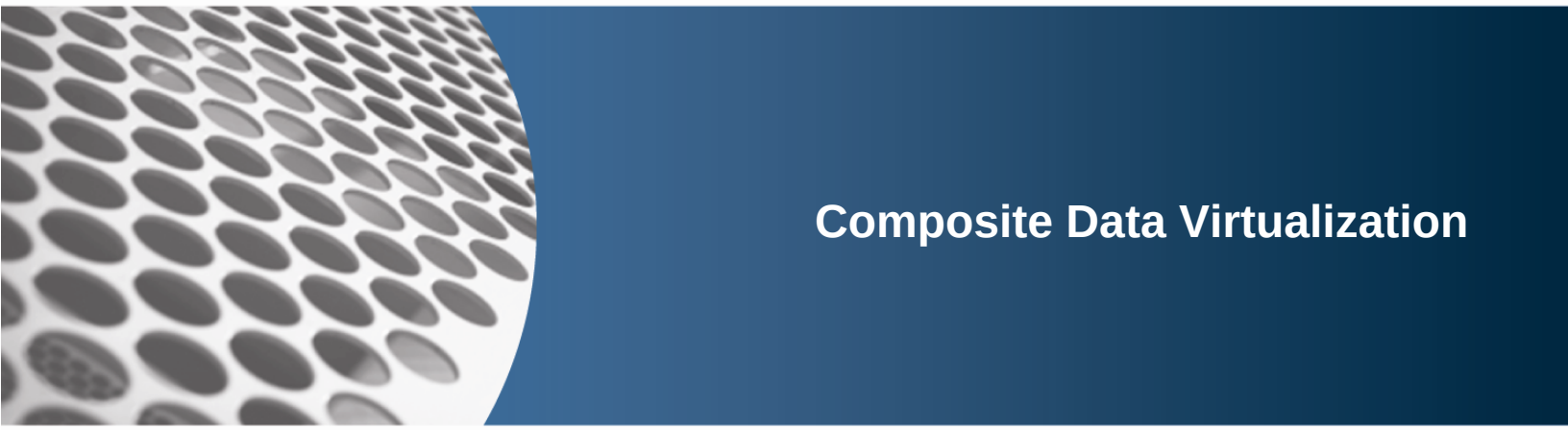
WHITE PAPER

COMPOSITE SOFTWARE




# *Composite PS Promotion and Deployment Tool*

# *Server Attribute Module User Guide*

Composite Professional Services

November 2014

## TABLE OF CONTENTS

## DOCUMENT CONTROL

### Version History

| Version | Date | Author | Description |
|---|---|---|---|
| 1.0 | 6/6/2011 | Mike Tinius | Initial revision for Server Attribute Module User Guide |
| 1.0.1 | 8/1/2011 | Mike Tinius | Revisions due Architecture changes |
| 3.0 | 8/21/2013 | Mike Tinius | Updated docs to Cisco format |
| 3.1 | 2/18/2014 | Mike Tinius | Prepare docs for open source. |
| 3.2 | 3/24/2014 | Mike Tinius | Changed references of XML namespace to www.dvbu.cisco.com |
| 3.3 | 11/17/2014 | Mike Tinius | Update license. |

### Related Documents

| Document | File Name | Author |
|---|---|---|
| *Composite PS Promotion and Deployment Tool User's Guide v1.0* | *Composite PS Promotion and Deployment Tool User's Guide v1.0.pdf* | Mike Tinius |
| | | |

### Composite Products Referenced

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

# INTRODUCTION

## *License*



This software is released under the Eclipse Public License. The details can be found in the file LICENSE. Any dependent libraries supplied by third parties are provided under their own open source licenses as described in their own LICENSE files, generally named .LICENSE.txt. The libraries supplied by Cisco as part of the Composite Information Server/Cisco Data Virtualization Server, particularly csadmin-XXXX.jar, csarchive-XXXX.jar, csbase-XXXX.jar, csclient-XXXX.jar, cscommon-XXXX.jar, csext-XXXX.jar, csjdbc-XXXX.jar, csserverutil-XXXX.jar, csserver-XXXX.jar, cswebapi-XXXX.jar, and customproc-XXXX.jar (where -XXXX is an optional version number) are provided as a convenience, but are covered under the licensing for the Composite Information Server/Cisco Data Virtualization Server. They cannot be used in any way except through a valid license for that product.

This software is released AS-IS!. Support for this software is not covered by standard maintenance agreements with Cisco. Any support for this software by Cisco would be covered by paid consulting agreements, and would be billable work.

## *Purpose*

The purpose of the Server Attribute Module User Guide is to demonstrate how to effectively use the Server Attribute Module and execute actions. Server Attributes are typically modified within the Studio Administration and Configuration. There are many different types of server attributes ranging from Composite Discover, Monitor, Server, Studio, Data Sources and the Change Management Service. The Server Attribute module will allow the update of these configuration items to be automated via scripts instead of having to manually set them in Composite Studio.

## *Audience*

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

- Architects
- Developers
- Administrators.
- Operations personnel.

## SERVER ATTRIBUTE MODULE DEFINITION

### *Method Definitions and Signatures*

1. **updateServerAttributes**

   Update a CIS server and studio configuration based on the list of server attributes ids that are passed in. Note: An exception will be thrown if an attribute is attempted to be updated that has an update rule of READ_ONLY. Information on whether an attribute is “READ_ONLY” or “READ_WRITE” can be found by generating the attribute definitions.

```
@param serverId - target server id from server XML configuration file

@param serverAttributeIds - list of server attributes Ids (comma
separated server attributes Ids).  Server Attribute Ids rules:

1. csv string like sa1,sa2 (only server attributes Ids in this list
   are processed.)

2. '*' or whatever is configured to indicate all resources (all server
   attribute Ids are processed in the module XML configuration file.

3. csv string with '-' or whatever is configured to indicate exclude
   resources as prefix like -sa1,sa2 (all resources in the module XML
   configuration file are processed except this passed in list)

@param pathToServerAttributesXML - path to the server attribute module
XML configuration file

@param pathToServersXML - path to the server XML configuration file

@throws CompositeException


public void updateServerAttributes(String serverId, String
serverAttributeIds, String pathToServerAttributesXML, String
pathToServersXML) throws CompositeException;
```

2. **generateServerAttributesXML**

   Generate a file containing all server attributes for a given starting path and a given update rule.

```
@param serverId - target server id from server XML configuration file

@param startPath - starting path of the server attribute folder e.g.
/server

@param pathToServerAttributeXML path including name to the server
attributes XML which will be generated

@param pathToServersXML - path to the server XML configuration file

@param updateRule - the type of rule described by the attribute
definition

READ_ONLY - only get attributes where updateRule=READ_ONLY
```

```
READ_WRITE - only get attributes where updateRule=READ_WRITE (this
should be considered the default behavior because READ_ONLY rules
cannot be updated)

* - get all attributes

@throws CompositeException


public void generateServerAttributesXML(String serverId, String
startPath, String pathToServerAttributeXML, String pathToServersXML,
String updateRule) throws CompositeException;
```

The following starting paths provide a listing of high-level categories which can be configured:

/cms – Composite Change Management Service configuration attributes
/discovery – Composite Discovery configuration attributes
/monitor – Composite Monitor configuration attributes
/server – Composite Information Server configuration attributes
/sources – Composite Data Sources configuration attributes
/studio – Composite Studio configuration attributes

The following structure is an example of what is generated:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ServerAttributeModule
xmlns:ns2="http://www.dvbu.cisco.com/ps/deploytool/modules">
    <serverAttribute>
        <id>studio1</id>
        <name>/studio/data/cursorFetchLimit</name>
        <type>INTEGER</type>
        <value>2000</value>
    </serverAttribute>
</ns2:ServerAttributeModule>
```

3. **generateServerAttributeDefinitionsXML**

Generate a file containing all server attribute definitions for a given starting path and a given update rule. Attribute Definitions describe how an attribute is defined and whether it can be updated or not.

```
@param serverId - target server id from server XML configuration file

@param startPath - starting path of the server attribute folder e.g.
/server

@param pathToServerAttributeXML - path including name to the server
attribute definition XML which will be generated

@param pathToServersXML - path to the server XML configuration file

@param updateRule - the type of rule described by the attribute
definition

READ_ONLY - only get attribute definitions where updateRule=READ_ONLY
```

```
READ_WRITE - only get attribute definitions where
updateRule=READ_WRITE

* - get all attribute definitions

@throws CompositeException


public void generateServerAttributeDefinitionsXML(String serverId,
String startPath, String pathToServerAttributeXML, String
pathToServersXML, String updateRule) throws CompositeException;
```

The following starting paths provide a listing of high-level categories of attribute definitions:

/cms – Composite Change Management Service configuration attributes
/discovery – Composite Discovery configuration attributes
/monitor – Composite Monitor configuration attributes
/server – Composite Information Server configuration attributes
/sources – Composite Data Sources configuration attributes
/studio – Composite Studio configuration attributes

The following structure is an example of what is generated:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ServerAttributeModule
xmlns:ns2="http://www.dvbu.cisco.com/ps/deploytool/modules">
    <serverAttributeDef>
        <id>studio1</id>
        <name>/studio/data/cursorFetchLimit</name>
        <type>INTEGER</type>
        <annotation>This value affects how many rows can be fetched by Studio
when looking at the results for a SQL request or procedure
output.</annotation>
        <defaultValue>1000</defaultValue>
        <displayName>Cursor Fetch Limit</displayName>
        <minValue>1</minValue>
        <unitName>rows</unitName>
        <updateRule>READ_WRITE</updateRule>
    </serverAttributeDef>
</ns2:ServerAttributeModule>
```

General Notes:

The arguments pathToServerAttributeXML 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.

## SERVER ATTRIBUTE MODULE XML CONFIGURATION

A full description of the PDToolModule XML Schema can be found by reviewing PDTool/docs/PDToolModules.xsd.html.

### *Description of the ServerAttributeModule XML for (serverAttribute)*

The ServerAttributeModule XML provides a structure “serverAttribute” for updating the CIS server attributes and generating the base attribute XML configuration file. Then entry point node is called ServerAttributeModule and contains a choice of children [serverAttribute | serverAttributeDef].

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ServerAttributeModule
xmlns:ns2="http://www.dvbu.cisco.com/ps/deploytool/modules">
<!-- Example of Simple Value Type -->
<!-- Contains a simple type and value -->
    <serverAttribute>
        <id>studio1</id>
        <name>/studio/data/cursorFetchLimit</name>
        <type>INTEGER</type>
        <value>2000</value>
    </serverAttribute>

<!-- Example of Value Array Type -->
<!-- Contains an item list of values -->
    <serverAttribute>
        <id>monitor43</id>
        <name>/monitor/server/connection/secondaryHosts</name>
        <type>STRING_ARRAY</type>
        <valueArray>
            <item>localhost</item>
            <item>host2</item>
        </valueArray>
    </serverAttribute>

<!-- Example of Value List Type -->
<!-- Contains an item list with type and value -->
    <serverAttribute>
        <id>studio2</id>
        <name>/studio/data/examplelist</name>
        <type>LIST</type>
        <valueList>
            <item>
                <type>STRING</type>
                <value>a1</value>
            </item>
            <item>
                <type>STRING</type>
                <value>b1</value>
            </item>
        </valueList>
    </serverAttribute>

<!-- Example of Value Map Type -->
<!-- Contains an entry list of Key/Value pairs each containing type
and value -->
```

```
    <serverAttribute>
        <id>studio3</id>
        <name>/studio/data/examplemap</name>
        <type>MAP</type>
        <valueMap>
            <entry>
                <key>
                    <type>STRING</type>
                    <value>key1</value>
                </key>
                <value>
                    <type>STRING</type>
                    <value>value1</value>
                </value>
            </entry>
            <entry>
                <key>
                    <type>STRING</type>
                    <value>key2</value>
                </key>
                <value>
                    <type>STRING</type>
                    <value>value2</value>
                </value>
            </entry>
       </valueMap>
    </serverAttribute>
</ns2:ServerAttributeModule>
```

### *Description of the ServerAttributeModule XML for (serverAttributeDef)*

The ServerAttributeModule XML provides a structure “serverAttributeDef” for generating the base attribute definition XML. This node is not used when updating server attributes. The entry point node is called ServerAttributeModule and contains a choice of children [serverAttribute | serverAttributeDef].

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ServerAttributeModule
xmlns:ns2="http://www.dvbu.cisco.com/ps/deploytool/modules">
<!-- Example of Simple Value Type -->
<!-- Contains a simple type and value -->
    <serverAttributeDef>
        <id>studio1</id>
        <name>/studio/data/cursorFetchLimit</name>
        <type>INTEGER</type>
        <annotation>This value affects how many rows can be fetched by Studio when
looking at the results for a SQL request or procedure output.</annotation>
        <defaultValue>1000</defaultValue>
        <displayName>Cursor Fetch Limit</displayName>
        <minValue>1</minValue>
        <unitName>rows</unitName>
        <updateRule>READ_WRITE</updateRule>
    </serverAttributeDef>

<!-- Example of Value Array Type -->
<!-- Contains an item list of values -->
    <serverAttributeDef>
        <id>monitor43</id>
        <name>/monitor/server/connection/secondaryHosts</name>
```

```
        <type>STRING_ARRAY</type>
        <annotation>The list of hosts that are known to be in the cluster that the
Monitor Server connects with.  If the Monitor Server fails to connect with a CIS
instance when it first starts up, it will try one of the other hosts in this list.
This list is automatically populated when a the Monitor Server is configured to
monitor a CIS instance as a cluster.</annotation>
        <displayName>Secondary CIS Hosts</displayName>
        <updateRule>READ_WRITE</updateRule>
    </serverAttributeDef>

<!-- Example of Value List Type -->
<!-- Contains an item list with type and value -->
    <serverAttributeDef>
        <id>studio2</id>
        <name>/studio/data/examplelist</name>
        <type>LIST</type>
        <annotation>Example List</annotation>
        <displayName>Example List</displayName>
        <unitName>List</unitName>
        <updateRule>READ_WRITE</updateRule>
    </serverAttributeDef>

<!-- Example of Value Map Type -->
<!-- Contains an entry list of Key/Value pairs each containing type
and value -->
    <serverAttributeDef>
        <id>studio3</id>
        <name>/studio/data/examplemap</name>
        <type>MAP</type>
        <annotation>Example Map</annotation>
        <displayName>Example Map</displayName>
        <unitName>Key-Value Map</unitName>
        <updateRule>READ_WRITE</updateRule>
    </serverAttributeDef>
</ns2:ServerAttributeModule>
```

### *Attributes of Interest*

***id*** – this value is generated by the "generate…XML()" methods. It uses the first token in the server attribute path + a sequence number when generating. The value must be unique within the XML configuration file. The values are not guaranteed to be the same for different executions of the "generate...XML()" methods. The user may configure an XML file by hand if they wish but bear in mind that the id must be unique.

***name*** – this value is the actual server attribute path and is unique across all server attribute configurations. The same path describes the server attribute and the server attribute definition.

***updateRule*** – this value is only provided by the server attribute definition. It is used by the method generateServerAttributesXML to determine which server attributes to generate to the configuration file based on the udpateRule input argument.

### *Attribute Value Restrictions*

***updateRule*** – contains either READ_ONLY or READ_WRITE.

***type*** – all occurrences of this element have the following restrictions:

```
<xs:simpleType name="AttributeTypeSimpleType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="BOOLEAN"/>
        <xs:enumeration value="BOOLEAN_ARRAY"/>
        <xs:enumeration value="BYTE"/>
        <xs:enumeration value="BYTE_ARRAY"/>
        <xs:enumeration value="DATE"/>
        <xs:enumeration value="DATE_ARRAY"/>
        <xs:enumeration value="DOUBLE"/>
        <xs:enumeration value="DOUBLE_ARRAY"/>
        <xs:enumeration value="FILE_PATH_STRING"/>
        <xs:enumeration value="FLOAT"/>
        <xs:enumeration value="FLOAT_ARRAY"/>
        <xs:enumeration value="FOLDER"/>
        <xs:enumeration value="INT_ARRAY"/>
        <xs:enumeration value="INTEGER"/>
        <xs:enumeration value="LIST"/>
        <xs:enumeration value="LONG"/>
        <xs:enumeration value="LONG_ARRAY"/>
        <xs:enumeration value="MAP"/>
        <xs:enumeration value="NULL"/>
        <xs:enumeration value="OBJECT"/>
        <xs:enumeration value="PASSWORD_STRING"/>
        <xs:enumeration value="PATH_STRING"/>
        <xs:enumeration value="SET"/>
        <xs:enumeration value="SHORT"/>
        <xs:enumeration value="SHORT_ARRAY"/>
        <xs:enumeration value="STRING"/>
        <xs:enumeration value="STRING_ARRAY"/>
        <xs:enumeration value="UNKNOWN"/>
    </xs:restriction>
</xs:simpleType>
```

## HOW TO EXECUTE

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 ServerAttributeModule.xml that was described in the previous section.

### *Script Execution*

The full details on property file setup and script execution can be found in the document "*Composite PS Promotion and Deployment Tool User's Guide v1.0.pdf*". The abridged version is as follows:

Windows: ExecutePDTool.bat -exec ../resources/plans/UnitTest-ServerAttribute.dp

Unix: ./ExecutePDTool.sh -exec ../resources/plans/UnitTest-ServerAttribute.dp

***Properties File (UnitTest-ServerAttribute.dp):***

Property File Rules:

```
# ----------------------------
# UnitTest-ServerAttributes.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.
#
```

Property 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.
```

Property File Example:

```
# ----------------------------------------
# Begin task definition list:
# ----------------------------------------
#
PASS   FALSE  ExecuteAction generateServerAttributesXML $SERVERID "/"
$MODULE_HOME/getServerAttributeModule.xml $MODULE_HOME/servers.xml "READ_WRITE"
#
PASS   FALSE  ExecuteAction generateServerAttributeDefinitionsXML   $SERVERID "/"
$MODULE_HOME/getServerAttributeDefinitionModule.xml $MODULE_HOME/servers.xml "*"
#
PASS   FALSE  ExecuteAction updateServerAttributes $SERVERID "server1"
$MODULE_HOME/ServerAttributeModule.xml $MODULE_HOME/servers.xml
```

### *Ant Execution*

The full details on build file setup and ant execution can be found in the document "*Composite PS Promotion and Deployment Tool User's Guide v1.0.pdf*". The abridged version is as follows:

Windows: ExecutePDTool.bat -ant ../resources/plans/build-ServerAttribute.xml

Unix: ./ExecutePDTool.sh -ant ../resources/plans/build-ServerAttribute.xml

***Build File:***

```
<?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="&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="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="serverAttributes"          value="studio1,studio2"/>
  <property name="pathToGenServerAttributeXML"
       value="${MODULE_HOME}/getServerAttributeModule.xml"/>
  <property name="pathToGenServerAttributeDefXML"
       value="${MODULE_HOME}/getServerAttributeDefModule.xml"/>

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

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

```
  <target name="default" description="Update CIS with environment specific parameters">

    <!-- Execute Line Here -->
  <executeJavaAction description="Update"
        action="updateServerAttributes"
        arguments="${SERVERID}^${serverAttributes}^${pathToServerAttributeXML}^${pathToServersXM
L}"     endExecutionOnTaskFailure="TRUE" />

    <!-- Windows or UNIX: Entire list of actions
  <executeJavaAction description="Generate Atribute" action="generateServerAttributesXML"

        arguments="${SERVERID}^/^${pathToServerAttributeXML}^${pathToServersXML}^READ_WRITE"
                    endExecutionOnTaskFailure="TRUE" />

  <executeJavaAction description="Generate Attribute Definitions"
        action="generateServerAttributeDefinitionsXML"
        arguments="${SERVERID}^/^${pathToServerAttributeDefsXML}^${pathToServersXML}^*"
                    endExecutionOnTaskFailure="TRUE" />

  <executeJavaAction description="Update"
        action="updateServerAttributes"
        arguments="${SERVERID}^${serverAttributes}^${pathToServerAttributeXML}^${pathToServersXM
L}"     endExecutionOnTaskFailure="TRUE" />
    -->
  </target>
</project>
```

### *Module ID Usage*

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

- 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   FALSE  ExecuteAction updateServerAttributes $SERVERID "server1,server2"
    $MODULE_HOME/ServerAttributeModule.xml $MODULE_HOME/servers.xml
    ```

    Example Ant build file

    ```
    <executeJavaAction description="Update"        action="updateServerAttributes"
    arguments="${SERVERID}^server1,server2^${pathToServerAttributeXML}^${pathToSer
    versXML}"
    ```

- 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   FALSE  ExecuteAction       updateDataSources $SERVERID "*"
          "$MODULE_HOME/DataSourceModule.xml" "$MODULE_HOME/servers.xml"
    ```

    Example Ant build file

```
<executeJavaAction description="Update"        action="updateServerAttributes"
arguments="${SERVERID}^*^${pathToServerAttributeXML}^${pathToServersXML}"
```

- 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   FALSE  ExecuteAction updateServerAttributes $SERVERID "-
server1,server2" $MODULE_HOME/ServerAttributeModule.xml
$MODULE_HOME/servers.xml
```

Example Ant build file

```
<executeJavaAction description="Update"        action="updateServerAttributes"
arguments="${SERVERID}^-server1,server2^${pathToServerAttributeXML}^
${pathToServersXML}"
```

## EXAMPLES

The following are common scenarios when using the ServerAttributeModule.

### *Scenario 1 – Generate Server Attributes XML*

**Description:**

Generate the server attributes xml for "/studio" properties and READ_WRITE update rule.

**XML Configuration Sample:**

Not applicable for this example.

**Execution Sample:**

Unix: ./PDTool.sh UnitTest-ServerAttribute.dp

Property file setup for UnitTest-ServerAttribute.dp:

```
# ----------------------------------------
# Begin task definition list for UNIX:
# ----------------------------------------
# Generate READ_WRITE server attributes to ServerAttributeModule2.xml
PASS   FALSE  ExecuteAction generateServerAttributesXML $SERVERID "/studio"
$MODULE_HOME/ServerAttributeModule2.xml $MODULE_HOME/servers.xml "READ_WRITE"
```

**Results Expected:**

The file getServerAttributeModule.xml is produced with only /studio server attributes populated.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ServerAttributeModule
xmlns:ns2="http://www.dvbu.cisco.com/ps/deploytool/modules">
    <serverAttribute>
        <id>studio1</id>
        <name>/studio/data/cursorFetchLimit</name>
        <type>INTEGER</type>
        <value>2000</value>
    </serverAttribute>
    <serverAttribute>
        <id>studio2</id>
        <name>/studio/data/examplelist</name>
        <type>LIST</type>
        <valueList/>
    </serverAttribute>
    <serverAttribute>
        <id>studio3</id>
        <name>/studio/data/examplemap</name>
        <type>MAP</type>
```

```
        <valueMap/>
    </serverAttribute>
    <serverAttribute>
        <id>studio4</id>
        <name>/studio/data/rowFetchSize</name>
        <type>INTEGER</type>
        <value>2000</value>
    </serverAttribute>
    <serverAttribute>
        <id>studio5</id>
        <name>/studio/data/xmlTextLimit</name>
        <type>INTEGER</type>
        <value>10000</value>
    </serverAttribute>
    <serverAttribute>
        <id>studio6</id>
        <name>/studio/lock/enabled</name>
        <type>BOOLEAN</type>
        <value>true</value>
    </serverAttribute>
</ns2:ServerAttributeModule>
```

### *Scenario 2 – Update Server Attributes*

**Description:**

Update the CIS configuration for specific studio server attributes.

**XML Configuration Sample:**

Use the Server Attribute XML file that was generated in scenario 1. Modify the following attributes so that you can tell they have been changed:

```
        <id>studio1</id>
        <name>/studio/data/cursorFetchLimit</name>
        <value>1500</value>
```

```
        <id>studio5</id>
        <name>/studio/data/xmlTextLimit</name>
        <value>200000</value>
```

**Execution Sample:**

Unix: ./ExecutePDTool.sh -exec ../resources/plans/UnitTest-ServerAttribute.dp

Property file setup for UnitTest-ServerAttribute.dp:

```
# ----------------------------------------
# Begin task definition list for UNIX:
# ----------------------------------------
```

```
# Update server attributes defined by the list "studio1,studio5"
PASS  FALSE ExecuteAction updateServerAttributes $SERVERID
"studio1,studio5" $MODULE_HOME/ServerAttributeModule2.xml
$MODULE_HOME/servers.xml
```

**Results Expected:**

The script will report “PASS” for the execution of this action.  Open Composite Studio and review the changes for Studio.

## EXCEPTIONS AND MESSAGES

The following are common exceptions and messages that may occur.

### Wrong Number of Arguments:

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

## 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***".

## 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