VCS client software such as subversion or perforce. The equivalent methods that end in a "2" perform the same functionality but allow the user to pass in a VCS Connection ID which references the VCS connection information the VCSModule.xml. Otherwise, the methods not ending in "2" will utilize the singularly defined connection information found in the configuration property file such as "deploy.properties".

1. **vcsInitWorkspace[2]** – Initialize the VCS local workspace by linking it to the specific VCS repository project and then checking out the resources.

2. **vcsCheckout[2]** – Check out CIS artifacts from version control, perform a diff merger on those artifacts with the target CIS instance, build the car file according to the diff and finally import the car file into the target CIS instance. This activity supports deleting resources in the target CIS instance. This is the primary use case for version control in the context of promotion and deployment.

3. **vcsCheckin[2]** – Check in CIS artifacts from the designated CIS server into version control. It should be noted that while this capability exists it is not the normal use case for deployment purposes.

4. **vcsForcedCheckin[2]** – Force check in of artifacts from the designated CIS server into version control even if the local workspace is not at the same revision as the server. This may be necessary when you want to overwrite a change that was made by someone else. Again, this is not the normal use case for deployment.

5. **vcsPrepareCheckin[2]** – Prepare check in by synchronizing the local workspace with the Head of VCS server for a given path

Functional Module Definitions for phase II

An explanation of each functional module and its actions are provided below. Phase II modules have not yet been implemented.

Connector Module

The Connector Module performs various actions with CIS connector configuration such as create, update and delete connectors.

1. **createOrUpdateConnectors** – given one ore more connector configurations stored in ConnectorModule.xml, create a connector if it does not exist or update the connector if it already exists.
2. **deleteConnectors** – delete a connector identified in the ConnectorModule.xml property file.

Domain Module

The Domain Module performs various actions with CIS domain configuration such as creating, updating or deleting domains.

1. **createOrUpdateDomains** – create a domain if it does not exit or update an existing domain.
2. **deleteDomains** – delete a domain.
3. **generateDomainsXML** – generate the domains that exist in the CIS instance into the DomainModule.xml file.

Resource Statistics Module

The Resource Statistics Module performs various actions with CIS resource cache configuration such as updating and refreshing the cache.

1. **updateResourceStatistics** – update the resource statistics.
2. **refreshResourceStatistics** – refresh the resource statistics.
3. **clearResourceStatistics** – clear the resource statistics.

CONCLUSION

Concluding Remarks

The PS Promotion and Deployment Tool is a set of pre-built modules intended to provide a turn-key experience for promoting CIS resources from one CIS 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 Composite Professional Services for the advancement of the "***PS Promotion and Deployment Tool***".

APPENDIX A

Data Source Attribute List

The following is a list of data source attributes and their types. To discover more information about a particular data source and its attributes, invoke the following Composite web service API method and provide it the path to the data source, the type=DATA_SOURCE and the detail=FULL: /services/webservices/system/admin/resource/operations/getResource.

Relational Data Source

Description

Name	Type	Example Value
connPoolTimeout	INTEGER	30
connPoolMinSize	INTEGER	10

Table 5.1: Data Source Attribute List

CSV File Source

Description

Name	Type	Example Value
Root	STRING	/tmp/FixedFileHeader
Filters	STRING	*.csv,*.txt

Table 5.2: Data Source Attribute List

XML File Source

Description

Name	Type	Example Value
url	STRING	file:///C:/CompositeSoftware/CIS5.2.0/docs/examples/productCatalog.xml
schemaLocation	STRING	file:///C:/CompositeSoftware/CIS5.2.0/docs/examples/productCatalog.xsd

Table 5.3: Data Source Attribute List

APPENDIX B

Server Attributes

The following is a list of server attributes and their types. To discover more attributes, recursively (repeatedly) invoke the following Composite web service API method and provide it a server attribute path: /services/webservices/system/admin/server/operations/getServerAttributes.

Change Management Service

Root: /cms

Path	Type	Example Value
/cms/central/enabled	BOOLEAN	true or false
/cms/edge		
/cms/request		

Table 6.1: Server Attribute Table

Composite Discovery

Root: /discovery

Path	Type	Example Value
/discovery/index/maximumConcurrentTasks	INTEGER	10
/discovery/relationship/scoreMinimum	INTEGER	15

Table 6.2: Server Attribute Table

Monitor

Root: /monitor

Path	Type	Example Value
/monitor/client/connection/allowRecentUserNames	BOOLEAN	true
/monitor/collector/enabled	BOOLEAN	true
/monitor/server/enabled	BOOLEAN	true

Table 6.3: Server Attribute Table

Server

Root: /server

Path	Type	Example Value
/server/sql/language/caseSensitive	BOOLEAN	false
/server/sql/language/ignoreTrailingSpaces	BOOLEAN	true
/server/memory/heap/javaHeapMaxOnServerRestart	INTEGER	4096
/server/webservices/baseURI	STRING	http://www.dvbu.cisco.com/ps.deploytool

Path	Type	Example Value
/server/config/net/wsdlhostname	STRING	deployserver.compositesw.com

Table 6.4: Server Attribute Table

Data Sources

Root: /sources

Path	Type	Example Value
/sources/common/defaultCommitRowLimit	INTEGER	2000
/sources/oracle/introspectAllObjects	BOOLEAN	true

Table 6.5: Server Attribute Table

Studio

Root: /studio

Path	Type	Example Value
/studio/data/cursorFetchLimit	INTEGER	2000
/studio/data/xmlTextLimit	INTEGER	100000

Table 6.6: Server Attribute Table

ABOUT COMPOSITE SOFTWARE

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

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

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

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

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

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

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

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

Printed in USA CXX-XXXXXX-XX 10/11