HP Server Automation

for the HP-UX, IBM AIX, Red Hat Enterprise Linux, Solaris, SUSE Linux Enterprise Server, VMware, and Windows® operating systems

Software Version: 9.0

Application Configuration User Guide

Document Release Date: June 2010

Software Release Date: June 2010

Documentation Updates

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://support.openview.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at **http://www.hp.com/go/hpsoftwaresupport**

This web site provides contact information and details about the products, services, and support that HP Software offers. HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests

- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

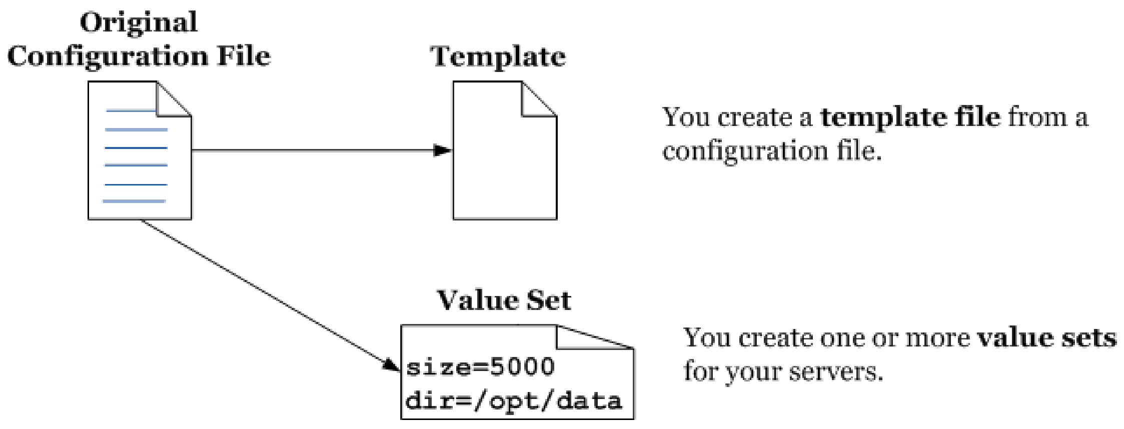
Contents

1 Quick Start to Application Configuration

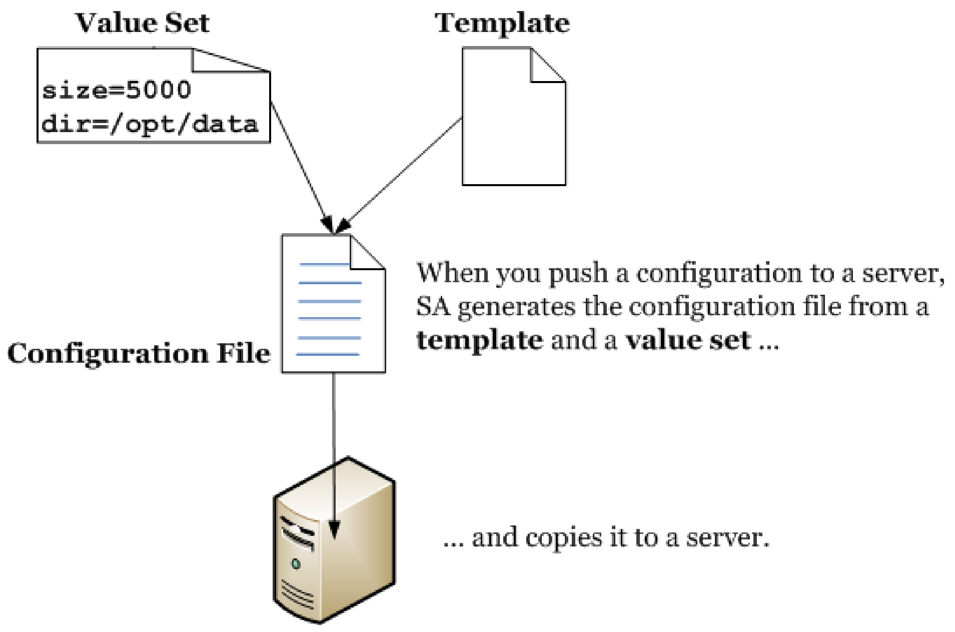
This chapter gives you the high level view of Application Configurations. It explains the steps you need to take to set up and manage your application configurations.

- For detailed instructions on how to perform these tasks, see Application Configuration Tasks on page 15.
- For background on Application Configurations, see Application Configuration Concepts on page 33.

To create and use application configurations, you must first create a **template** file and a **value set** as shown in the following diagram. A template is a model of a configuration file. A value set is the data values that will be merged with the template to create an instance of the configuration file to be pushed to a server. See below for detailed instructions.


Original Configuration File
Template
You create a **template file** from a configuration file.
Value Set
size=5000
dir=/opt/data
You create one or more **value sets** for your servers.

After creating a template and value set, you **attach** the application configuration to one or more servers and **push** the configurations to the servers, as shown in the following diagram. See below for detailed instructions.


Value Set
size=5000
dir=/opt/data
Template
Configuration File
When you push a configuration to a server, SA generates the configuration file from a **template** and a **value set** ...
... and copies it to a server.

To create and use application configurations, perform the following steps:

1 **Determine the configuration files you want to manage**. Choose the application or system configuration files you want to manage. For example, for the Apache Web server, you could manage the http.conf, password.conf, obj.conf, mimetypes, and magnus.conf files.

2 **Create a template file for each configuration file**. For each configuration file you want to manage, create a template file based on the configuration file. The template is a model of your original configuration file with place-holders for values that vary between servers. The template specifies what values in the configuration file are fixed and what values are variable and will be different on different servers. See also:

— Creating a Configuration Template on page 16.

— For XML configuration files, create an XML or XML-DTD template file. For more information, see Managing XML Configuration Files on page 61.

— For other configuration files, create a template file using the Configuration Modeling Language (CML). For more information, see CML Reference on page 129 and CML Tutorial 2 - Create a Template of a Web Server Configuration File on page 95.

— About Configuration Templates and Script Templates on page 34.

3 **Create a template file for each script.** In some cases you may need to run a script before or after updating a configuration file. You must create a template file from the scripts using CML. If you do not need to run any scripts with your configuration files, then skip this step. See also:

— Creating a Template from a Script on page 21.

— About Configuration Templates and Script Templates on page 34.

— About Running Scripts with Application Configurations on page 49.

4. **Import your template files into the SA Library**. Once you have created all of your templates from the configuration files and scripts, import your templates into the SA Library. Or you can create them directly in the SA Client. For more information, see Importing and Validating a Template File on page 18.
5. **Create an Application Configuration object.** An Application Configuration object is simply a container that holds one or more templates. See also:
   - Creating an Application Configuration on page 15.
   - About Application Configuration Objects on page 33.
6. **Add your templates to an Application Configuration object**. Once you have created and imported your templates into the SA Library, add them to an Application Configuration object. You also specify the order in which the files are installed and when the scripts, if any, are executed. See also:
   - Adding or Removing Templates from an Application Configuration on page 19.
7. **Attach application configurations to servers**. Once you have created and configured your Application Configuration, attach it to the servers that use it. This creates an instance of the application configuration. You can have multiple instances of an application configuration on a server. Each instance can be for a different purpose. For example, you could have an instance that configures a staging version of the application and an instance that configures a production version of the application. Or if you had three instances of the application running on a server, you could have three instances of the application configuration to configure each instance of the application. See also:
   - Attaching an Application Configuration to a Server or Device Group on page 23.
   - Using Application Configurations in Software Policies on page 57.
8. **Create value sets for your servers**. Set the values that will be placed into the template to generate the actual configuration file that will be pushed to the server. You can set default values at multiple levels that get inherited unless they are overridden at a lower level. See also:
   - Importing Value Sets from an Existing Configuration File on page 42.
   - About Value Sets on page 36
   - Value Set Levels and Value Set Inheritance on page 36
9. **Compare the actual configuration files with the configuration template**. (Optional) You can easily compare a configuration template with the actual configuration file on the server and see if there are any differences. This shows you the contents of the configuration file on the server and the values that will be copied to the server when you push the application configuration to the server. See also:
   - Comparing a Configuration Template with a Target Configuration File - Preview on page 29.
   - Comparing Two Configuration Templates on page 30.
10. **Push configuration changes**. To update the configuration files on servers you “push” the application configuration to those servers. No changes are made to the actual configuration files on the server until you push those changes to the servers. Application configuration changes can be pushed to individual servers or groups of servers. See also:
    - Pushing Application Configurations on page 26.
    - About Pushing Application Configurations to Servers on page 51.

11 **Audit configurations, monitor compliance and remediate**. To audit your configurations, monitor compliance and remediate configurations that have changed, see Application Configuration Compliance on page 51 and "Audit and Remediation" in the *SA User Guide: Application Automation*.

2 Application Configuration Tasks

This section describes how to perform the following tasks:

Creating an Application Configuration

An application configuration is a container for configuration template files and, optionally, scripts that are executed when the application configuration is pushed to servers. For details, see About Application Configuration Objects on page 33.

To create an application configuration, perform the following steps:

1. From the SA Client navigation pane, select Library and then select the By Type tab.
2. Locate and open the Application Configuration node. Open the Configurations node. Open the operating system group and select the operating system that the application configuration applies to. Note that an Application Configuration can apply to multiple operating systems. You can specify this in a later step.
3. Select the **Actions ➤ New** menu. This displays the File Configuration screen where you can specify the properties and contents of the Application Configuration.

4. In the Properties view, specify the name and description of the Application Configuration. In addition, specify the following:
   - **Location:** Specify where in the SA Library you want to store the Application Configuration.
   - **Version**: The version can be any string you use for tracking changes to the Application Configuration. The version is not incremented automatically.
   - **OS**: Specify which operating systems this application configuration applies to. Only servers with the specified operating systems will be able to use the Application Configuration. Only templates with at least one OS setting the same as this setting will be able to be contained in the application configuration.
   - **Availability**: Use this setting to keep track of which Application Configurations are tested and ready to be used and which are not yet tested or are deprecated. This setting does not change what can be done with the Application Configuration.
   - **Localization Files**: Select the "+" button to add templates for generating configuration files in local languages. For details, see Localizing Configuration File Element Names on page 30.
5. In the Content view, select the **Actions ➤ Add** menu or the "+" button to add a template to the Application Configuration.
   - Use the "-" button to remove the selected configuration templates.
   - Use the up and down arrows to change the order in which the configuration templates are installed.
   - Select the Template Preview view to see the contents of the selected template file.
   - Select the File Values view to see what values will be placed into the selected template file to generate the configuration file.
   - Select the File Preview view to see what the selected configuration file will look like with the current value set.
6. Select **File ➤ Save** to save your Application Configuration.

Creating a Configuration Template

A configuration template is similar to a native application configuration file, but with its variable parts "templatized" with the Configuration Modeling Language (CML) and instructions for moving values between the configuration file and the value set. See About Configuration Templates and Script Templates on page 34.

Any scripts that you want executed with the application configuration must also be written in CML template format. For more information, see Creating a Template from a Script on page 21.

To create a configuration template, perform the following steps:

1. From the SA Client navigation pane, select Library and then select the By Type tab.
2. Locate and open the Application Configuration node. Open the Templates node. Open the operating system group and select the operating system where the template file will be used. Note that a template can apply to multiple operating systems. You can specify this in a later step.

3 Select the **Actions ➤ New** menu. This displays the Templates screen where you define the properties of the template.

4 Select the Properties view and enter the name and description of the template file as well as the following information:

- **Location:** Specify where in the SA Library you want to store the template.
- **Version**: The version can be any string you use for tracking changes to the template. The version is not incremented automatically.
- **Type**: Specify whether it is a template file, a script or a localization file:
  - A template file is a model of configuration file.
  - Scripts are executed before or after pushing configuration files to the server. For more information, see Creating a Template from a Script on page 21.
  - A localization file is used for configuration files customized for different locales. For more information, see Localizing Configuration File Element Names on page 30.
- **Parser Syntax**: Select the type of syntax used by the template:
  - CML Syntax for all text configuration files other than XML files, and for all script files.
  - XML Syntax for configuration files written in XML.
  - XML DTD Syntax for configuration files written in XML that also use a DTD.
- **OS**: Specify which operating systems this template applies to. Only servers with the specified operating systems will be able to use the template. Only application configurations with at least one OS setting the same as this setting will be able to contain the template.
- **Availability**: Use this setting to keep track of which templates are tested and ready to be used and which are not yet tested or are deprecated. This setting does not change what can be done with the template. You can use this field value as a search criteria.
- **Auditable**: Set this when you want to enable auditing for this template file. For more information, see Auditing Application Configurations on page 57.

5 Select the Content view.

6 Enter your CML or XML or XML DTD text directly in the template editor. See Editing CML or XML in a Template below for the editing operations and syntax highlighting. For details on CML and XML, see the CML Reference on page 129 and Managing XML Configuration Files on page 61.

7 Select **Validate** to parse the CML or XML syntax and check for errors.

8 Select **File ➤ Save** to save your template.

9 Add your template to an application configuration object. See Adding or Removing Templates from an Application Configuration on page 19.

Editing CML or XML in a Template

You can edit the CML or XML of your template in the Content view. The template editor provides editing operations and syntax highlighting as described below.

1. From the SA Client navigation pane, select Library and then select the By Type tab.
2. Locate and open the Application Configuration node. Open the Templates node. Open the operating system group and select the operating system where the template file is located. Note that a template can apply to multiple operating systems.
3. Select a template.
4. Select the **Actions ➤ Open** menu or right-click and select the Open menu or press the Enter key. This displays the Templates screen showing the selected template.
5. Select the Content view. This displays the contents of your template.
6. Enter your CML or XML text directly in the template editor.

   The CML syntax is highlighted in colors to make reading it easier:

   — Green: CML instructions are displayed in green. CML key words are in bold green.

   — Blue: Variables that will be replaced by actual values from the value set are displayed in blue.

   — Black: Fixed text is displayed in black.

   — Gray: Comments are displayed in gray.

   The CML editor also provides the following editing operations:

   — Right-click in the CML text and use the cut, copy and paste operations.

   — The Actions menu provides find, replace, undo and redo operations.

   — The Validate button and the Actions ➤Validate menu check the syntax of your template and report any errors.

   For more information on CML and XML, see the CML Reference on page 129 and Managing XML Configuration Files on page 61.

Importing and Validating a Template File

You can write a CML template or an XML template with a text editor and import it into the SA Library for use in an Application Configuration. You can also validate the template from SA before importing it. See About Configuration Templates and Script Templates on page 34.

➤ For configuration files on Windows servers which are encoded in UTF-8, the first three characters of the configuration file might contain a Byte Order Mark (BOM). If you import this file into an Application Configuration Template, the BOM will appear in the template after the file is imported. If you do not want this BOM to be included in the Application Configuration Template, remove it after you upload the configuration file into the template.

UTF-16 encoding is not supported in the SA Client.

To validate and import a template file, perform the following steps:

1. From the SA Client select the Navigation pane, select Library then select the By Type tab.
2. Locate and open the Application Configuration node. Open the Templates node. Open the operating system group and select the operating system where the template file will be used. Note that a template can apply to multiple operating systems. You can specify this in a later step.
3. Select the **Actions ➤ Validate Template...** menu.
4. Locate and select your template file, select the appropriate encoding and select Open. SA checks the syntax of your template and reports the results. If there are errors in the file, correct them and revalidate.
5. Select the **Actions ➤ Import Template...** menu.
6. Locate and select your template file, select the appropriate encoding and select Open. Note that UTF-16 encoding is not supported in the SA Client. SA imports the template and displays the Template screen.
7. Follow the steps under Creating a Configuration Template starting from step 4 on page 17.

Viewing Configuration Template Sources

You can view the contents of your configuration template and view its CML or XML source. This can be helpful to understand which list merging modes have been set in the template before you push the Application Configuration to a server. For information on Application Configuration sequence merge modes, see Sequence Aggregation on page 164.

To view the source for a configuration template, perform the following steps:

1. From the SA Client navigation pane, select Library then select the By Type tab.
2. Locate and open the Application Configuration node. Open the Templates node. Open the operating system group and navigate to the operating system where the template file is. Note that a template can apply to multiple operating systems.
3. Select a configuration template and select **Actions ➤ Open**.
4. Select the Content view to display the CML or XML contents of the configuration template.

Adding or Removing Templates from an Application Configuration

An Application Configuration contains one or more templates. To add or remove a template from an Application Configuration, perform the following steps:

1. From the SA Client navigation pane, select Library and then select the By Type tab.
2. Locate and open the Application Configuration node. Open the Configurations node. Open the operating system group and navigate to the operating system where the Application Configuration is. Note that an Application Configuration can apply to multiple operating systems.

3 Select an Application Configuration and select **Actions ➤ Open**.

4 Select the Content view.

5 To add a template to the application configuration, select **Actions ➤ Add** or select the "+" button. Select the desired template and select OK.

At least on OS setting of the template must match an OS setting of the application configuration.

The customer setting of the folder containing the template must include the customer setting of the application configuration object. Otherwise the template will not be included in the list of available templates. For more information on folder settings, see "Folder Permissions" in the *SA Administration Guide*.

6 To remove a template, select the template and select **Actions ➤ Remove** or select the "-" button.

7 Select **File ➤ Save** to save your changes.

Specifying Template Order in the Application Configuration

An Application Configuration can contain one or more configuration templates and related scripts. You can specify the order of the templates in your application configurations. The templates will be pushed to managed servers in the order in which they appear in the application configuration. For example, you may need to apply changes to certain configuration files before others.

➤ The order of execution of scripts in an application configuration is determined by the script type: data-manipulation, pre-install, post-install or post-error. The order of the scripts in the application configuration is irrelevant. For more information, see Types of Application Configuration Scripts on page 49.

To specify the order of the templates in your Application Configuration, perform the following steps:

1 From the SA Client navigation pane, select Library and then select the By Type tab.

2 Locate the Application Configuration node and open it. Open the Configurations node. Open the operating system group and navigate to the operating system where the Application Configuration is. Note that an Application Configuration can apply to multiple operating systems.

3 Select an Application Configuration and select **Actions ➤ Open**.

4 Select the Content view. This displays all the configuration templates and scripts (if any) in the Application Configuration. Notice that the templates and scripts are numbered in order.

5 To reorder a template or script, select it then select **Actions ➤ Move Up** or **Actions ➤ Move Down** or select the up-arrow and down-arrow icons.

6 Select **File ➤ Save** to save your changes.

Creating a Template from a Script

To include a script in an application configuration object you must copy the script into a CML template then import the template into the application configuration object. For details on CML, see CML Reference on page 129. See also About Running Scripts with Application Configurations on page 49.

The following example shows a simple Unix shell script that performs a touch command and an echo command:

```
#/bin/sh
touch abc.txt
echo abc >>abc.txt
```

The following example shows this Unix shell script converted to CML:

```
@###########################################################
# /tmp/simple-script/TouchABC.sh                           #
# Version 0.1                                              #
# Author: <name>                                           #
###########################################################@
@!namespace=/simple-script-namespace/@
@!filename-key="/TouchABC"@
@!filename-default=/tmp/simple-script/TouchABC.sh@
#/bin/sh
touch abc.txt
echo abc >>abc.txt
```

To create a template from this script, perform the following steps.

1. Create a template as described in Creating a Configuration Template on page 16. Use the CML version of your script as the contents of the template.
2. Set the Type field to the appropriate script type. For the above example, you would set the Type to “Unix .SH script”. Other supported script types are listed in Table 4 on page 50.
3. Set the Parser Syntax field to “CML Syntax” since all scripts must be written in CML syntax.
4. Set the remaining fields as described in Creating a Configuration Template on page 16.
5. Select **File ➤ Save**.
6. Select **File ➤ Close**.
7. Add your template to an application configuration object as described in Adding or Removing Templates from an Application Configuration on page 19.
8. After adding your template to an application configuration object, open the application configuration object.
9. Select the Configuration Values view.
10. Select the script template and right-click to display the menu.
11. Select the script type, which specifies when the script will run: Data-manipulation, Pre-install, Post-install or Post-error. These types are described in Table 3 on page 49.
12. Select **File ➤ Save**.
13. Select **File ➤ Close**.

Managing Non-Text Configurations by Running a Data Manipulation Script

With SA you can manage non-text configurations by creating a data-manipulation script that extracts the non-text configuration data and places it in a text file. The text file can then be manipulated by SA the same as any other text configuration file and placed back into its original form. For a description of script types, see Types of Application Configuration Scripts on page 49.

Some examples of non-text configuration data are:

- An SQL database with a data manipulation script could run some SQL queries and place the data in a text file.
- An IIS server's configuration with a data-manipulation script that reads the metabase information into a text file.
- A binary file with a data manipulation script that extracts the values from the binary file and places them into a text file.

Running Data Manipulation Scripts Manually

The Run Script button allows you to execute a data manipulation script associated with an application configuration and to prepare a target configuration file on a managed server so its values can be imported into a value set.

To run a data manipulation script manually, perform the following steps:

1. From the SA Client navigation pane, select the Devices tab.
2. Select either All Managed Servers or Device Groups. Navigate to the desired server or device group.
3. Select a server or device group and select the **Actions ➤ Open** menu.
4. If you selected a server, perform the following steps. If you selected a device group, skip to the next step.
   a. Select the Management Policies tab.
   b. Open the Configured Applications node in the navigation pane. This displays all the application configurations attached to the server.
   c. Open the application configuration node you want to push. This displays the value sets for the application configuration.
   d. Select the value set you want to push. You must select a value set at the server instance level.
5. If you selected a device group, perform the following steps.
   a. Open the Configured Applications node in the navigation pane. This displays all the application configurations attached to the servers and a list of all the servers.
   b. Open the Servers node. This displays all the servers in the device group.
   c. Open the particular server node where you want to run the data manipulation script. This displays the application configuration attached to that server.

d Open the application configuration node you want to push. This displays the value sets for the application configuration.

e Select the value set you want to push. You must select a value set at the server instance level.

6 With a value set at the server instance level selected, select the Run Script button. This displays the confirmation dialog.

7 Select Yes to run the data-manipulation script on the server.

8 Once the data-manipulation script has run, extracted the configuration data and placed it into a text file, you can manage the text file the same as any other configuration file and place the data from the text file back into its original form.

Attaching an Application Configuration to a Server or Device Group

After you have created an Application Configuration, added all the necessary configuration templates and scripts and edited its default values, you must attach it to a server or public device group. Once you attach an Application Configuration to a server or group of servers, you must push the application configuration to the servers as described in Pushing Application Configurations on page 26.

➤ The customer setting of the folder containing the application configuration must include the customer setting of the managed servers where you intend to push the application configuration. For more information on folder settings, see “Folder Permissions” in the *SA Administration Guide*. For more information about servers and customers, “Customer Accounts” in the *SA Users Guide: Server Automation*.

➤ You can only attach Application Configurations to individual servers or to public device groups. You cannot attach them to private device groups.

Attaching an Application Configuration to a Single Server

To attach an Application Configuration to a single server, perform the following steps:

1 From the SA Client navigation pane, select the **Devices** tab.

2 Select **Servers ➤ All Managed Servers**.

3 From the Content pane, select a server.

4 Select the **Actions ➤ Open** menu.

5 Select the **Management Policies** tab, then select **Configured Applications**.

6 Select the Installed Configurations tab.

7 Select the **Actions ➤ Add Configuration...** menu.

8 In the Select Application Configuration screen, select the Application Configuration that you want to attach to the managed server.

You can use the search tool 🔍 to search by a specific criteria such as name, date last modified and so on.

Note that the customer setting of the folder containing the application configuration must include the customer setting of the managed servers where you intend to push the application configuration. For more information on folder settings, see "Folder Permissions" in the *SA Administration Guide*. For more information about servers and customers, "Customer Accounts" in the *SA Users Guide: Server Automation*.

9 Select **OK** to attach the Application Configuration to the server.

10 Select the Save Changes button.

11 You can now set the Application Configuration's values for that server. For more information on setting up the Application Configuration values, see About Value Sets on page 36.

Attaching an Application Configuration to a Device Group

To attach an Application Configuration to a device group, perform the following steps.

➤ You can only attach Application Configurations to public device groups. You cannot attach Application Configurations to private device groups.

1 From the SA Client navigation pane, select the Devices tab.

2 Open the Device Groups node and navigate to a public device group.

3 From the Content pane, select the desired device group.

4 Select the **Actions ➤ Open** menu.

5 In the device group screen, select the Configured Applications view.

6 Select the **Actions ➤ Add Configuration** menu.

7 In the Select Application Configuration dialog box, select an Application Configuration.

Use the search tool 🔍 to search by a specific criteria such as name, date last modified and so on.

8 Enter an Instance Name. This is the name of the value set at the group instance level. For more information, see Setting Values at the Group Instance Level on page 46.

9 Select OK to attach the Application Configuration to the device group. The specified configuration file can be pushed to all servers in the group.

10 Set the values to be pushed to the servers. For more information on setting values, see About Value Sets on page 36.

Detaching an Application Configuration from a Server or Device Group

To detach an application configuration from a server you must open the server and remove all the instances of the application configuration at the server instance level. To detach an application configuration from a device group you must open the device group and remove all the instances at the group instance level. The following sections provide details.

Detaching an Application Configuration from a Server

To detach an application configuration from a server, you must remove all the instances at the server instance level as described below. For more information, see Value Set Editor at the Server Level on page 46. Perform the following steps.

1 Open the server in the SA Client. The application configuration must be attached to the server. See Attaching an Application Configuration to a Server or Device Group on page 23.

2 Select the Management Policies tab on the left.

3 Open the Configured Applications node in the Management Policies pane on the left.

4 Open the desired application configuration node under the Configured Applications node. This displays all the instances of the application configuration at the server instance level.

5 Under the application configuration node, select one of the instances.

6 Select the **Actions ➤ Remove Configuration** menu or right-click and select the **Remove Configuration** menu. This removes the selected instance.

7 Repeat step 5 and step 6 for each instance. When you remove the last instance, the application configuration is detached from the server.

8 Select the Save Changes button.

Detaching an Application Configuration from a Device Group

To detach an application configuration from a device group, you must remove all the instances of the application configuration at the group instance level as described below. For more information, see Setting Values at the Group Level on page 45. Perform the following steps.

1 From the SA Client navigation pane, select the Devices tab.

2 Open the Device Groups node and navigate to a public device group.

3 From the Content pane, select the desired device group.

4 Select the **Actions ➤ Open** menu.

5 In the device group screen, select the Configured Applications view and open the Configured Application node. This displays the application configurations attached to the device group.

6 Open the desired application configuration node under the Configured Applications node. This displays the value sets at the group instance level.

7 Under the desired application configuration node, select one of the application configuration instances at the group instance level.

8 Select the **Actions ➤ Remove Configuration** menu or right-click and select the **Remove Configuration** menu. This removes the selected instance.

9 Repeat step 7 and step 8 above for each instance at the server instance level. When you remove the last instance, the application configuration is detached from the server.

10 Select the Save Changes button.

Pushing Application Configurations

Anytime you change values in a value set, to merge those changes with the configuration file on the target server you must push the application configuration to the server. For more information, see About Pushing Application Configurations to Servers on page 51.

To push application configuration changes to a server or group of servers, perform the following steps:

1 From the SA Client navigation pane, select the Devices tab.

2 Select either All Managed Servers or Device Groups. Navigate to the desired server or device group.

3 Select a server or device group and select the **Actions ➤ Open** menu.

4 If you selected a server, select the Management Policies tab. If you selected a device group, skip to the next step.

5 Open the Configured Applications node in the navigation pane and select the value set you want to push. You must select a value set at the server instance level or a value set at the group instance level.

6 You can optionally preview the changes that will be made on an individual server by selecting the **Preview** button. The Comparison screen shows any differences. Select **Close** when you are finished.

7 When you are ready to apply the changes to the server or servers, select **Push**.

8 In the Push Configurations screen, verify the Application Configuration and the value set to be pushed.

9 Select Start Job to accept the remaining defaults for Scheduling, Notifications and Job Status. Otherwise select Next.

10 In the Scheduling pane, specify when you want the Application Configuration to be pushed. You can use the Scheduling pane to schedule the job to run in the future or to run at regular recurring intervals such as weekly or monthly.

11 Select Start Job to accept the remaining defaults for Notifications and Job Status. Otherwise select Next.

12 In the Notifications pane, optionally specify one or more email addresses and a ticket identifier.

13 Select Start Job to accept the remaining defaults for Job Status. Otherwise select Next.

14 Select Start Job.

15 After the job has started, you can view its status by selecting the Jobs and Sessions tab then Job Logs on the main SA Client screen.

Modifying Push Timeout Values

By default, when you push an Application Configuration, the default timeout value is ten minutes, plus one minute for each template inside the Application Configuration. Each template in that Application Configuration appends its timeout to the base timeout for the Application Configuration.

For example, if you have an Application Configuration that contains three templates, the default timeout value for the entire Application Configuration is 13 minutes. If you pushed the template and the entire push took longer than 13 minutes, the push will time out and the operation is cancelled, including any changes that were already made.

To extend a template's timeout value, you can use the CML timeout tag for individual templates inside the Application Configuration. The CML timeout tag syntax is as follows:

```
@!timeout=1@
```

Valid values are 0-999, in minutes.

If the Application Configuration times out in the middle of a push, all changes to the target file of the push are backed out and the operation is cancelled.

For details on the CML timeout tag, see the CML Reference on page 129.

Scheduling an Application Configuration Push

You can schedule an Application Configuration push to run immediately, in the future or on a recurring schedule, such as daily, weekly, or monthly. To schedule an Application Configuration push, follow the steps listed under Pushing Application Configurations on page 26 but when you get to the Scheduling step, enter the frequency and time when you want the push to occur.

Once you have scheduled the job, you can view its status by selecting the Jobs and Sessions tab the Recurring Schedules on the main SA Client screen.

Restoring a Configuration File to a Previous State

Every time you push an Application Configuration to a server, the configuration files are saved in a configuration push history list. At any time, you can restore an Application Configuration to a previous state in the history list.

To restore an Application Configuration to a previous state, perform the following steps:

1. From the SA Client navigation pane, select the Devices tab.
2. Select All Managed Servers and locate the desired server.
3. Select a server and select the **Actions ➤ Open** menu.
4. Select the Management Policies tab.
5. Select the Configured Applications node in the navigation pane. This displays all the application configurations installed on the server.
6. Select the Configuration History tab. This displays all the push jobs that have run on the server.

7. Select the line in the history that you want to restore.
8. Select the Restore button. This displays the Restore Configurations wizard.
9. Verify the server and history instance to restore and select the Next button.
10. Select the restore type:
    - "Previous to selected push (undo)" restores the configuration file to the values immediately before the selected history instance. That is, it undoes the selected instance.
    - "Following selected push (redo)" restores the configuration file to the values of the selected history instance. That is, it repushes the selected instance.
11. Select the Next button.
12. Select the Start Job button. This restores the configurations on the server to the selected history instance.
13. After the job has started, you can view its status by selecting the Jobs and Sessions tab then Job Logs on the main SA Client screen.

Searching and Filtering Job Results

You can search and filter the results of a push or restore job. This is useful when your job runs on a large number of servers. Note that you can search and filter any kinds of SA jobs, not just push and restore jobs. Perform the following steps.

1. From the SA Client navigation pane, select the **Jobs and Sessions** tab.
2. Select **Job Log**. This displays a list of jobs that have run on your servers.
3. Select a job from the job list.
4. Select the Actions ➤ Open menu. This displays the details of the selected job.
5. In the job screen, select Job Status. This displays the results of the job.
6. Select any step in the Actions column.
7. On your keyboard, type Ctrl-F. This displays the Find tool for the steps in the job. Enter the text you want to search for in the text box and use the buttons to search and highlight.
8. Select the detailed text in the lower box.
9. On your keyboard, type Ctrl-F. This displays the Find tool for the details of the job results. Enter the text you want to search for in the text box and use the buttons to search and highlight.
10. To remove the Find tool, select the text box and type Esc on your keyboard.

Comparing a Configuration Template with a Target Configuration File - Preview

Before you push an application configuration to a server, you can compare the proposed application configuration with the target configuration file on the server with the Preview button. You can select a server from your managed servers or you can select a server in a device group.

Select a Server from Your Managed Servers

To compare the values in a configuration template with an actual configuration file on a server, perform the following steps.

1. From the SA Client navigation pane, select the **Devices** tab.
2. Select **Servers ➤ All Managed Servers**.
3. From the Content pane, select a server.
4. Select the **Actions ➤ Open** menu.
5. Select the **Management Policies** tab.
6. Open the **Configured Applications** node to display all the application configurations attached to the server.
7. Open the desired application configuration node to display all the instances of that application configuration.
8. Select an instance of the application configuration.
9. In the content pane, select a template from the drop-down list.
10. Select the Preview button. This compares the generated configuration file generated from the template and value set with the actual configuration file on the server and displays both files side by side with color coding to make it easier to interpret.
    - **Green**: This indicates that new information has been added.
    - **Blue**: This indicates that information has been modified.
    - **Red**: This indicates that information has been deleted.
    - **Black**: This indicates no changes.

Select a Server from A Device Group

To compare the values in a configuration template with an actual configuration file on a server in a device group, perform the following steps.

1. From the SA Client navigation pane, select the **Devices** tab.
2. Select **Device Groups ➤ Public** to display your public device groups.
3. Navigate to the desired device group and from the Content pane, select a public device group.
4. Select the **Actions ➤ Open** menu.

5. Open the **Configured Applications** node to display all the application configurations attached to the device group and the servers in the device group.
6. Open the Servers node to display all the servers in the device group.
7. Open the desired server mode to display the application configurations attached to that server.
8. Open the desired application configuration node to display all the instances of the application configuration.
9. Select an instance of the application configuration.
10. In the content pane, select a template from the drop-down list.
11. Select the Preview button. This compares the generated configuration file generated from the template and value set with the actual configuration file on the server and displays both files side by side with color coding to make it easier to interpret.
    - **Green**: This indicates that new information has been added.
    - **Blue**: This indicates that information has been modified.
    - **Red**: This indicates that information has been deleted.
    - **Black**: This indicates no changes.

Comparing Two Configuration Templates

To compare two configuration templates, perform the following steps:

1. From the SA Client navigation pane, select Library then select the By Type tab.
2. Locate and open the Application Configuration node. Open the Templates node. Open the operating system group and navigate to the operating system where the template files are. Note that a template can apply to multiple operating systems.
3. Select a configuration template.
4. Hold down the Ctrl key on the keyboard and select the second template.
5. Right click and select the Compare menu. The Comparison screen displays the differences between the two files. Use the arrows in the upper right of the screen to navigate through the two files. The following colors indicate the differences:
    - Blue indicates information that is different between the two templates.
    - Red indicates information that has been removed.
    - Green indicates information that has been added.
    - Black indicates identical text.
6. When you are finished viewing the differences, select **Close**.

Localizing Configuration File Element Names

You can display the names of configuration file elements in your local language in the value set editor of the SA Client. This can be useful when your system administrators who will be specifying value sets use a different language.

Localization files represent locale-specific resource files, where you define localized strings for configuration template elements. These localized strings display in the value set editor.

Localization templates are used only in the value set editor in the SA Client and are ignored during a push.

Creating a Localization File

To localize the names of configuration file elements in the SA Client, you must first create a localization template in the SA Library. Perform the following steps.

1. From the SA Client navigation pane, select Library and then select the By Type tab.
2. Locate and open the Application Configuration node. Open the Templates node. Open the operating system group and select the operating system where the template file will be used. Note that a template can apply to multiple operating systems. You can specify this in a later step.
3. Select the **Actions ➤ New** menu. This displays the Templates screen where you define the properties of the template.
4. Select the Properties view and enter a description of the localization template file as well as the following information:
   - **Name**: Specify the name of the localization file. The naming convention for localization template files requires that localization files end with .`<locale>`. The values for `<locale>` are defined by ISO-639 and are two-letter lower case codes for representing the names of languages. For more information on ISO 639, see http://www.loc.gov/standards/iso639-2/php/code_list.php

     For example, .es represents Spanish, .en represents English, .fr represents French, .zh represents Chinese, and .hi represents Hindi.
   - **Location:** Specify where in the SA Library you want to store the localization template.
   - **Version**: The version can be any string you use for tracking changes to the template. The version is not incremented automatically.
   - **Type**: Specify Localization file as the type.
   - **Availability**: Use this setting to keep track of which localization files are tested and ready to be used and which are not yet tested or are deprecated. This setting does not change what can be done with the localization file. You can use this field value as a search criteria.
5. Select the Content view.
6. Enter your localization instructions directly in the template editor. The format for localization instructions is:

   ```
   /printable/<name space>/<variable> <localized string>
   ```

   Where `printable` is a key word indicating that the line is a localization instruction.

   `<name space>` is the name space where the desired variable is stored in the database.

   `<variable>` is the variable defined in a configuration template.

   `<localized string>` is the text that will be displayed in the value set editor instead of the full name space and variable name.

   See Editing CML or XML in a Template on page 18 for the editing operations and syntax highlighting.

7. Select **Validate** to parse the syntax and check for errors.
8. Select **File ➤ Save** to save your template.
9. Add your localization template to an application configuration object as described in Applying a Localization Template below.

Applying a Localization Template

After creating a localization template, you apply it to an application configuration so the configuration file elements display in your local language. Perform the following steps.

1. Open your application configuration object as follows.
   a. From the SA Client navigation pane, select Library and then select the By Type tab.
   b. Locate and open the Application Configuration node. Open the Configurations node. Open the operating system group and navigate to the operating system where the Application Configuration is. Note that an Application Configuration can apply to multiple operating systems.
   c. Select an Application Configuration and select **Actions ➤ Open**.
2. Select the Properties view.
3. Under the Localization Files in the content pane, select the “+” button or select **Actions ➤ Add.**
4. In the Select Localization Template screen, select a localization template.
5. Select OK.
6. Select **File ➤ Save**. This applies your localization template and displays the configuration file elements in the language specified by the localization template. Whenever anyone displays the value set editor for the application configuration, the localized strings will be displayed rather than the raw name space and variable names.

3 Application Configuration Concepts

With HP Server Automation you can manage configuration files, including XML configuration files, from a central location and easily propagate changes and updates across multiple servers in your data center. You can manage a single configuration file such as the /etc/hosts file on UNIX systems, or multiple complex configuration files associated with an application, such as the configuration files associated with a large business application such as WebLogic or Websphere.

Configuration files are any files on your servers that you need to control the contents of. This includes configuration files for applications as well as system configuration files, such as:

- Files that control the behavior of application servers, web servers, databases, or other applications. For example, you can manage the httpd.conf file for the Apache Web Server running across several different servers. You can attach an application configuration for this file and enter specific values for each instance of that file on each server.
- Files on your servers such as the /etc/hosts, /etc/fstab, /etc/passwd, or /etc/groups files on UNIX servers.
- Any other files on your servers that are used to configure your servers.

In addition, you can do the following with application configurations:

- **Run scripts** before or after the configuration file is placed on the server. For example, you could use a script that restarts the application after pushing the configurations to the server. For details, see About Running Scripts with Application Configurations on page 49.
- Use application configurations in a **software policy** as part of deployment and ongoing management of applications. For details, see Using Application Configurations in Software Policies on page 57.
- **Audit** to determine if servers conform to the desired application configuration file contents. For details, see Auditing Application Configurations on page 57.
- **Check compliance** of your servers against the application configuration you have defined. See Application Configuration Compliance on page 51.
- **Restore** a previous configuration file. For details, see Restoring a Configuration File to a Previous State on page 27.

About Application Configuration Objects

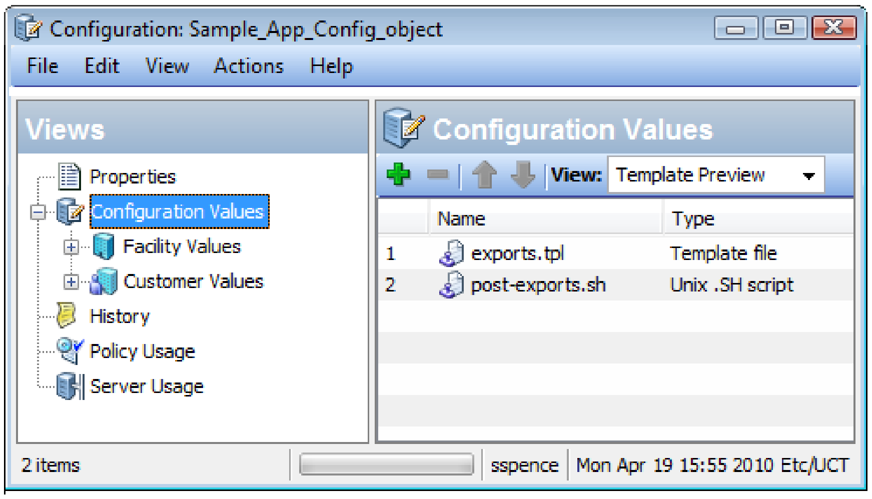
To manage a configuration file on a server, you need to create an application configuration object and at least one configuration template in the SA Library. The application configuration object is just a container for templates and optional scripts.

When you push an application configuration to one or more servers, SA does the following:

- Generates configuration files from the configuration templates and the specified value sets.
- Copies the generated configuration files to the servers.

Figure 1 below shows an example application configuration that contains a template named “exports.tpl” and a Unix shell script file named “post-exports.sh”. The “.tpl” file extension indicates these are templates.

Figure 1 Application Configuration Containing a Configuration Template and a Shell Script



	Name	Type
1	exports.tpl	Template file
2	post-exports.sh	Unix .SH script

2 items | sspence | Mon Apr 19 15:55 2010 Etc/UCT

About Configuration Templates and Script Templates

A **configuration template** is a model of a configuration file with variables in place of the data values that vary from server to server. Configuration templates also contain instructions about how to recreate the configuration file from the template and from a value set during a push operation. The configuration template defines the fixed parts of the configuration file and the variable parts. The template also contains instructions to parse the configuration file and to recreate the configuration file to push it to managed servers.

You create configuration template files using the SA Configuration Modeling Language (CML) or in XML. Use XML for all XML-based configuration files and CML for all other configuration files. For complete information on CML, see CML Reference on page 129. For information on XML configuration files, see Managing XML Configuration Files on page 61.

You define the values that will be used to generate the final configuration file. These values are stored as **value sets** in the SA database. The CML or XML in the configuration template merges the values in a value set with the template file to generate the target configuration file. For more information, see About Value Sets on page 36.

You can also use the configuration template to import data from a configuration file and build a value set. For more information, see Importing and Validating a Template File on page 18.

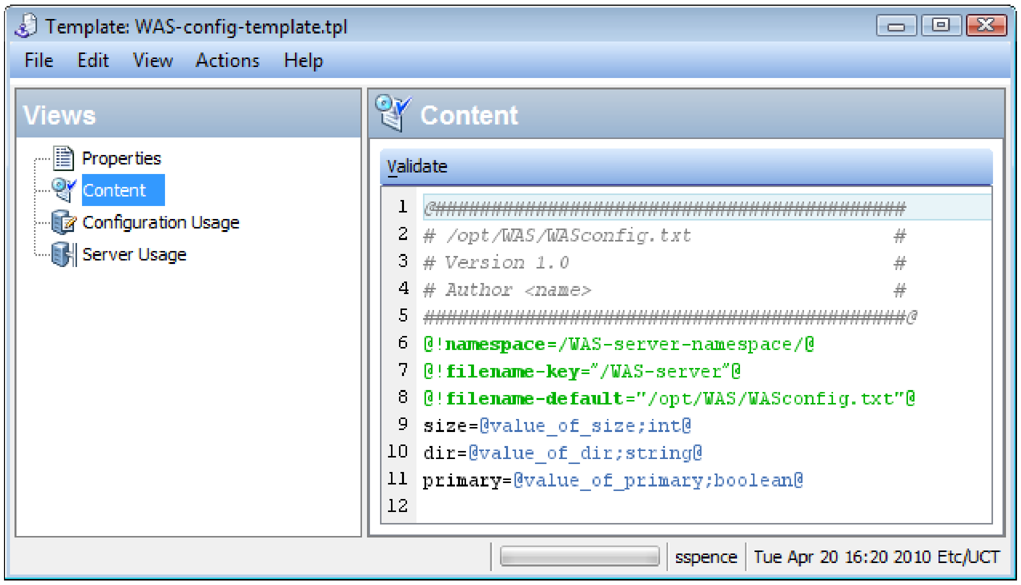
To manage configuration files, create a configuration template for each file you want to manage and add it to an Application Configuration. For more information, see Creating an Application Configuration on page 15.

CML Configuration Templates

The Configuration Modeling Language (CML) provides a way to create a template of a configuration file that is merged with a value set to generate an actual configuration file. Figure 2 below shows a sample configuration template file in Configuration Modeling Language (CML).

For more information, see CML Tutorial 1 - Create an Application Configuration for a Simple Web App Server on page 85, CML Tutorial 2 - Create a Template of a Web Server Configuration File on page 95 and CML Reference on page 129.

Figure 2 Example Configuration Template Written in CML



```
@##########################################
# /opt/WAS/WASconfig.txt                  #
# Version 1.0                             #
# Author <name>                           #
###########################################@
@!namespace=/WAS-server-namespace/@
@!filename-key="/WAS-server"@
@!filename-default="/opt/WAS/WASconfig.txt"@
size=@value_of_size;int@
dir=@value_of_dir;string@
primary=@value_of_primary;boolean@

```

XML and XML-DTD Configuration Templates

You can also manage XML configuration files on your managed servers. Using XML configuration templates, you can model XML configuration file values, check those values against actual XML on target servers, and push changes to the target files. You can model XML configuration files that use a DTD as well as XML configuration files that do not.

An XML configuration template functions much like a CML template, but uses some application configuration options defined in the comments. With XML configuration templates, you can also use tags to customize the way the file is displayed in the SA Client.

For more information, see Managing XML Configuration Files on page 61.

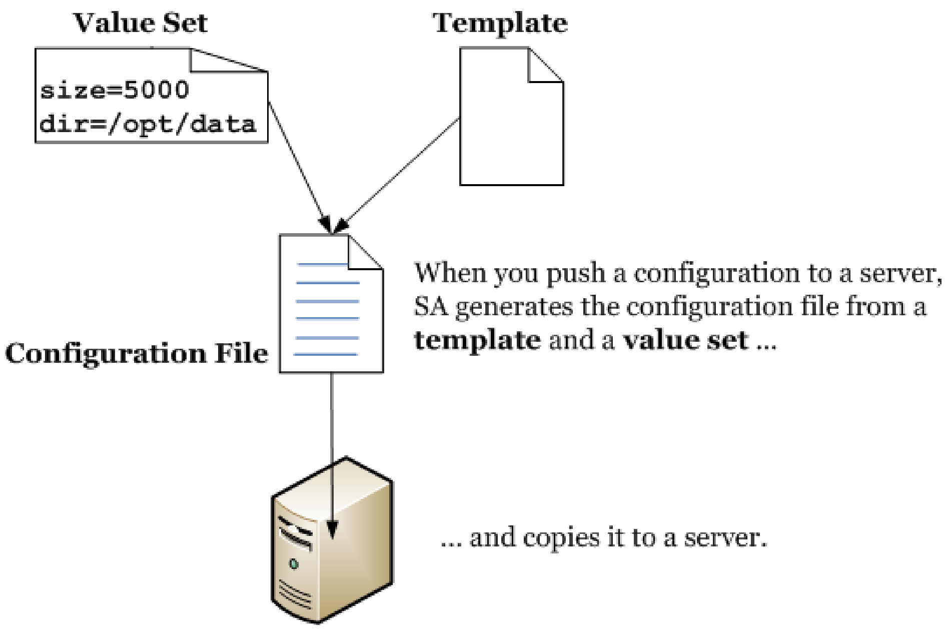
Script Templates

You can add scripts to an Application Configuration that are executed before or after the configuration values are copied to the target server. To include a script in an application configuration object you must copy the script into a CML template then import the script template into the application configuration object.

For more information, see About Running Scripts with Application Configurations on page 49, Creating a Template from a Script on page 21.

About Value Sets

A value set is a set of data values that are merged with a template file to generate a target configuration file. The resulting configuration file can be pushed to servers.


Value Set
size=5000
dir=/opt/data
Template
Configuration File
When you push a configuration to a server, SA generates the configuration file from a **template** and a **value set** ...
... and copies it to a server.

At its simplest, you define a value set at the "Server Instance" level for a particular server. The values in the "Server Instance" value set are merged with the configuration template to generate the actual configuration file that will be pushed to the server.

Value Set Levels and Value Set Inheritance

Setting values for each individual server at the server instance level works for a small number of servers but for a large number of servers you can use value set inheritance to set default values at higher levels that apply to larger and larger subsets of your managed servers and are inherited by lower levels. Each level inherits the values from the levels above, unless the level explicitly blocks inheritance or sets a value that overrides the inherited value.

Table 1 below lists all the value set inheritance levels, how they are either inherited or overridden at lower levels and how to set values at each level. Details on setting values at each level are described below.

Table 1 Value Set Levels and Inheritance of Value Sets

Level	**Description of the Level**	**How to Set a Value**
Application	This level defines values that apply to the application configuration itself. It applies to all servers that have the application configuration attached, unless overridden by any level below.	Open the App Config object. Select Content ➤ Application Values. Select a template. Select the "File Values" view. See Setting Values at the Application Level on page 43.
Facility	This level defines values for a facility. It applies to all servers in the specified facility that have the application configuration attached, unless overridden by any level below.	Open the App Config object. Select Content ➤ Facility Values ➤ <*facility*>. Select a template. Select the "File Values" view. See Setting Values at the Facility Level on page 44.
Customer	This level defines values for a customer. It applies to all servers belonging to the specified customer that have the application configuration attached, unless overridden by any level below.	Open the App Config object. Select Content ➤ Customer Values ➤ <*customer*>. Select a template. Select the "File Values" view. See Setting Values at the Customer Level on page 44.
Group	This level defines values for a device group. It applies to all servers in the specified device group that have the application configuration attached, unless overridden by any level below.	Open the device group. Select Configured Applications ➤ <*app config name*>. Select a template. See Setting Values at the Group Level on page 45.

Table 1 Value Set Levels and Inheritance of Value Sets

Level	Description of the Level	How to Set a Value
Group Instance	This level defines values for one particular instance of an application configuration. It applies to all servers in the specified device group that the instance is attached to, unless overridden by any level below.	Open the device group. Select Configured Applications ➤ *<app config name>* ➤ *<instance name>*. Select a template. See Setting Values at the Group Instance Level on page 46.
Server	This level defines values for a server. It applies to all instances of the application configuration on the specified server, unless overridden by the level below.	Open the server. Select Management Policies tab. Select Configured Applications ➤ *<app config name>*. Select a template. See Setting Values at the Server Level on page 47.
Server Instance	This level defines values for one particular instance of an application configuration on the server. It applies only to the specified application configuration instance on the specified server and overrides all levels above.	Open the server. Select Management Policies tab. Select Configured Applications ➤ *<app config name>* ➤ *<instance name>*. Select a template. See Setting Values at the Server Instance Level on page 48.

Table 2 below illustrates how application configuration values are inherited. The values in each row represent values set at that level. The bottom row shows the actual values that will be pushed to a server.

Table 2 Inheritance of Application Configuration Values

Level	**Values Set at Each Level**						
Application	2	1	Z	Y	X	W	V
Facility		U	T	S	R	Q	P
Customer			O	N	M	L	K
Group				J	I	H	G
Group Instance					F	E	D
Server						C	B
Server Instance							A
Inherited Results	2	U	O	J	F	C	A

Blocking Inheritance

You can block inheritance at any level by selecting Block Inheritance in the value set editor for any variable. All values set above the blocked level will not be inherited. Values at levels below the blocked level are still inherited. A lower level must set the value or the variable will have no value.

1 Open the value set editor at any level.
2 In the value column for any variable, select a value. This displays a drop-down list and a "..." button on the right end of the edit box.
3 Select either the drop-down list or select the "..." button. This displays several choices for values of the variable.
4 Select Block Inheritance.
5 Select File ➤ Save or the Save Changes button.
6 For more information, see Setting Values in the Value Set Editor below.

About the Value Set Editor

The value set editor displays the variables defined in the template and lets you set the values of the variables. The value set editor is displayed when you select a value set of the application configuration at any of the inheritance levels. This section describes how to use the value set editor.

For examples of the value set editor at each inheritance level, see Value Set Editor at the Application, Facility and Customer Levels on page 42, Value Set Editor at the Group Level on page 44 and Value Set Editor at the Server Level on page 46.

Setting Values in the Value Set Editor

You can set any value for a variable in the value set editor as long as it conforms to the data type of the variable. Type the value in the Value column next to the desired variable.

In addition, you can set values in any of the following ways:

- Select the drop-down list on the right end of the edit box and select one of the following values:
  - Empty String - This specifies that a zero-length string is placed in the value set.
  - Block Inheritance - This specifies that values at a higher level will not be inherited. A lower level must set the value or the variable will have no value.
  - Agent version, Auth domain, Chassis Id, Customer Id, Customer Name and so forth - This specifies that the selected information about the managed server will be placed in the value set.
- Select the "..." button to the right of the edit box to display the following choices:
  - No value - Leaves the value empty in the value set.
  - Block Inheritance - This specifies that values at a higher level will not be inherited. A lower level must set the value or the variable will have no value.
  - Any Value - Enter any value.

- — Object Attribute - Select one of the values from the list above.
- — Custom Attribute - Enter a custom attribute. For more information on custom attributes, see the *SA User Guide: Application Automation*.
- — Deployment Automation Value - Enter a value from an application managed by Deployment Automation. For more information, see the *Deployment Automation User Guide*.

- Right click in the edit box to display the edit menu.

Setting Fields in the Value Set Editor

At whichever level you are setting values, the value set editor displays the fields described below for the value set being edited.

Figure 3 below shows the server RHEL004 and, in particular, the application configuration named "WAS-app-config" attached to this server. The value set at the server instance level is displayed and can be edited. See this diagram for an example of the following fields.

- **Template**: Lists the template files contained in the application configuration object and lets you select the template to view and modify. Select the template you want to view.
- **Filename**: Specifies the name of the configuration file on the managed server that is the target of the configuration template. If no file name is set, then the file name is inherited. If no file name is set anywhere in the application configuration hierarchy, then the file name listed in the configuration template is used.

  If you have multiple instances of an application on a server, use this field to indicate the full path name for each target configuration file.
- **Encoding**: Specifies the character encoding for the target configuration file. The default is the encoding used on the managed server. (Note that UTF-16 encoding is not supported in the SA Client.)
- **Preserve Format**: Specifies whether to preserve spacing, comments and ordering of the target configuration file. SA will attempt to preserve as much of the target configuration file as possible, but may not be able to preserve all comments and formatting. This option is required if your template uses the `@!partial-template@` CML tag.

  If this is not turned on for the template, all comments and formatting will be removed from the file and the default ordering and spacing from the template will be used.

  Note that for XML-based templates, preserve format will not preserve white space or attribute ordering within an XML tag. Preserve format will preserve white space and ordering for everything except the white space in the tags themselves and the ordering of the attributes in those tags. After a push, extra white space inside the tags disappear and the ordering of the attributes may change. Eliminating white space and changing the attribute order has no effect on the meaning of the XML.
- **Preserve Values**: Specifies whether to preserve the values contained in the target configuration file on the managed server if there is no corresponding value in the value set. By default, this option is turned off.

  Preserve Values lets you request that any values in the configuration file on the server that are not also stored in the SA database are to be preserved. This is useful if you do not intend to import all the current values from the configuration file into SA, or if your users or programs sometimes modify configuration files outside of SA. This allows you to manage some configuration file changes outside of SA.

- **Show Inherited Values**: Displays the resulting value set including values inherited from higher levels. When turned off, displays only the values set at the current level. This view is read-only and is available only when viewing a value set at the server instance level.

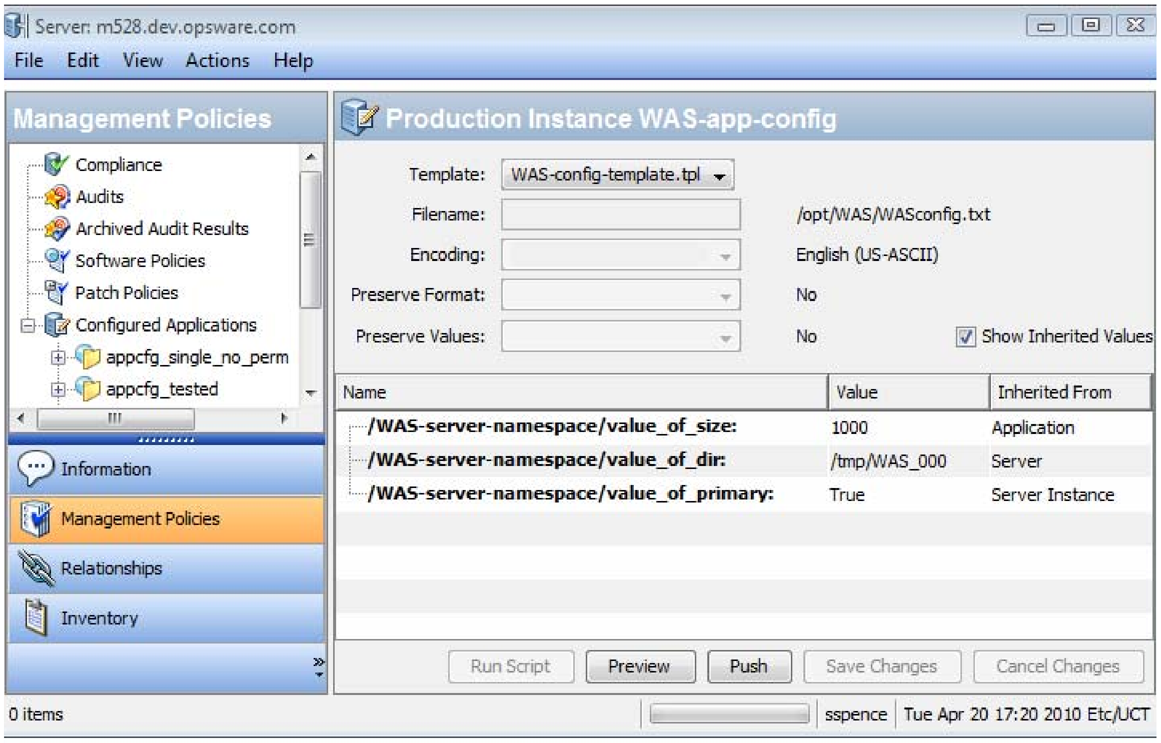
Meaning of Columns in the Value Set Editor

When you are editing value sets, the SA Client displays the following information about the configuration template values. Figure 3 below shows the server M536 and, in particular, an instance of the application configuration named “WAS-app-config” attached to this server. The value set at the server instance level is displayed and can be edited. The columns Name, Value and Inherited From are described below.

- **Name**: Lists the names of the variables in your template file. The name can be a simple type, a list of simple types, or a multidimensional list. Multidimensional lists are displayed beneath their parent. Elements that are required are in bold. You can double-click to show or hide multidimensional lists. To add another entry to a list type value, right-click the parent and choose **Add Item**. Required fields are set in the configuration template. Required fields cannot be empty or you will not be able to preview or push the application configuration.
- **Value**: Lists the values in value set being displayed. You can either enter a literal value or choose an attribute from the Server’s settings, such as customer name, customer ID, chassis ID, device ID, and so on. If you leave a setting blank, then the setting is inherited from its parent or ancestor (if a parent or ancestor has values configured). To set the value to a custom attribute for the value, click the “...” button to use the Set Value dialog box.
- **Inherited From**: Indicates where the value is inherited from. This column is only displayed when viewing at the server instance level and when you have Show Inherited Values selected.

  If the Preserve Values option is set, the configuration file on the server becomes the outermost level of the inheritance hierarchy. That is, if no value exists in the value set, the value in the file will be preserved.

Figure 3 Value Set Editor at the Server Instance Level



Template: WAS-config-template.tpl
Filename: /opt/WAS/WASconfig.txt
Encoding: English (US-ASCII)
Preserve Format: No
Preserve Values: No
[✓] Show Inherited Values

Name	Value	Inherited From
/WAS-server-namespace/value_of_size:	1000	Application
/WAS-server-namespace/value_of_dir:	/tmp/WAS_000	Server
/WAS-server-namespace/value_of_primary:	True	Server Instance

Importing Value Sets from an Existing Configuration File

While can set values in a value set manually, you can also import values into a value set from an existing configuration file on a managed server. Perform the following steps.

1 Display the value set editor at the level where you want to import the values. See the sections below for how to display the value set editor at each level.

2 In the value set editor, make sure the Filename field shows the absolute file name of the configuration file that contains the values that you want to import.

3 Right-click in the Value column and choose **Import Values**. All of the values for the configuration template are replaced with the values from the actual configuration file.

Selecting the **Import Values** menu item reads the existing configuration file on the managed server, parses the values, and saves them at the selected level. After you import the values, you can modify any of them and push the changes back to the server, if desired

4 Select the Save Changes button.

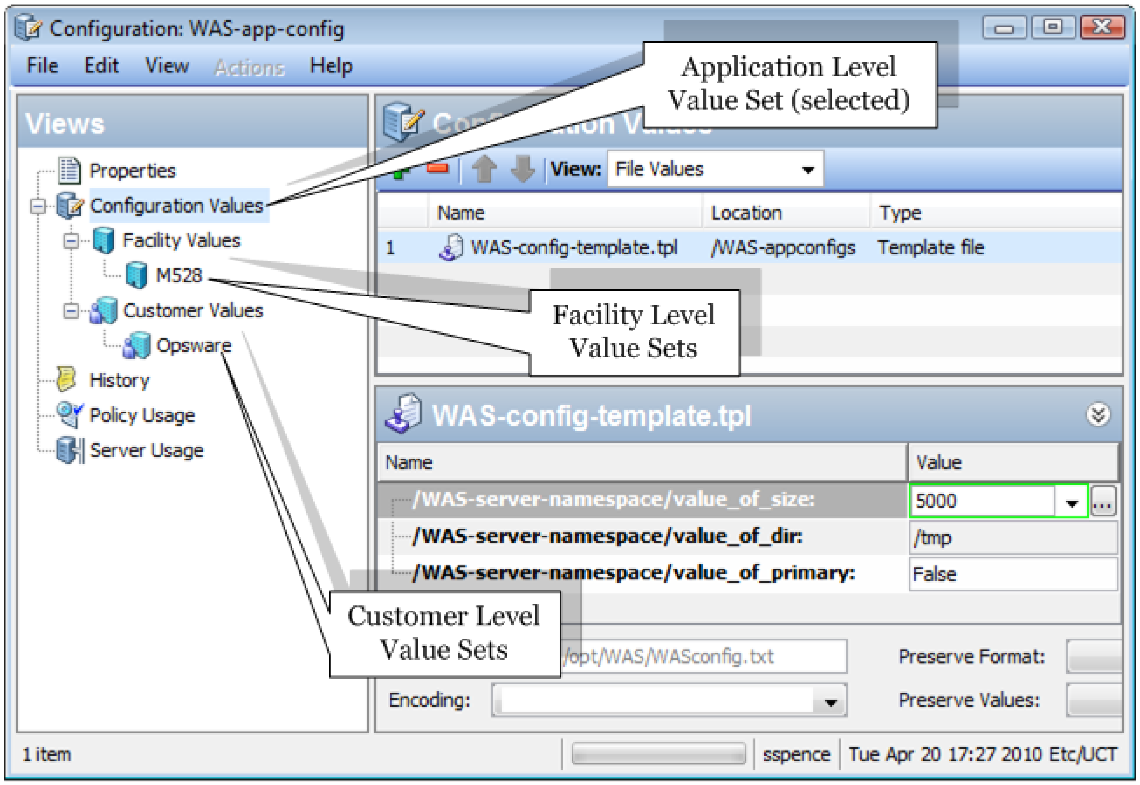
Value Set Editor at the Application, Facility and Customer Levels

Figure 4 below shows the value set editor at the application, facility and customer levels. This view is only available when you open the application configuration object.

- Selecting “Configuration Values” displays the value set editor at the application level. See Setting Values at the Application Level on page 43.

- Selecting a facility under “Facility Values” displays the value set editor at the facility level. See Setting Values at the Facility Level on page 44.
- Selecting a customer under “Customer Values” displays the value set editor at the customer level. See Setting Values at the Customer Level on page 44.

Figure 4 Value Set Editor at the Application, Facility and Customer Levels - Application Level Selected



	Name	Location	Type
1	WAS-config-template.tpl	/WAS-appconfigs	Template file

Name	Value
/WAS-server-namespace/value_of_size:	5000
/WAS-server-namespace/value_of_dir:	/tmp
/WAS-server-namespace/value_of_primary:	False

/opt/WAS/WASconfig.txt

Setting Values at the Application Level

The application level defines values that apply to the application configuration itself. It applies to all servers that have the application configuration attached, unless overridden by any level below.

To set values at the application level, perform the following steps. See Figure 4 for an example showing the value set editor at the application, facility and customer levels.

1 Open the application configuration object in the SA Client.

2 Open the Content node in the Views pane.

3 Select the “Configuration Values” node.

4 Select a template in the application configuration.

5 Select “File Values” from the “View:” drop-down list. This displays the value set editor in the lower right where you can set default values at the application level.

6 Enter values in the text boxes. Use the right-click menu items for additional editing capabilities. See also Setting Values in the Value Set Editor on page 39.

7 Optionally select File Preview from the “View:” drop-down list to see the resulting file.

8 Select File ➤ Save to save your changes.

Setting Values at the Facility Level

The facility level defines values for a facility. It applies to all servers in the specified facility that have the application configuration attached, unless overridden by any level below.

To set values at the facility level, perform the following steps. See Figure 4 for an example showing the value set editor at the application, facility and customer levels.

1 Open the application configuration object in the SA Client.

2 Open the Configuration Values view to display the Facility Values node.

3 Open the Facility Values node and select a facility.

4 Select a template in the application configuration.

5 Select File Values from the View drop-down list. This displays the value set editor in the lower right.

6 Enter values in the text boxes. Use the right-click menu items for additional editing capabilities. See also Setting Values in the Value Set Editor on page 39.

7 Optionally select File Preview from the View drop-down list to see the resulting values.

8 Select File ➤ Save to save your changes.

Setting Values at the Customer Level

The customer level defines values for a customer. It applies to all servers belonging to the specified customer that have the application configuration attached, unless overridden by any level below.

To set values at the customer level, perform the following steps. See Figure 4 for an example showing the value set editor at the application, facility and customer levels.

1 Open the application configuration object in the SA Client.

2 Open the Configuration Values view to display the Customer Values node.

3 Open the Customer Values node and select a customer.

4 Select a template in the application configuration.

5 Select File Values from the View drop-down list. This displays the value set editor in the lower right.

6 Enter values in the text boxes. Use the right-click menu items for additional editing capabilities. See also Setting Values in the Value Set Editor on page 39.

7 Optionally select File Preview from the View drop-down list to see the resulting values.

8 Select File ➤ Save to save your changes.

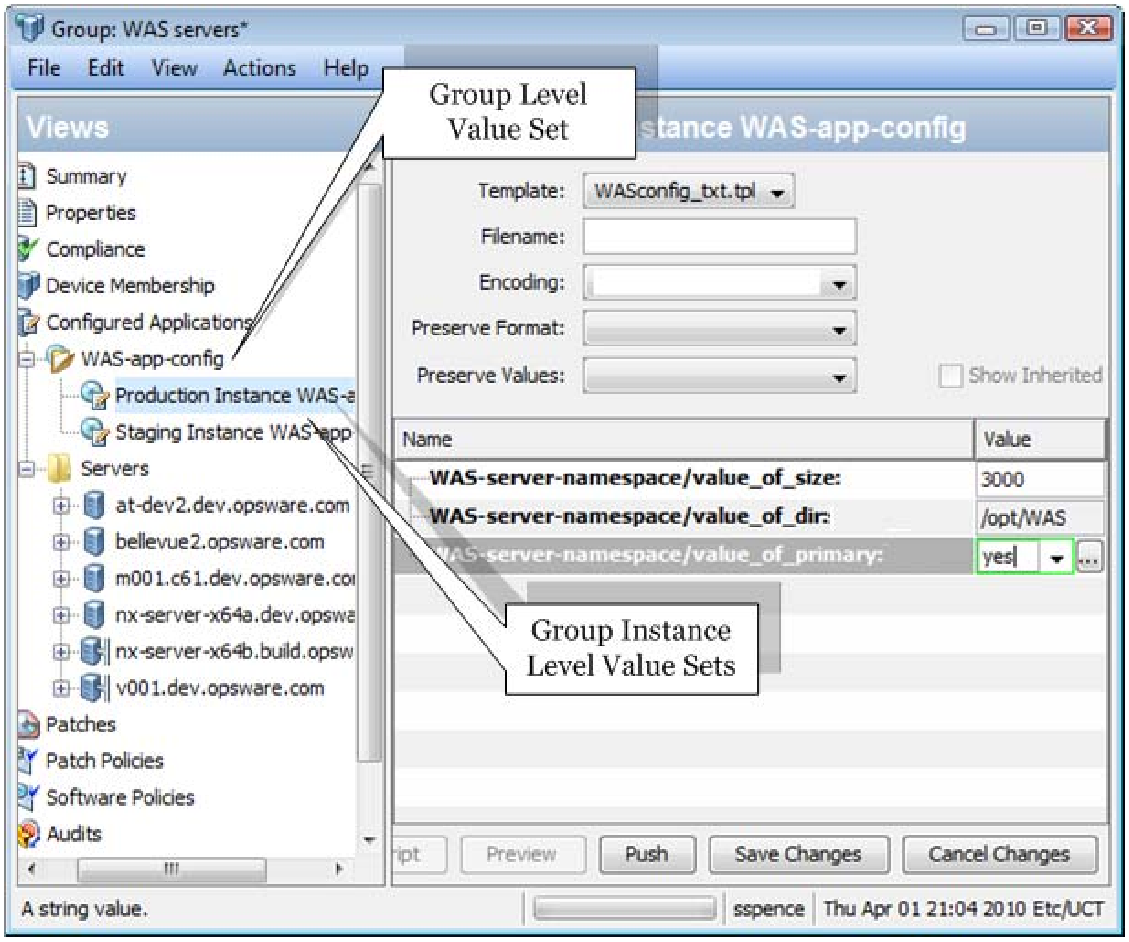
Value Set Editor at the Group Level

The group level defines values for a device group. It applies to all servers in the specified device group (provided the application configuration is attached to the device group), unless overridden by any level below.

Figure 5 below shows the value set editor at the group and group instance level. The group and group instance level are only available when the application configuration is attached to a device group. This view is only available when you open a device group to which the application configuration is attached. For more information, see Attaching an Application Configuration to a Server or Device Group on page 23.

- Selecting the application configuration named “WAS-app-config” displays the value set editor at the group level. See Setting Values at the Group Level on page 45.
- Selecting one of the application configuration instances displays the value set editor at the group instance level. This example shows two application configuration instances, named “Production Instance WAS-appconfig” and “Staging Instance WAS-appconfig”. The value set editor is displaying the value set for the “Production Instance WAS-appconfig”. See Setting Values at the Group Instance Level on page 46.

Figure 5 Value Set Editor at the Group Level and the Group Instance Level, Displaying Values for the Instance “Production Instance WAS-appconfig”



Template: WASconfig_txt.tpl
Filename:
Encoding:
Preserve Format:
Preserve Values:

Name	Value
WAS-server-namespace/value_of_size:	3000
WAS-server-namespace/value_of_dir:	/opt/WAS
WAS-server-namespace/value_of_primary:	yes

Setting Values at the Group Level

The group level defines values for a device group. It applies to all servers in the specified device group, unless overridden by any level below.

Before you can set values at the group level, you must attach the application configuration to a device group. See Figure 5 above for an example showing the value set editor at the group and group instance levels.

To set values at the group level, perform the following steps.

1 Open the device group in the SA Client. The application configuration must be attached to the device group.
2 Open the Configured Applications node in the View pane on the left.
3 Select the desired application configuration node under the Configured Applications node. This displays the value set editor on the right.
4 Select the desired template file in the "Template:" drop-down list.
5 Enter values in the text boxes. Use the right-click menu items for additional editing capabilities. See also Setting Values in the Value Set Editor on page 39.
6 Select the Save Changes button.

Setting Values at the Group Instance Level

The group instance level defines values for an instance of the application configuration attached to the device group. It applies to all servers in the specified device group, unless overridden by any level below.

Before you can set values at the group instance level, you must attach the application configuration to a device group. See Figure 5 above for an example showing the value set editor at the group and group instance levels.

To set values at the group instance level, perform the following steps.

1 Open the device group in the SA Client. The application configuration must be attached to the device group. See Attaching an Application Configuration to a Server or Device Group on page 23.
2 Open the Configured Applications node in the View pane on the left.
3 Open the desired application configuration node under the Configured Applications node.
4 Select the desired instance of your application configuration. For example, Figure 5 above shows the application configuration instance named "Production Instance WAS-appconfig" selected. This is an instance of the application configuration object named WAS-app-config. This application configuration is attached to the device group and two instances of the application configuration are defined.
5 Select the desired template file in the "Template:" drop-down list.
6 Enter values in the text boxes. Use the right-click menu items for additional editing capabilities. See also Setting Values in the Value Set Editor on page 39.
7 Select the Save Changes button.

Value Set Editor at the Server Level

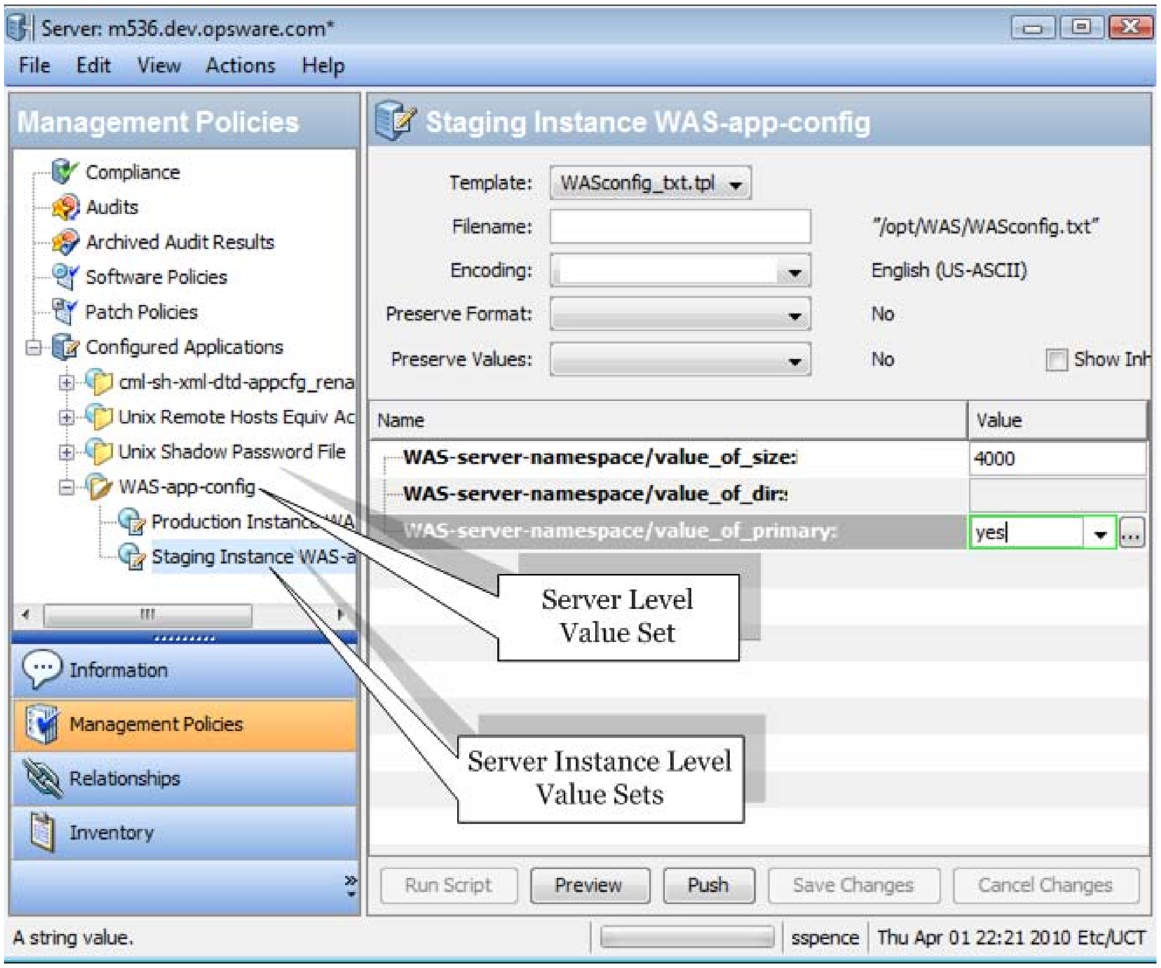
The server level defines values for a server. It applies to all instances of the application configuration on the specified server, unless overridden by the level below.

Figure 6 below shows the value set editor at the server and server instance level. This view is only available when you open a server to which the application configuration is attached.

- Selecting the application configuration named "WAS-app-config" displays the value set editor at the server level. See Value Set Editor at the Server Level on page 46.

- Selecting either of the instances named “Production Instance WAS-appconfig” or “Staging Instance WAS-appconfig” displays the value set editor at the server instance level. See Setting Values at the Server Instance Level on page 48.

Figure 6 Value Set Editor at the Server Level and the Server Instance Level, Displaying Values for the Instance “Staging Instance WAS-appconfig”



Template: WASconfig_txt.tpl
Filename: "/opt/WAS/WASconfig.txt"
Encoding: English (US-ASCII)
Preserve Format: No
Preserve Values: No

Name	Value
WAS-server-namespace/value_of_size:	4000
WAS-server-namespace/value_of_dir:	
WAS-server-namespace/value_of_primary:	yes

A string value. sspence Thu Apr 01 22:21 2010 Etc/UCT

Setting Values at the Server Level

The server level defines values for a server. It applies to all instances of the application configuration on the specified server, unless overridden by the level below.

Before you can set values at the server level, you must attach the application configuration to a server. See Figure 6 above for an example showing the value set editor at the server and server instance levels.

To set values at the server level, perform the following steps.

1 Open the server in the SA Client. The application configuration must be attached to the server. See Attaching an Application Configuration to a Server or Device Group on page 23.

2 Select the Management Policies tab on the left.

3 Open the Configured Applications node in the Management Policies pane on the left.

4 Select the desired application configuration node under the Configured Applications node.

5. Select the desired template file in the “Template:” drop-down list.
6. Enter values in the text boxes. Use the right-click menu items for additional editing capabilities. See also Setting Values in the Value Set Editor on page 39.
7. Select the Save Changes button.

Setting Values at the Server Instance Level

The server instance level defines values for one particular instance of an application configuration on the server. It applies only to the specified application configuration instance on the specified server and overrides all levels above.

Before you can set values at the server instance level, you must attach the application configuration to a server. See Figure 6 above for an example showing the value set editor at the server and server instance levels.

To set values at the server instance level, perform the following steps.

1. Open the server in the SA Client. The application configuration must be attached to the server. See Attaching an Application Configuration to a Server or Device Group on page 23.
2. Select the Management Policies tab on the left.
3. Open the Configured Applications node in the Management Policies pane on the left.
4. Open the desired application configuration node under the Configured Applications node.
5. Select the desired instance under the desired application configuration node.
6. Select the desired template file in the “Template:” drop-down list.
7. Enter values in the text boxes. Use the right-click menu items for additional editing capabilities. See also Setting Values in the Value Set Editor on page 39.
8. Select the Save Changes button.

About Running Scripts with Application Configurations

You can add scripts to an Application Configuration that are executed before or after the configuration values are copied to the target server.

For example, you might want to add a **Pre-install script** that stops an application and a **Post-install script** that restarts the application after configuration changes have been made. If an error occurs during the push or post-install script, you can run a **Post-error script**.

Or you might need a **Data-manipulation script** to handle non-text configuration data. If you are configuring an IIS server, you can use a Data-manipulation script to read the metabase information into a flat file. When the information in the flat file gets parsed with the configuration template, you can run a Post-install script to write the updated information back to the metabase information.

When pushing an application configuration that contains a JScript or VBScript pre-install, post-install or post-error script, the push may succeed even if the script fails. In these cases, the push ignores the script errors. The application configuration does not detect the script failure and allows the push to complete without errors.

If you plan to use these types of scripts, you must make sure that the scripts are free of errors and ensure the script returns a non-zero exit status by invoking WScript.Quit(<status>).

Types of Application Configuration Scripts

Table 3 below lists the types of scripts you can use in application configuration objects. The script type specifies when the script is invoked. You can define at most one of each script type. If you define a script but do not specify one of these types, that script will be treated like a configuration template. That is, it will be pushed to the server but not executed.

Table 3 Types of Scripts, Specifying When the Script Runs

Script Type	Description
Data-manipulation	Runs before any pre-install script and serves the purpose of parsing a non-text configuration file to make it parseable by the CML template. The data-manipulation script is also useful when you only want to scan and import an existing file managed by the application configuration. If this script fails, the application configuration is not pushed to the server.
Pre-install	Runs before an actual push occurs. For example, a pre-install script could stop an application or service. If this script fails, the application configuration is not pushed to the server.
Post-install	Runs after the actual push occurs. For example, a post-install script could restart a service after a push.
Post-error	Runs only if the push fails or if the post-install script fails. For example, a post-error script could restore a backed-up file.

To specify the script type, perform the following steps:

1 In the SA Client, open the application configuration object that contains the template.
2 Select the Configuration Values view to display the templates contained in the application configuration object.
3 Select the template and right-click to display the menu.
4 Select the script type, listed in Table 3.
5 Select the **File ➤ Save** menu.

See also Creating a Template from a Script on page 21.

Table 4 below lists the types of scripts you can use in application configuration objects. This type specifies the syntax and execution environment of the script.

Table 4 Types of Script Source

Script Source Type	Description
Windows .BAT script	Windows batch command files.
Windows .JS script	Javascript files running on Windows.
Windows .CMD script	Windows batch command files.
Windows .VBS script	Windows Visual Basic script.
Windows .WSF script	Windows script file.
Windows .PY script	Python file running on Windows.
Unix .SH script	Unix shell scripts.
Other Unix scripts	Any other scripts that run on Unix.

To specify the script type, perform the following steps:

1 In the SA Client, open the template.
2 Select the Properties view.
3 In the Type field, select the script type, listed in Table 4.
4 Select the **File ➤ Save** menu.

See also Creating a Template from a Script on page 21.

About Pushing Application Configurations to Servers

Anytime you change values in a value set, to merge those changes with the configuration file on the target server you must push the application configuration to the server. When you push an application configuration, all the values in the value set replace the values in the configuration files on the target managed servers. If the configuration file does not exist on the target server, a new file is created on the server when you push. In addition, all the scripts in the application configuration are executed based on the script type.

When you push an application configuration, the following events occur.

- SA runs the data-manipulation script, if specified.
- SA generates the target configuration files from the templates and value sets.
- SA backs up the existing configuration files.
- SA runs the pre-install script, if specified.
- SA copies the generated configuration files to the server.
- SA executes the post-install script, if specified.
- SA executes the post-error script, if specified and if either the pre-install script fails or the post-install script fails or the copy operation fails.

For information about how application configurations are pushed in the context of Software Policies and Audits, see Using Application Configurations in Software Policies on page 57.

For more information about using scripts in application configurations, see Creating a Template from a Script on page 21 and Managing Non-Text Configurations by Running a Data Manipulation Script on page 22.

For instructions on how to push an application configuration, see Pushing Application Configurations on page 26.

➤ The way in which sequences (of lists and scalars) are merged when you push depends upon how values have been set in the Application Configuration inheritance hierarchy and what sequence merge modes have been configured in the CML template for the Application Configuration. For more information about sequence merging, see Sequence Aggregation on page 164.

Application Configuration Compliance

Application configuration compliance enables you to determine whether or not the values of an application configuration attached to a server (or a group of servers) match the configuration file values on the target server.

A server is considered compliant if the target configuration file values match the values defined in the application configuration. When a target configuration does not match the values defined in the application configuration, the server is considered non-compliant.

The SA Client displays the following compliance statuses for application configurations:

- **Compliant**: All of the values in the application configurations attached to a server or device group (or several servers and groups) match the configuration values on the target server. Represented by the ● icon.

  For Device Groups, AppConfig compliance is based upon the compliance status of all servers (and servers in any subgroups) that belong to a group. By default, group compliance is determined by a default threshold: if more than five percent of all servers in a group have a status of Non-Compliant, the entire group is considered Non-Compliant. To change this default setting, see Changing Device Group Compliance Settings in the *SA User Guide: Application Automation*.

- **Non-compliant**: At least one of the values defined in an application configuration does not match the values in a configuration file (or files) on a target server. Represented by the ✗ icon.

  For Device Groups, non-compliance is based upon the compliance status of all the servers (and servers in any subgroups) that belong to a group. By default, group non-compliance is determined by a default threshold: if more than five percent of all servers in a group have a status of Non-Compliant, the entire group is considered Non-Compliant. To change this default setting, see Changing Device Group Compliance Settings in the *SA User Guide: Application Automation*.

- **Scan Started**: The application configuration compliance information is currently being calculated. Represented by the ⧗ icon.
- **Scan Needed**: The application configuration compliance information is undefined, perhaps because a compliance scan was never run (for example, on a new installation), or the configuration on the server (or servers in the device group) changed since the last time information was reported to the SA Client. Represented by the ○ icon.
- **Not Applicable**: The application configuration compliance information does not apply and is represented by a dash (—). This is displayed for templates that do not have the “Auditable” property checked. For more information, see Creating a Configuration Template on page 16.

You can view application configuration compliance for individual servers or groups of servers:

- Application Configuration Compliance for a Single Server
- Application Configuration Compliance for Multiple Servers

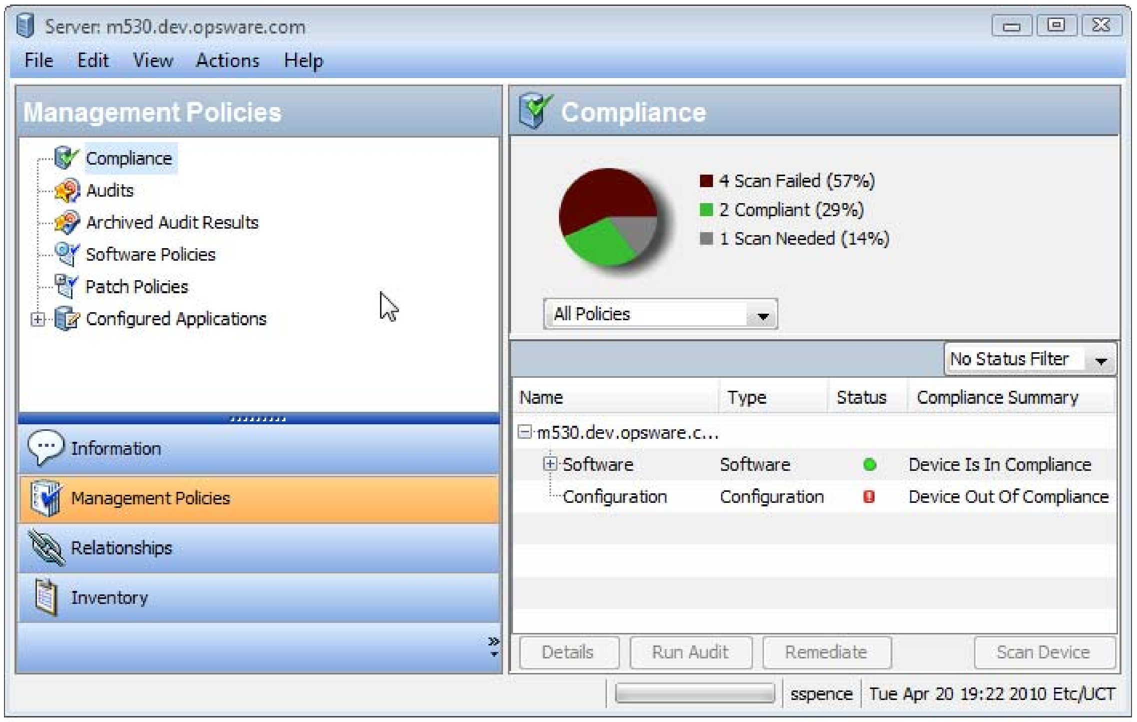
If you make any changes to an application configuration, such as editing its values in the value set editor, any server or group of servers it is attached to will cause a compliance status of Scan Needed.

Application Configuration Compliance for a Single Server

For a single server, the compliance view displays overall compliance for all application configurations attached to the server. If more than one application configuration is attached to the server, then you can see the aggregate compliance status for all application configurations, plus each individual configuration’s compliance status.

Figure 7 shows a single server’s application configuration compliance.

Figure 7 Application Configuration Compliance For a Server



Name	Type	Status	Compliance Summary
m530.dev.opsware.c...			
Software	Software	●	Device Is In Compliance
Configuration	Configuration	●	Device Out Of Compliance

If any differences are discovered between the application configuration and the actual configuration file on the target server, the lower pane shows the category that is non-compliant. If the server has several application configurations attached to it, and any one of the configuration files targeted by the application configuration is different from the application configuration, then the server's status is non-compliant.

For more information on how to run an application configuration compliance scan, see Scanning Servers for Application Configuration Compliance on page 56.

Application Configuration Compliance for Multiple Servers

You can view the application configuration compliance status for multiple servers. From the SA Client navigation pane, select Devices then select Device Groups or Servers. Select a device group or a set of servers, then from the View menu select Compliance. This displays the aggregate compliance status for the selected servers.

An application configuration attached to a group of servers is considered compliant if less than five percent of the servers in the group are out of compliance. If over five percent are out of compliance, the aggregate compliance is considered non-compliant. You can change this percent by selecting the Administration tab in the SA Client, then selecting Compliance Settings.

The details pane for a group of servers in the Compliance View shows whether or not all of the application configurations are compliant, but does not expand to show a breakdown of individual servers and application configurations.

You can view server group application configuration compliance status in the following ways:

- Viewing Application Configuration Compliance For Multiple Servers on page 54.
- Viewing Application Configuration Compliance For Multiple Device Groups on page 54.

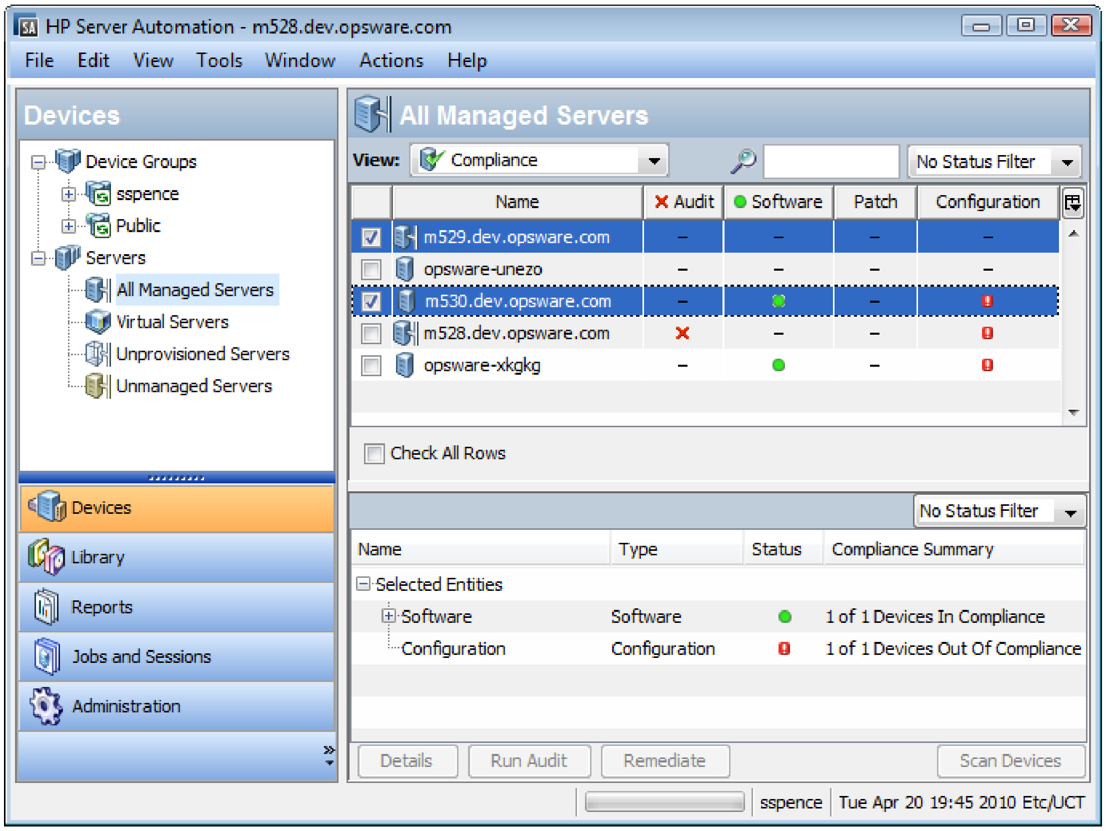
- Viewing Application Configuration Compliance For a Single Device Group on page 55.

Viewing Application Configuration Compliance For Multiple Servers

To view application configuration compliance for multiple servers, perform the following steps:

1. From the SA Client Navigation pane, select **Devices ➤ Servers ➤ All Managed Servers**.
2. From the View drop-down list, select **Compliance**.
3. To see compliance levels for more than one server, select the check box next to the servers, and a roll up of compliance for the selected servers displays in the bottom details pane, as shown in Figure 8.

Figure 8 Application Configuration Compliance for Multiple Servers



View: Compliance

	Name	✗ Audit	● Software	Patch	Configuration
✓	m529.dev.opsware.com	–	–	–	–
	opsware-unezo	–	–	–	–
✓	m530.dev.opsware.com	–	●	–	❗
	m528.dev.opsware.com	✗	–	–	❗
	opsware-xkgkg	–	●	–	❗

Name	Type	Status	Compliance Summary
Selected Entities			
Software	Software	●	1 of 1 Devices In Compliance
Configuration	Configuration	❗	1 of 1 Devices Out Of Compliance

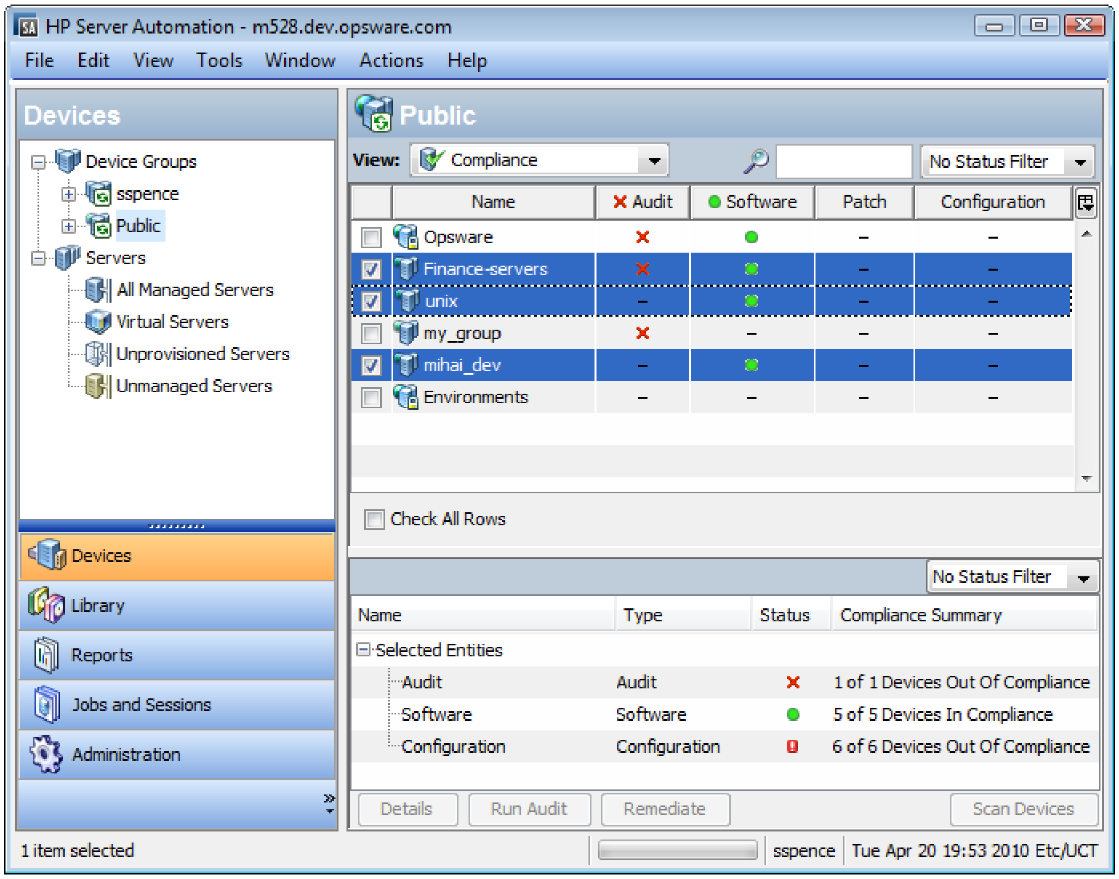
Viewing Application Configuration Compliance For Multiple Device Groups

To view application configuration compliance for multiple device groups, perform the following steps:

1. From the SA Client navigation pane, select **Devices ➤ Device Groups**.
2. Select a device group or a folder containing device groups.
3. From the View drop-down list, select Compliance. This displays the compliance status for all the groups.

4 To see compliance levels for more than one group, select the check box next to the servers, and a summary of compliance for the selected groups displays in the bottom details pane, as shown in Figure 9.

Figure 9 Application Configuration Compliance for Multiple Device Groups



	Name	✗ Audit	● Software	Patch	Configuration
☐	Opsware	✗	●	–	–
☑	Finance-servers	✗	●	–	–
☑	unix	–	●	–	–
☐	my_group	✗	–	–	–
☑	mihai_dev	–	●	–	–
☐	Environments	–	–	–	–

Name	Type	Status	Compliance Summary
Selected Entities			
Audit	Audit	✗	1 of 1 Devices Out Of Compliance
Software	Software	●	5 of 5 Devices In Compliance
Configuration	Configuration	!	6 of 6 Devices Out Of Compliance

Viewing Application Configuration Compliance For a Single Device Group

To view application configuration compliance for one device group, perform the following steps:

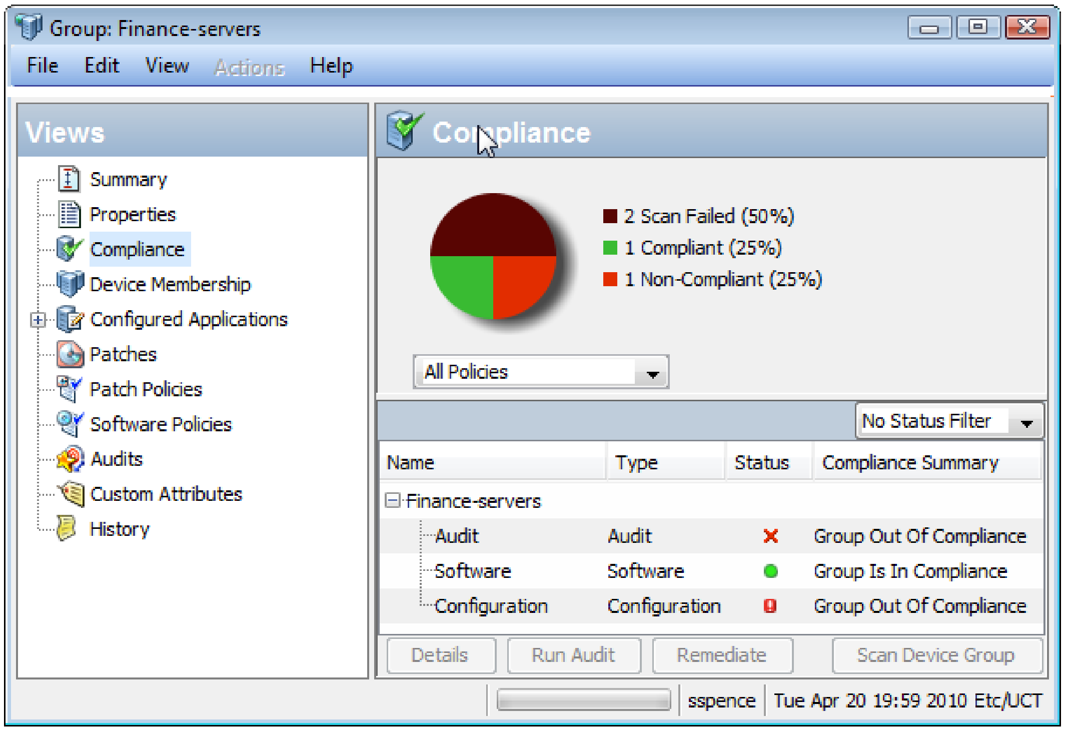
1 From the SA Client navigation pane, select **Devices ➤ Device Groups**.

2 Navigate to the desired device group and select it.

3 Right-click and select **Open** or select **Actions ➤ Open**. This displays the device group.

4 From the Views pane, select Compliance. This displays aggregate compliance for each policy type for all members of the group as a whole, as opposed to compliance status for each individual server, as shown in Figure 10 below.

Figure 10 Application Configuration Compliance for a Device Group



- 🟥 2 Scan Failed (50%)
- 🟩 1 Compliant (25%)
- 🟥 1 Non-Compliant (25%)

Name	Type	Status	Compliance Summary
Finance-servers			
Audit	Audit	✗	Group Out Of Compliance
Software	Software	●	Group Is In Compliance
Configuration	Configuration	!	Group Out Of Compliance

Scanning Servers for Application Configuration Compliance

After an application configuration has been pushed to a server, the configuration file on the server can be changed or altered, either intentionally or by accident. Or the values defined in the application configuration may have changed. When a configuration file's values on a target server do not match the values defined in the application configuration, the configuration file is considered non-compliant.

You can scan for configuration compliance on a server to determine if any of the configuration files on the server are out of compliance with the values stored in the configuration templates. You can schedule the scan to occur at regular intervals. To scan a server or multiple servers for configuration compliance, perform the following steps:

1 From the SA Client Navigation pane, select Devices.

2 Select either Device Groups or All Managed Servers. If you selected Device Groups, select a device group to display the servers that belong to it.

3 From the content pane, select a server. You can also select multiple servers or device groups and scan them all.

4 From the **Actions** menu, select **Scan ➤ Configuration Compliance** or select **Schedule ➤ Configuration Compliance Scan.**

   - If you selected **Scan ➤ Configuration Compliance,** SA scans the devices to determine compliance and displays the status in the Scan Configuration Compliance screen.
   - If you selected **Schedule ➤ Configuration Compliance Scan**, SA displays the Schedule Job screen where you can specify when you want the job to complete and other job parameters.

Auditing Application Configurations

With SA you can audit configuration files on servers to determine whether or not those files meet your organization's configuration standards. You can create audit rules that specify how a configuration file on your servers should be defined and audit those servers regularly to check that a configuration file is configured properly. If you find a mismatch between the audit rule definition and the target configuration file values, you can remediate the servers to fix the problem.

For example, to ensure that an /etc/hosts file on a managed server only defines certain host names for a specific IP address, you can define an audit rule that specifies the acceptable list of host name and IP address pairs. When you run the audit, if the hosts file contains any values other than what you specified in the rule, the audit results will show an error and you can remediate the problem.

The general process of auditing application configurations follows these steps:

1. **Create an Audit and Audit Rule**: To audit a configuration file on a server, you first create an audit. When you create the audit, you specify a source server (or a snapshot or a snapshot specification) upon which the configuration rule will be based. Then you select an application configuration template to construct the rule. The rule defines the exact values you want to check in the target configuration files. For each audit rule, specify the location of the configuration file on the target server.
2. **Select Target Servers**: In the audit, select the target servers for the audit. You can select a single server, multiple servers, or groups of servers.
3. **Run or Schedule the Audit**: You can schedule the audit to run once or on a recurring basis. You can also specify email addresses where audit results will be sent.
4. **Check Audit Results**: Check the audit results to see if the configuration files on the target servers match the values defined in the audit rule. If there are discrepancies, you can compare the rule and the target file to see the differences so you can decide how to remediate the servers.
5. **Remediate Servers**: To fix any differences found in the audit results, you can remediate the servers or any of the rules or all of the rules, to ensure that the target configuration matches the rule.

For more information on using audits and snapshots, see Audit and Remediation in the *SA User Guide: Application Automation*. In particular, see the section "Configuring Application Configuration Rule."

Using Application Configurations in Software Policies

Application configurations can be a powerful tool when used inside a software policy. A software policy defines an ideal state of an application including all the packages, patches, scripts, and other objects to be installed on a server, as well as the way configuration files for the application should be set on the server. When you install the software policy on a managed server, SA applies all the contents to the servers targeted by the policy, including all the values defined in the application configuration.

Using the compliance view in the SA Client, you can view the compliance status of the software installed from the policy. For example, if someone removes a patch from the software policy or installs a new package on the server or changes one of the configuration files defined in the policy, the policy will show as out of compliance in the compliance view. To make sure the application is installed and configured correctly, you can remediate the server.

For more information on using and creating software policies, see Software Management in the *SA User Guide: Application Automation*.

To use application configurations in a software policy, follow these steps:

1. **Define Application**: Before building a software policy, an application expert gathers all the necessary packages and patches that comprise the application. In addition, gather the configuration templates that define and manage the configuration files associated with the application.
2. **Import Packages and Patches into SA**: Once the components of a software policy have been defined, import all the packages and patches that comprise the application into the SA Library, so they can be placed in the software policy.
3. **Create Application Configuration and Set Values**: Define the configuration values that will be used to generate the configuration files. For example, if the software policy is being created to deploy an Apache Web Server, the application expert uses the value set editor to define the default values for the httpd.conf file. Add any pre- or post-installation scripts to the application configuration, for example, to restart the Apache service after the application configuration is pushed during the software policy remediation.
4. **Test Application Configurations**: Before adding the application configurations to a software policy and deploying the application to a server, it is a good idea to attach the application configuration to a server and make sure that the application is working properly before creating the software policy. You can preview pushing configurations to a server to verify their correctness.
5. **Create Software Policy**: Once all of the components of the software policy have been defined, created, and imported into SA, the application expert creates a software policy that specifies the software to be installed, the order in which its components will be installed, including all of the patches, packages, and application configurations. When the software policy is saved in the SA Library, it is then accessible to the systems administrators who deploy, test, and manage the application.
6. **Attach Policy to Servers or Groups of Servers**: After the software policy has been created and saved, the system administrator attaches the policy to a server or group of servers in a device group.
7. **Remediate Servers to Install the Software**: The system administrator deploys the software to one or more servers by remediating the software policy on servers. Remediation ensures that everything defined in the policy is deployed on the target servers in the order specified in the policy.
8. **Test Application and Iterate Changes**: After the system administrator installs the application using software policy remediation, before the application is put into a production environment, the application should be tested to make sure it works properly and contains the correct components. In addition, each part of the application that is affected by its configuration files should be checked to ensure it is configured properly.
9. **Roll Out the Application**: After the application is deployed and in use, the system administrator can perform ongoing management and maintenance tasks, such as running software compliance scans to determine the compliance status of servers where the application is deployed, remediating non-compliant servers, and generating software compliance reports.

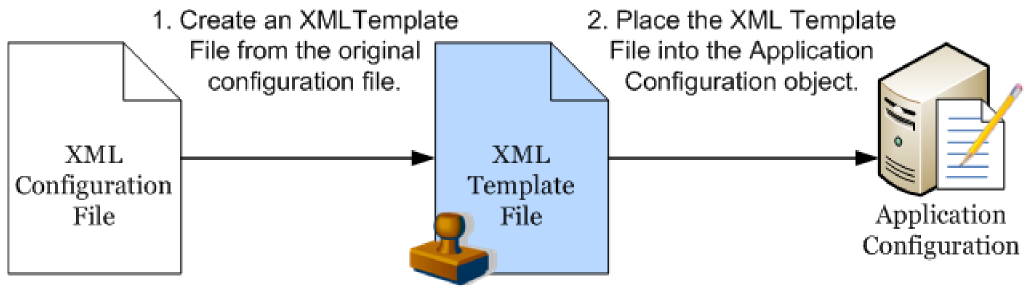
For more information on using software policies, see the *SA Policy Setter's Guide* and the *SA User Guide: Application Automation*.

4 Managing XML Configuration Files

With SA you can manage XML configuration files from a central location and propagate changes across multiple servers in your data center. You can create, edit, and store configuration file values to ensure that the XML configuration files on your managed servers are correct. You can manage XML files that use a DTD as well as XML files that do not.

This chapter discusses how XML configuration templates are structured so you can manage generic (non-DTD) XML files, as well as XML files that reference a DTD. Since XML is well-structured, SA needs only a minimum amount of information to be able to model and manage XML-based configuration files.

To manage XML configuration files you first need to create a **template** file for your XML configuration file. After creating the template, you must add it to an **application configuration** object so you can manage, edit, and make changes to the native configuration files on managed servers.


1. Create an XMLTemplate File from the original configuration file.
2. Place the XML Template File into the Application Configuration object.
XML Configuration File
XML Template File
Application Configuration

The following section describes a simple XML file and shows how to create an application configuration for a non-DTD based XML file and one for a DTD-based XML file.

See also the following examples:

- XML Tutorial 1 - Create a Non-DTD XML Configuration Template on page 71.
- XML Tutorial 2 - Create an XML-DTD Configuration Template on page 77

Example: Travel Manager Application and XML Configuration File

This section describes an example web application that uses a simple XML file to control its configuration and shows how to create an application configuration to manage that file.

Travel Manager is a web application designed to help people manage their travel by performing such tasks as booking hotels, rental cars, tracking expenses, and so on. Travel Manager uses the MySQL Relational Database Management System (RDMS) as the repository for user data and some of the application's configuration data.

Since the Travel Manager is designed to be deployed over many different networks, each with a different database server, it is important to provide flexibility in the information used to connect to the MySQL server. The application is designed to retrieve connection information from an XML configuration file, `mysql.xml`.

With application configurations, you can set the configuration file values necessary for accessing the local MySQL database. For example, the user name and password used to open a connection to the database may be different for each installation of the Travel Manager application. Modifications to these values can be made to the configuration file without requiring a recompilation of the Travel Manager application code.

Only four values in the file `mysql.xml` are required for the Travel Manager to be able to connect to the local MySQL database, each of which is represented as an element in the application's XML file:

- **Host**: Host name of the server on which the MySQL RDMS has been installed.
- **Name**: Name of the database on the host server.
- **User**: User name credentials used to open a connection to the database.
- **Password**: Password necessary to open a connection to the database.

Contents of the Travel Manager mysql.xml File

The following is an example of the Travel Manager `mysql.xml` configuration file:

```
<?xml version="1.0" ?>
<db-config>
  <db-host>localhost</db-host>
  <db-name>wrightevents</db-name>
  <db-user>root</db-user>
  <db-password>hp-pass</db-password>
</db-config>
```

Contents of the Travel Manager mysql.xml DTD-Based XML File

The following is an example of the Travel Manager `mysql.xml` configuration file that references a DTD:

```
<?xml version="1.0"?>
<!DOCTYPE db-config PUBLIC "-//Williams Events//Travel Manager//EN"
"mysql2.dtd">
<db-config>
<db-host>localhost</db-host>
<db-name>wrightevents</db-name>
<db-user>root</db-user>
<db-password>hp-pass</db-password>
</db-config>
```

Non-DTD XML Configuration Templates

You can create a non-DTD based XML configuration template written as a single XML comment with three pieces of required information that enables the template to extract and store values from a target XML file:

- `ACM-NAMESPACE`: Defines the location where values read from the target XML file on the managed server will be stored in the database. The name space must be unique and the path must start with a forward slash (/).
- `ACM-FILENAME-DEFAULT`: Defines the default absolute path of the target XML configuration file on the managed server.
- `ACM-FILENAME-KEY`: Defines the location in the name space where the target XML configuration file name will be stored.

When you set a configuration template's properties to use XML syntax, the labels displayed in the value set editor are the same as the tag names for the each corresponding element inside the XML file.

For a full list of template settings for XML templates, see XML Configuration Template Settings on page 69.

➤ For information on setting the parser syntax to XML for a configuration template, see Creating a Configuration Template on page 16.

Non-DTD XML Configuration Template for mysql.xml

The following example shows the XML configuration template based on the `mysql.xml` file. The template file is named to `mysql.tpl` to indicate it is a template file.

```
<!--
ACM-NAMESPACE = /TravelManager/
ACM-FILENAME-KEY = /files/TravelManager
ACM-FILENAME-DEFAULT = /var/www/html/we/mysql.xml
ACM-TIMEOUT = 1
-->
```

This example shows that the XML configuration template references the target XML file (`/var/www/html/we/mysql.xml`), so it can be parsed by the application configuration parser, and its values read and stored in the SA Library.

The `mysql.tpl` configuration template contains the following required information:

- `ACM-NAMESPACE`: Defines the location where values read from the `mysql.xml` file on the managed server will be stored in the database. The name space must be unique and the path must start with a forward slash (/).
- `ACM-FILENAME-DEFAULT`: Defines the default absolute path of the `mysql.xml` file on the managed server.
- `ACM-FILENAME-KEY`: Defines the location in name space where the `mysql.xml` file name will stored.
- `ACM-TIMEOUT`: (Optional) Represents the number of minutes that are added to the configuration template's default timeout value of ten minutes during a push.

The default timeout value for an entire application configuration is ten minutes plus the timeout for each configuration template inside the application configuration. So if this template were the only template inside an application configuration (which has a ten minute timeout), and this value is set to 1, the overall timeout value for the entire application configuration when pushed would be eleven minutes.

DTD-Based XML Configuration Templates

An XML-DTD configuration template is actually just an XML DTD with some application configuration options defined in the comments. Since the DTD standard defines the syntax and layout of an XML file, there is no need to redefine that syntax in another language.

For DTD-based XML files, XML-DTD configuration templates require the same three basic attributes required for a generic XML file — `ACM-NAMESPACE, ACM-FILENAME-DEFAULT, and ACM-FILENAME-KEY` – plus three other attributes:

- `ACM-DOCTYPE`: Defines the name of the root element in the XML file. The root element follows the opening `<!DOCTYPE` declaration found in the target XML configuration file.
- `ACM-DOCTYPE-SYSTEM-ID`: Defines the name of the associated DTD file on the managed server. This value is typically found in the XML configuration file as the SYSTEM attribute in the DOCTYPE element.
- `ACM-DOCTYPE-PUBLIC-ID`: Defines a string that represents a public identifier of the XML document. This value is typically found in the XML configuration file as the PUBLICID attribute of a DOCTYPE element.

For a complete list of all XML configuration file attributes, see XML Configuration Template Settings on page 69.

XML-DTD Configuration Template for mysql.xml

The following is an example of the configuration template created for the Travel Manager DTD-based XML file.

```
<!--
ACM-FILENAME-KEY = /files/TravelManager
ACM-FILENAME-DEFAULT = /var/www/html/we/mysql.xml
ACM-NAMESPACE = /TravelManager/
ACM-TIMEOUT = 1
ACM-DOCTYPE = db-config
ACM-DOCTYPE-SYSTEM-ID = mysql.dtd
ACM-DOCTYPE-PUBLIC-ID = -//Williams Events//Travel Manager//EN
-->
<!ELEMENT db-config (db-host,db-name,db-user,db-password)>
<!ELEMENT db-host (#PCDATA)>
<!ELEMENT db-name (#PCDATA)>
<!ELEMENT db-user (#PCDATA)>
<!ELEMENT db-password (#PCDATA)>
```

In this example, the DOCTYPE attributes reference specific XML and DTD information that enables the parser to extract information from both the DTD file and the referenced XML file.

Specifically, the DTD-based XML configuration templates must contain the following information:

- `ACM-DOCTYPE`: The root node of the targeted XML file. For `mysql.xml`, the root node is `dbconfig`.
- `ACM-DOCTYPE-SYSTEM-ID`: The name of the DTD file being targeted by the configuration template. In the example of `mysql.xml`, the DTD being used is named `mysql.dtd`.
- `ACM-DOCTYPE-SYSTEM-ID`: The public ID of the XML file.

Customizing XML DTD Element Display

There are two optional settings you can add to your XML-DTD configuration template that allow you to customize how elements from the target XML-DTD configuration file are displayed in the value set editor in the SA Client. The `ACM-PRINTABLE` and `ACM-DESCRIPTION` optional settings allow you to control the names of elements as they appear in the SA Client:

- `ACM-PRINTABLE`: Defines the label for each element from the XML file that is displayed in the value set editor when the XML-DTD template is shown in the SA Client.
- `ACM-DESCRIPTION`: Defines mouse-over text when a user moves a mouse pointer over the field defined in `ACM-PRINTABLE` in the value set editor in the SA Client.

Explicit versus Positional Display Settings

You can set the printable and description values for attributes and elements inside the XML-DTD configuration template in either of two ways: positionally or explicitly.

- With *positional* definitions, `ACM-PRINTABLE` and `ACM-DESCRIPTION` are inserted directly after the element or attribute they are describing inside the XML-DTD configuration template.
- With *explicit* definitions, `ACM-PRINTABLE` and `ACM-DESCRIPTION` can be defined anywhere in the template.

Adding Positional Custom Display Settings

The positional method for adding element tables and mouse-over text to an XML template is to add a comment immediately after the element or attribute definition you want to define, and in that comment set the `ACM-PRINTABLE` and `ACM-DESCRIPTION` values. In other words, for either XML elements or attributes, you can specify a label and a mouse-over description for the label directly.

In the following example, each XML element from `mysql.xml` defines a `ACM-PRINTABLE` and `ACM-DESCRIPTION` setting immediately after each element in the XML-DTD template.

```
<!ELEMENT db-config (db-host,db-name,db-user,db-password)>
<!--
ACM-PRINTABLE = database configuration
ACM-DESCRIPTION = The db-config element specifies the data structure that
contains the information needed to connect to a database.
-->

<!ELEMENT db-host (#PCDATA)>
```

```
<!--
ACM-PRINTABLE = database hostname
ACM-DESCRIPTION = The db-host element specifies the name of the host computer
(the server) on which the database engine is running.
-->

<!ELEMENT db-name (#PCDATA)>
<!--
ACM-PRINTABLE = database name
ACM-DESCRIPTION = The db-name element specifies the name of the database.
-->

<!ELEMENT db-user (#PCDATA)>
<!--
ACM-PRINTABLE = database user
ACM-DESCRIPTION = The db-user element specifies the user identification used
to connect to the database.
-->

<!ELEMENT db-password (#PCDATA)>
<!--
ACM-PRINTABLE = database passsword
ACM-DESCRIPTION = The db-password element specifies the password used to
connect to the database.
-->
```

Adding Explicit Custom Display Settings

The explicit method for adding settings to an XML-DTD template allows you to define `ACM-PRINTABLE` and `ACM-DESCRIPTION` values anywhere in the configuration template by specifying the element name with the `ACM-ELEMENT` tag and optionally the attribute name with the `ACM-ATTRIBUTE` tag.

For this method the `ACM-ELEMENT` tag is required, even when defining printable and description values for attributes, because attributes are always associated with specific elements.

Once you have set the `ACM-ELEMENT` and the `ACM-ATTRIBUTE` tags, you can also set the `ACM-DESCRIPTION` and `ACM-PRINTABLE` tags within the same comment block. You should only use one definition per comment-block. In other words, define a `ACM-PRINTABLE` and `ACM-DESCRIPTION` for a single element, and then start a new comment block for the next element.

The `ACM-ELEMENT` tag and `ACM-ATTRIBUTE` tag (when applicable) should be defined before the `ACM-PRINTABLE` and `ACM-DESCRIPTION` tags.

For example, to customize the `mysql.tpl` template, you would construct the template as follows:

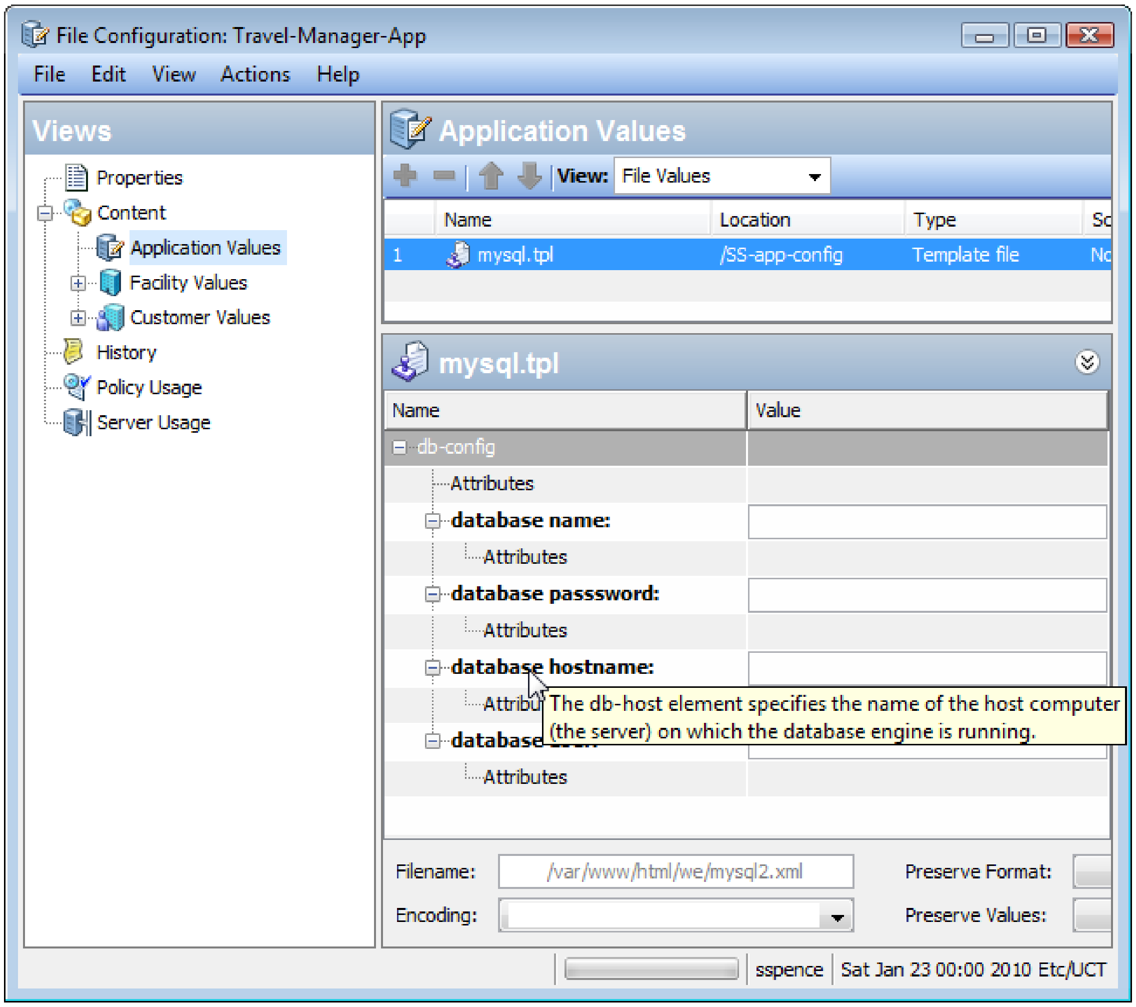
```
<!--
ACM-TIMEOUT = 1
ACM-FILENAME-KEY = /files/TravelManager
ACM-FILENAME-DEFAULT = /var/www/html/we/mysql2.xml
ACM-NAMESPACE = /TravelManager/
ACM-DOCTYPE = db-config
```

```
ACM-DOCTYPE-SYSTEM-ID = mysql.dtd
ACM-DOCTYPE-PUBLIC-ID = -//Williams Events//Travel Manager//EN
-->

<!ELEMENT db-config (db-host,db-name,db-user,db-password)>
<!ELEMENT db-host     (#PCDATA)>
<!ELEMENT db-name     (#PCDATA)>
<!ELEMENT db-user     (#PCDATA)>
<!ELEMENT db-password (#PCDATA)>

<!--
ACM-ELEMENT = db-config
ACM-PRINTABLE = database configuration
ACM-DESCRIPTION = The db-config element specifies the data structure that
contains the information needed to connect to a database.
-->

<!--
ACM-ELEMENT = db-host
ACM-PRINTABLE = database hostname
ACM-DESCRIPTION = The db-host element specifies the name of the host computer
(the server) on which the database engine is running.
-->

<!--
ACM-ELEMENT = db-name
ACM-PRINTABLE = database name
ACM-DESCRIPTION = The db-name element specifies the name of the database.
-->

<!--
ACM-ELEMENT = db-user
ACM-PRINTABLE = database user
ACM-DESCRIPTION = The db-user element specifies the user identification used
to connect to the database.
-->

<!--
ACM-ELEMENT = db-password
ACM-PRINTABLE = database passsword
ACM-DESCRIPTION = The db-password element specifies the password used to
connect to the database.

-->
```

Customizing How Elements Display in the SA Client

In both cases, whether you add these attributes positionally or explicitly, the end result is the same: the value set editor displays the element names (defined in ACM-PRINTABLE) and the mouse-over text (defined in ACM-DESCRIPTION) in the SA Client, as shown in Figure 11.

Figure 11 Custom Element Names and Mouse-Over Text



	Name	Location	Type	Sc
1	mysql.tpl	/SS-app-config	Template file	Nc

Name	Value
db-config	
Attributes	
database name:	
Attributes	
database passsword:	
Attributes	
database hostname:	
Attribu	
database	
Attributes	

The db-host element specifies the name of the host computer (the server) on which the database engine is running.

Filename: /var/www/html/we/mysql2.xml
Encoding:
Preserve Format:
Preserve Values:

sspence | Sat Jan 23 00:00 2010 Etc/UCT

XML Configuration Template Settings

Table 5 describes all the XML settings available when you create a generic or DTD-based XML configuration template. The list indicates if the setting is required or optional and whether or not it applies only to XML-DTD templates.

Table 5 XML and XML-DTD Template Settings

Attribute	Description
`ACM-FILENAME-KEY=<key>` Required; no default value.	`filename-key` identifies a path to the key in a value set that contains the name of the file being generated.
`ACM-FILENAME-DEFAULT=<filename >` Required; no default value.	`filename-default` identifies the default file name returned if there is no file name in the value set.
`ACM-NAMESPACE=<string>` Required; no default value.	`namespace` identifies a location where XML elements are stored in the database.
`ACM-TIMEOUT=<integer>` Optional; (default value is 0)	`timeout` represents the number of minutes that are added to the application configuration's total timeout. A valid timeout is any integer from 0-999 inclusive. The timeouts of all the configuration templates in an application configuration are added together and that number is added to the default timeout of ten minutes for configurations, which is the final timeout value for the entire configuration. Note that any pre- or post-installation scripts in the application configuration that run longer than ten minutes will time out and cancel the entire push job.
`ACM-DOCTYPE = <string>` Required; no default value. XML-DTD templates only.	`doctype` represents the name of the root element in an XML file. This is in the `DOCTYPE` tag at the beginning of the XML file.
`ACM-DOCTYPE-SYSTEM-ID = <string>` Required; no default value. XML-DTD templates only.	`system-id` represents the system ID of the DTD file that is the basis of the configuration template. This value is in the DOCTYPE tag at the beginning of the XML file.
`ACM-DOCTYPE-PUBLIC-ID = <string>` Required; no default value. XML-DTD templates only.	`public-id` represents the public ID of the XML file parsed with the configuration template. This value is in the DOCTYPE tag at the beginning of the XML file DTD options.

Table 5 XML and XML-DTD Template Settings (cont'd)

Attribute	Description
`ACM-ELEMENT=<element name>` Optional XML-DTD templates only.	`element` sets the element that the current options describe. This option defaults to whatever element or attribute comes before this section in the DTD file.
`ACM-ATTRIBUTE=<attribute name>` Optional XML-DTD templates only.	`attribute` sets the attribute that the current options describe. This option is ignored if no attribute is set. This attribute defaults to whatever element or attribute comes before this section in the file.
`ACM-PRINTABLE=<printable>` Optional XML-DTD templates only.	`printable` sets the printable value for the element or attribute in the SA Client. This value appears in the value set editor to the left of the field. This is usually set to something short and descriptive.
`ACM-DESCRIPTION=<description>` Optional XML-DTD templates only.	`description` sets the description for the current element or attribute to be displayed in the SA Client. This value displays when you mouse over the name or value fields in the value set editor. Use this to describe the purpose of the field in the value set editor as well as the valid values for this field.

5 XML Tutorial 1 - Create a Non-DTD XML Configuration Template

This tutorial shows how to create a configuration template for a non-DTD XML configuration file. It shows you how to create a configuration template using XML syntax, add it to an application configuration and then attach the application configuration to a managed server. Then you will import values from the `mysql.xml` configuration file on your managed server, make changes to some of those values, and push the new configuration file back to the managed server.

This tutorial is based on the Travel Manager example application described at Example: Travel Manager Application and XML Configuration File on page 61.

Sample Non-DTD XML File mysql.xml

Below is the contents of the XML configuration file for the travel manager application:

```
<?xml version="1.0" ?>
<db-config>
  <db-host>localhost</db-host>
  <db-name>wrightevents</db-name>
  <db-user>root</db-user>
  <db-password>hp-pass</db-password>
</db-config>
```

1. Create an XML Configuration Template

Create a configuration template based on the `mysql.xml` configuration file using the SA Client.

1 From the SA Client navigation pane, select Library and then select the By Type tab.

2 Open the Application Configuration node, then open the Templates node. This shows all the operating system groups.

3 Open an operating system node and select a specific operating system under one of the operating system nodes. For this example, select the operating system of one of your servers where you can install this application configuration.

4 From the **Actions** menu, select **New**.

5 In the Properties view, enter the following information:

- **Name**: TM-MySql
- **Description**: This is the template for the mysql.xml configuration file for the Travel Manager application.

- **Location**: You can leave the default location in the SA Library of /, or select another location to store your template file. Note that the customer setting of the folder containing the template must include the customer setting of the application configuration object. Otherwise the template will not be included in the list of available templates. For more information on folder settings, see "Folder Permissions" in the *SA Administration Guide*.
- **Version**: 0.1.
- **Type**: Template file
- **Parser Syntax**: XML Syntax
- **OS**: Select all the operating systems that the configuration template can be installed on.

6 Select **File ➤ Save**.

7 Keep the Template window open for the next task.

2. Add XML Settings

Since the XML configuration file `mysql.xml` provides most of the structural settings needed to parse the file's contents, an XML configuration template in SA only requires three pieces of information in an XML comment: `ACM-NAMESPACE`, `ACM-FILENAME-KEY` and `ACM-FILENAME-DEFAULT`.

1 Select the Content view in the navigation pane.

2 Copy and paste the following XML into the Content pane:

```
<!--
ACM-NAMESPACE = /TravelManager
ACM-FILENAME-KEY = /files/TravelManager
ACM-FILENAME-DEFAULT = /var/www/html/we/mysql.xml
-->
```

3 Select the Validate button to make sure the XML is valid.

4 Select the **File ➤ Save** menu to save your template.

5 Select the **File ➤ Close** menu.

These XML lines define the following:

- `ACM-NAMESPACE`: Specifies a unique name space which is required for each configuration template. In this example, since a name space for the Travel Manager application has already been established, you could reuse the root name space and append the service name. For example:

  `ACM-NAMESPACE = /TravelManager/web/mysql`

- `ACM-FILENAME-KEY`: Specifies a path to the key in the name space that stores the file name of the file being generated.
- `ACM-FILENAME-DEFAULT`: Specifies the path on the target server where the Travel Manger application's `mysql.xml` file is stored. This can be overridden for specific servers or groups of servers.

3. Create an Application Configuration to Contain the Template

In this step you create an application configuration object to contain your configuration template.

1 In the SA Client navigation pane, select Library and then select the By Type tab.

2 Open the Application Configuration node, then open the Configurations node. This shows all the operating system groups.

3 Open the operating system node and select the same operating system you used when you created the template in the previous steps. The OSs specified for the application configuration must be a subset of the OSs specified for the template.

4 From the **Actions** menu, select **New**.

5 In the Properties view of the File Configuration screen, specify the following properties:

- **Name**: Tm-MySql-Config
- **Description**: This is the application configuration for the mySQL configuration file for the Travel Manager application.
- **Location**: You can leave the default folder location in the SA Library of /, or select another folder to store your application configuration. Note that the customer setting of the folder containing the application configuration must include the customer setting of the managed servers where you intend to push the application configuration. For more information on folder settings, see “Folder Permissions” in the *SA Administration Guide*.
- **Version**: 0.1.
- **OS**: Select one or more operating systems of managed servers that the application configuration can be installed on.

6 Select the Content view in the navigation pane.

7 Select the add button “+” or the **Actions** ➤ **Add** menu to add the template.

8 In the Select Configuration Template screen, select the TM-MySql template and then select **OK**. This adds the template to the application configuration object.

9 Select **File** ➤ **Save** and then **File** ➤ **Close**. The application configuration and the configuration template inside of it are ready to be attached to a server where the configuration file is stored.

4. Attach the Application Configuration to a Managed Server.

Now that you have created the configuration template and application configuration object, you need to attach the application configuration to the server where the Travel Manager application is installed and specify the path to where the `mysql.xml` file is stored on the managed server.

To attach the application configuration to a server, perform the following steps:

1 From the SA Client navigation pane, select Devices, then select Servers ➤ All Managed Servers.

2 Locate a server where you can simulate installing the Travel Manager application configuration. The server's operating system must match one of the operating systems specified in the application configuration.

3 Select the server, and from the **Actions** menu, select **Open**.

4 In the server screen, select the Management Policies tab.

5 In the navigation pane, select Configured Applications. This displays the application configurations that are attached to the server.

6 Select the Installed Configurations tab.

7 From the **Actions** menu, select **Add Configuration**.

8 In the Select Application Configuration screen, select the Tm-MySql-Config application configuration.

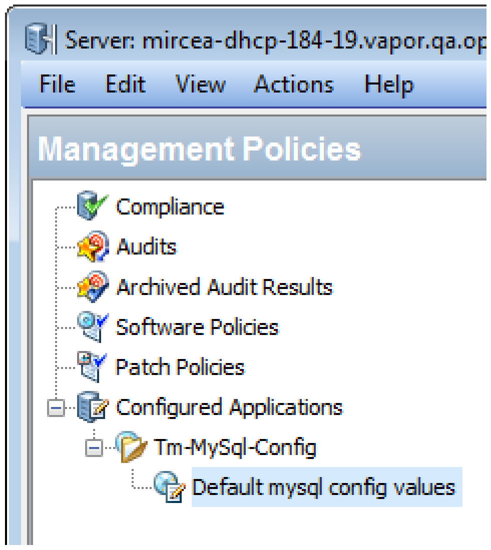
9 In the Instance Name field, enter "Default mysql config values". This creates a value set at the server instance level. For details, see Value Set Editor at the Server Level on page 46.

10 Select **OK**. The application configuration is now attached to the server.

11 Select the Save Changes button. Leave the server screen open for the next step.

Figure 12 below shows the Tm-MySql-Config application configuration attached to the server and the "Default mysql config values" value set. This value set is at the server instance level.

Figure 12 Application Configuration Attached to a Server, with the Value Set at the Server Instance Level Highlighted



Note that at this point, if the server you are adding the application configuration to has more than one instance of the `mysql.xml` configuration file because the server is hosting several instances of the application, you can right-click the "Default mysql config values" node of the configuration and select **Duplicate**. This creates another value set where you can set the file name path to point to the other instance of the application. For more information on the different levels of value sets, see Value Set Levels and Value Set Inheritance on page 36.

5. Configure Application Configuration Settings for the Server

Now that you have attached the application configuration to the managed server, you need to configure it for the server and set values for the configuration file.

➤ To import the values from the configuration file as described below, copy and paste the XML listed in Sample Non-DTD XML File mysql.xml on page 71 above into the target file `/var/www/html/we/mysql.xml` on your managed server. This will enable the import values step below.

1 Expand the Tm-MySql-Config node to show the value set at the server instance level. This value set is named “Default mysql config values”.

2 From the Contents pane, configure the following settings in the application configuration’s Value Set Editor:

- **Filename**: The original path and file name of the target XML file on the managed server is displayed to the right of the Filename field. This value is the same value for FILENAME-DEFAULT defined in the template. If this path name is acceptable for this server, you can leave this field empty. If you want the configuration file placed in a different location on this server, set the correct path to the target XML file on the target server in the Filename field.
- **Encoding**: Select the character encoding for the managed configuration file. The default encoding is the encoding used on the managed server. (Note that UTF-16 encoding is not supported.)
- **Preserve Format**: Select this option if you want to keep comments and preserve as much of the ordering and spacing of the original XML configuration file from the target server. SA will preserve as much of the target file as possible. For more information, see Setting Fields in the Value Set Editor on page 40.
- **Preserve Values**: To preserve the values contained in the actual configuration file on the server when no value is provided in a value set, select **Yes** for this option. With this option set to Yes, the target file’s values will be used unless overridden by values at any level of the inheritance hierarchy. If this option is set to No, and no value exists in the value set, no entry will be placed in the configuration file. For more information, see Setting Fields in the Value Set Editor on page 40.
- **Show Inherited Values**: Select this option to show the values in the value set and the inheritance level. When set to No, only the values set at the current inheritance level are displayed. When set to Yes, all values in the value set are displayed, those set at the current level and those that are inherited. This view is read-only.

3 Right-click inside the value set editor and select **Import Values**. Importing values will read the XML file on the managed server and copy the XML file’s contents to the value set at the server instance level.

4 To save changes, select the Save Changes button. Leave the server screen open for the next step.

6. Edit Values and Push the Configuration

The last steps are to edit values in the value set editor and then push the configuration to the server. When you push an application configuration, all the values in the value set replace the values in the configuration files on the target managed servers. Any scripts contained in the application configuration are executed based on what type they are. If the configuration file does not exist on the target server, the file is created when you perform the push.

To edit values and push the application configuration, perform the following steps:

1. Modify one or more values of the value set by editing in the Value column. For a description of the columns, see Meaning of Columns in the Value Set Editor on page 41.
2. After you have set values for the application configuration, select **Preview** to see the existing file on the server and the file that would be pushed to the server.
3. Select **Push** to copy the new application configuration to the server.
4. In the Push Configuration screen, select Start Job. Examine the status and results of the push job.

6 XML Tutorial 2 - Create an XML-DTD Configuration Template

This chapter shows how to create an XML-DTD configuration template to manage an XML configuration file that references a DTD, using the Travel Manager application as an example. For background on the Travel Manager example, see Example: Travel Manager Application and XML Configuration File on page 61.

You will first create the XML-DTD template as a text file and then import the text file into the SA Library and add it to an application configuration object. You will then attach the application configuration to a managed server. Finally you will edit some values in the application configuration and push those changes to the target XML file on the managed server.

Sample Travel Manager DTD-based XML File: mysql.xml

For the Travel Manager application, below is the `mysql.xml` XML configuration file. It is stored in `/var/www/html/we/mysql.xml.`

```
<?xml version="1.0"?>
<!DOCTYPE db-config PUBLIC "-//Williams Events//Travel Manager//EN"
"mysql.dtd">
<db-config>
<db-host>localhost</db-host>
<db-name>wrightevents</db-name>
<db-user>root</db-user>
<db-password>hp-pass</db-password>
</db-config>
```

Sample Travel Manager XML DTD File: mysql.dtd

For the Travel Manager application, below is the accompanying `mysql.dtd` DTD. It is stored in `/var/www/html/we/mysql.dtd.`

```
<!ELEMENT db-config (db-host,db-name,db-user,db-password)>
<!ELEMENT db-host     (#PCDATA)>
<!ELEMENT db-name     (#PCDATA)>
<!ELEMENT db-user     (#PCDATA)>
<!ELEMENT db-password (#PCDATA)>
```

1. Create XML-DTD Template in a Text Editor

In this task, you will create the source for the XML-DTD configuration template using a text editor. To create an XML-DTD configuration template in a text editor, perform the following steps:

1 In a text editor, enter the following information:

```
<!--
ACM-TIMEOUT = 1
ACM-FILENAME-KEY = /files/TravelManager
ACM-FILENAME-DEFAULT = /var/www/html/we/mysql.xml
ACM-NAMESPACE = /TravelManager/
ACM-DOCTYPE = db-config
ACM-DOCTYPE-SYSTEM-ID = mysql.dtd
ACM-DOCTYPE-PUBLIC-ID = -//Williams Events//Travel Manager//EN
-->
```

This information is required (except ACM-TIMEOUT) and is used by the application configuration parser to read both the XML-DTD and the XML file you want to manage:

- `ACM-TIMEOUT`: (Optional) Represents the number of minutes that are added onto the configuration template's default timeout value (ten minutes) during a push.
- `ACM-FILENAME-KEY`: Defines the location in name space where the `mysql.xml` filename will stored.
- `ACM-FILENAME-DEFAULT`: Defines the default location (absolute path) of the `mysql.xml` file on the managed server.
- `ACM-NAMESPACE`: This value defines the location where values read from the `mysql.xml` file on the managed server will be stored in the database. This name space must be unique, and the path must start with a forward slash (/).
- `ACM-DOCTYPE`: Defines the name of the root element in the XML file. The root element follows the opening `<!DOCTYPE` declaration found in the target XML configuration file.
- `ACM-DOCTYPE-SYSTEM-ID`: Defines the name of associated DTD file on the managed server. This value can typically be found in the XML configuration file as the SYSTEM attribute in the DOCTYPE element.
- `ACM-DOCTYPE-PUBLIC-ID`: Defines a string that represents a public identifier of the XML document. This value can typically be found in the XML configuration file as the PUBLIC-ID attribute of a DOCTYPE element.

2 Save the file, giving it the name `mysql-dtd.tpl`. Keep the file open for the next task.

2. Add Custom Settings for Element Descriptions in the Value Set Editor

In this task you will add some extra information to the XML-DTD template file that allows you to customize the display of each element from the target XML file as seen in the value set editor in the SA Client.

There are two optional settings you can add to your XML-DTD configuration template that allow you to customize how elements from the target XML-DTD configuration file are displayed in the value set editor in the SA Client:

- `ACM-PRINTABLE`: Defines a label for each element from the XML file that will be displayed in the value set editor when the XML-DTD template is shown in the SA Client.
- `ACM-DESCRIPTION`: Defines mouse-over text when a user moves a mouse pointer over the field defined in `ACM-PRINTABLE` in the value set editor in the SA Client.

See Figure 13 below for an example of how these elements appear in the value set editor in the SA Client.

This example uses the explicit method for placing these custom settings inside the XML-DTD template. For more information on this method of placing custom settings, see Explicit versus Positional Display Settings on page 65.

To add custom settings in the XML-DTD template, perform the following steps:

1 In the text editor with the `mysql-dtd.tpl` file, add the following information for each XML element being referenced by the DTD. For example, after the main information in the template, add a list of each element contained in the source XML file and then for each element, make an XML comment using these three ACM setting tags:

   - `ACM-ELEMENT`: Declares the element from the XML file that the following the `ACM-PRINTABLE` and `ACM-DESCRIPTION` settings will describe. This option defaults to whatever element or attribute came before this section in the DTD file.
   - `ACM-PRINTABLE`: Set a short, descriptive label for the element when it is displayed in the value set editor.
   - `ACM-DESCRIPTION`: Set mouse-over text for the element.

   The example XML-DTD template file should look like this:

```
ACM-TIMEOUT = 1
ACM-FILENAME-KEY = /files/TravelManager
ACM-FILENAME-DEFAULT = /var/www/html/we/mysql.xml
ACM-NAMESPACE = /TravelManager/
ACM-DOCTYPE = db-config
ACM-DOCTYPE-SYSTEM-ID = mysql.dtd
ACM-DOCTYPE-PUBLIC-ID = -//Williams Events//Travel Manager//EN
-->
<!ELEMENT db-config (db-host,db-name,db-user,db-password)>
<!ELEMENT db-host     (#PCDATA)>
<!ELEMENT db-name     (#PCDATA)>
<!ELEMENT db-user     (#PCDATA)>
<!ELEMENT db-password (#PCDATA)>
<!--
ACM-ELEMENT = db-config
ACM-PRINTABLE = database configuration
ACM-DESCRIPTION = The db-config element specifies the data structure that
contains the information needed to connect to a database.
-->
<!--
ACM-ELEMENT = db-host
ACM-PRINTABLE = database hostname
ACM-DESCRIPTION = The db-host element specifies the name of the host
computer (the server) on which the database engine is running.
```

```
-->
<!--
ACM-ELEMENT = db-name
ACM-PRINTABLE = database name
ACM-DESCRIPTION = The db-name element specifies the name of the database.
-->
<!--
ACM-ELEMENT = db-user
ACM-PRINTABLE = database user
ACM-DESCRIPTION = The db-user element specifies the user identification
used to connect to the database.
-->
<!--
ACM-ELEMENT = db-password
ACM-PRINTABLE = database passsword
ACM-DESCRIPTION = The db-password element specifies the password used to
connect to the database.
 -->
```

2 Save and close the file.

3. Import the XML-DTD Configuration File

In this task you will import your template file and create a new configuration template that will manage the target XML and DTD files. To import the XML-DTD configuration file into the SA Library, perform the following steps:

1 From the SA Client navigation pane, select Library and then select the By Type tab.

2 Locate and open the Application Configuration node. Open the Templates node. Open an operating system group and navigate to the operating system that the template file applies to. Note that a template can apply to multiple operating systems. For example, you could select one of the Red Hat operating system versions.

3 From the **Actions** menu, select **Import Template**.

4 Navigate to the file you created in the previous step and select it. Set the encoding if it is not the default.

5 Select Open. This imports your template file and displays it in the Templates screen.

6 Select the Properties view and enter the following information:

- **Name:** mysql-dtd.tpl
- **Description**: This is the template for the mysql.dtd (mysql.xml) file for the Travel Manager application.
- **Location**: Specify where in the SA Library you want to store the template.
- **Version**: 0.1.
- **Type**: Template file
- **Parser Syntax**: XML DTD Syntax.
- **OS**: Select the appropriate operating system.

7 Select the Contents view to display the contents of the template file you just imported.

8 Select the **Validate** button to confirm that the syntax is valid before proceeding.

9 Select **File ➤ Save**.

10 Select **File ➤ Close.**

4. Create an Application Configuration Object

An application configuration is a container for configuration template files. In this step you will create an application configuration and import the template.

1 From the SA Client navigation pane, select Library and then select the By Type tab.

2 Locate and open the Application Configuration node. Open the Configurations node. Open the operating system group and select the operating system that the application configuration applies to. Note that an Application Configuration can apply to multiple operating systems. You can change this in a later step.

3 Select the **Actions ➤ New** menu. This displays the File Configuration screen where you can specify the properties and contents of the Application Configuration.

4 In the Properties view, specify the following:

- **Name:** TM-mysql-dtd
- **Description**: This is the application configuration for the mysql.xml and mysql.dtd files for the Travel Manager application.
- **Location:** Specify where in the SA Library you want to store the Application Configuration.
- **Version**: 0.1
- **OS**: Select the appropriate operating system.

5 In the Content view, select the **Actions ➤ Add** menu or the "+" button to add a template to the Application Configuration.

6 In the Select Configuration Template screen, select the mysql-dtd.tpl template file.

7 Select **OK**.

8 Select **File ➤ Save** to save your Application Configuration.

9 Select **File ➤ Close.**

5. Attach the Application Configuration to a Managed Server

In this task you will to attach the application configuration to the server where the Travel Manager application is installed, and then enter the path name to the `mysql.dtd` configuration file. To attach the application configuration to a server, perform the following steps.

1 From the SA Client navigation pane, select Devices ➤ Servers ➤ All Managed Servers.

2 Select a server, and from the **Actions** menu, select **Open**. Make sure the operating system of the server you select matches the operating system specified on the application configuration and the template.

3 Select the Management Policies tab.

4 Select the Configured Applications node.

5 Select the Installed Configurations tab.

6 From the **Actions** menu, select **Add Configuration**.

7 In the Select Application Configuration screen, select the application configuration TM-mysql-dtd.

8 In the Instance Name field, enter "Value set 1 for mysql.xml".

9 Select **OK**. The application configuration is attached to the server.

10 Select the **Save Changes** button.

11 Leave the application configuration screen open for the next step.

6. Import Values from the Configuration File

The next step is to set values in a value set. While you can set values in a value set manually, the easiest way is to import the values from an existing configuration file. In this step you will import the values from a configuration file on the server.

To import the values from the configuration file as described below, copy and paste the XML listed in Sample Travel Manager DTD-based XML File: mysql.xml on page 77 above into the target file `/var/www/html/we/mysql.xml` on your managed server. Copy and paste the DTD listed in Sample Travel Manager XML DTD File: mysql.dtd on page 77 above into the target file `/var/www/html/we/mysql.dtd`. This will enable the import step below.

1 In the server Management Policies, open the Configured Applications node. This displays the application configuration attached to the server.

2 Open the TM-mysql-dtd node. This displays the server instance value sets, which are the node under the TM-mysql-dtd node.

3 Select the "Value set 1 for mysql.xml" node. This is the server instance value set for the mysql-dtd.tpl configuration template.

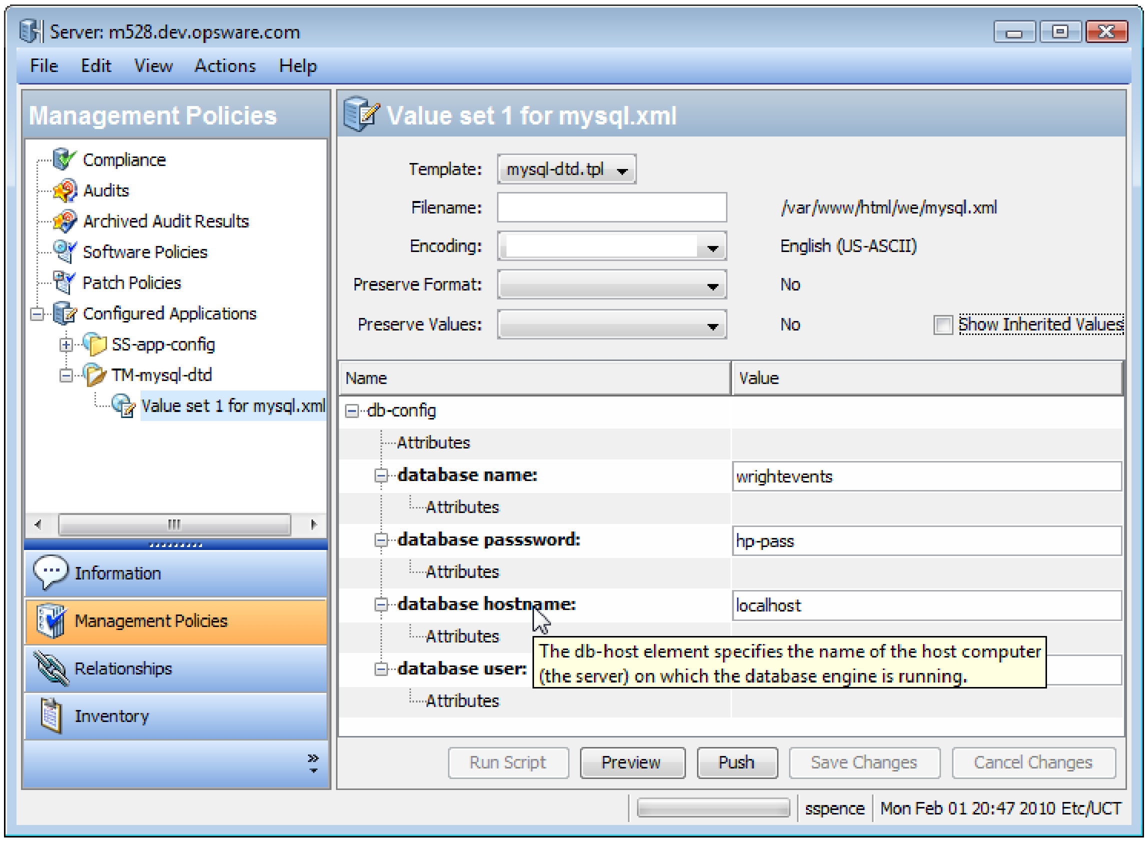
4 Right-click on any value under the Value column and select the **Import Values** menu.

5 In the Confirmation Dialog, select Yes. This imports the values from the file `/var/www/html/we/mysql.xml` into the value set at the server instance level.

6 Select the Save Changes button.

Figure 13 below shows the XML-DTD template with the server instance value set and the mouse-over text displayed from the ACM-DESCRIPTION element.

Figure 13 Value Set and Mouse-Over Text for XML-DTD Configuration Template



Template: mysql-dtd.tpl

Filename: /var/www/html/we/mysql.xml

Encoding: English (US-ASCII)

Preserve Format: No

Preserve Values: No

Name	Value
db-config	
Attributes	
database name:	wrightevents
Attributes	
database passsword:	hp-pass
Attributes	
database hostname:	localhost
Attributes	
database user:	
Attributes	

The db-host element specifies the name of the host computer (the server) on which the database engine is running.

sspence | Mon Feb 01 20:47 2010 Etc/UCT

7 Leave the application configuration displayed for the next step.

7. Edit Values and Push the Configuration

The last step is to edit values in the value set editor and then push the configuration to the server. When you push an application configuration, all the values in the value set replace the values in the configuration files on the target managed servers. All scripts in the application configuration are also executed. If the configuration file does not exist on the target server, the file is created as part of the push.

To edit values and push the application configuration, perform the following steps:

1 Make sure that the server instance level value set is displayed by selecting the “value set 1 for mysql.xml” node in the navigation pane, as shown in Figure 13.

2 Modify the password value under the Value column.

3 Select the Save Changes button.

4 Select the Preview button to display the difference between the existing configuration file on the server and the configuration file that will be pushed to the server.

5 After examining the comparison screen, select the Close button.

6 Select the Push button. This displays the Push Configurations wizard.

7 Select the Start Job button. This starts the push operation.

8. Examine the Job Status of the push job. Select any step to display details on that step.
9. When the job completes, select the Close button.
10. You can log on to the server and examine the `mysql.xml` configuration file to verify that it was updated on the server.

7 CML Tutorial 1 - Create an Application Configuration for a Simple Web App Server

This chapter demonstrates how to set up and manage a simple configuration file for a Web Application Server running on two servers. Each server runs the Web Application Server and needs to be configured separately. This tutorial shows how to create an application configuration, a configuration template, value sets and two instances of the application configuration, one for each server. Finally it shows how to push the application configuration to each server.

1. Determine the configuration files to be managed

The web application server uses one configuration file named WASconfig.txt. This file is located in the directory /opt/WAS/WASconfig.txt. The contents of this file are as follows:

```
size=1000
dir=/tmp/WAS_001
primary=yes
```

2. Create a template for the configuration file

You can create a template in either of two ways:

- Create a template in a text file and import the text file into the SA Library.
- Create a template directly in the SA Library.

Both methods are described below. Choose one method and follow the steps.

Create a template file and import it into SA

1 In a text editor, copy the configuration file into an empty file:

```
size=1000
dir=/tmp/WAS_001
primary=yes
```

2 Create a template that models this file using CML. First add a comment block and the required CML metadata defining the name space and the target configuration file name.

— The name space defines the key where information for this template will be stored in the database.

— The file name key defines where the default file name will be stored in the database.

— The default file name specifies the name that will be used for the resulting configuration file.

```
@########################################################
# /opt/WAS/WASconfig.txt                                #
# Version 1.0                                           #
# Author <name>                                         #
########################################################@
@!namespace=/WAS-server-namespace/@
@!filename-key="/WAS-server"@
@!filename-default="/opt/WAS/WASconfig.txt"@
size=1000
dir=/tmp/WAS_01
primary=yes
```

3 Next change the variable parts of the configuration file to variables using CML tags:

```
@########################################################
# /opt/WAS/WASconfig.txt                                #
# Version 1.0                                           #
# Author <name>                                         #
########################################################@
@!namespace=WAS-server-namespace/@
@!filename-key="/WAS-server"@
@!filename-default="/opt/WAS/WASconfig.txt"@
size=@value_of_size;int@
dir=@value_of_dir;string@
primary=@value_of_primary;boolean@
```

4 Save the file with an extension of ".tpl", for example WASconfig_txt.tpl

5 Import the template file into the SA Library. Follow the steps at Importing and Validating a Template File on page 18.

Create a template file directly in SA

1 In the SA Client, select the Library tab.

2 Select the By Type tab.

3 Open the Application Configurations node and the Templates node. Navigate to the OS family and the OS version where the application runs. For this example, select Red Hat Enterprise Linux AS 4.

4 Select Actions ➤ New. This displays the Templates screen.

5 Enter the template name "WASconfig_txt.tpl" and a brief description. Select the location in the SA Library to store the template file. Set the version string. Set the Type to "Template file". Set the Parser Syntax to "CML Syntax".

6 Select the Content view to display a text editor.

7 Type or paste in the CML text. This is the same CML text as shown above.

8 Select Actions ➤ Validate to check the syntax of your CML. Make any needed corrections.

9 Select File ➤ Save to save your template.

10 Close the template screen.

3. Create an application configuration object

Create an application configuration object to contain the configuration template.

1 In the SA Client, select the Library tab.

2 Select the By Type tab.

3 Open the Application Configurations node and the Configurations node. Navigate to the OS family and the OS version where the application runs. For this example, select Red Hat Enterprise Linux AS 4.

4 Select Actions ➤ New. This displays the Configuration screen.

5 Enter the name “WAS-app-config”, a brief description and version string of your application configuration. Select the location in the SA Library to store the application configuration.

6 Select File ➤ Save to save your application configuration.

4. Add the template file to the application configuration object

1 Open the “WAS-app-config” application configuration object you created in the previous step.

2 Select the Configuration Values view.

3 Select the “+” button or select Actions ➤ Add. This displays the Select Configuration Template screen.

4 Select your “WASconfig_txt.tpl” template file and select OK.

5 Select File ➤ Save to save your changes to the application configuration object.

6 Select File ➤ Close to close the application configuration object.

5. Attach the application configuration object to servers

Two servers are running the web application server, RHEL001 and RHEL008. RHEL001 is the primary server and RHEL008 is the secondary server. Create two instances of the application configuration by attaching the application configuration object to these two servers as follows:

1 Locate the primary server RHEL001 in the SA Client.

2 Select the RHEL001 server and select Actions ➤ Open.

3 Select the Management Policies tab.

4 Select the Configured Applications node.

5 Select the Actions ➤ Add Configuration menu.

6 Select the “WAS-app-config” application configuration object.

7 In the Instance Name field, enter “Primary Instance of WAS-app-config” and select OK.

8 Select Save Changes. This creates an instance of the application configuration for the server RHEL001.

9 Repeat the above steps with the secondary server RHEL008 except in the Instance Name field, enter “Secondary Instance of WAS-app-config” and select OK. This creates a second instance of the application configuration for the server RHEL008.

6. Set default values

The required values for the configuration files for the two servers are shown below:

Table 6 Configuration Values for Two Servers Running the Web Application Server

Server: RHEL001	Server: RHEL008

`size=1000`
`dir=/tmp/WAS_001`
`primary=yes`
	`size=1000`
	`dir=/tmp/WAS_008`
	`primary=no`

You can set default values for the configuration file that can be inherited by the individual servers or that can be overridden by each individual server. If an individual server does not override the default value, it uses the inherited default value.

The following table shows which values will be set with default values and which will be set by individual servers:

Table 7 Application Level Default Values for the Configuration File for the Web Application Server

Default Value	Description
`size=1000`	Set this to a default value of 1000 at the application level. All servers attached to this application configuration will use this value unless they override it.
`dir`	Do not set this to a default value. Each server will set this value at the server level or at the server instance level.
`primary=no`	Set this to a default value of “no” at the application level. All servers attached to this application configuration will use this value unless they override it.

Set Application Level Default Values

To **set the application level default values**, perform the following steps:

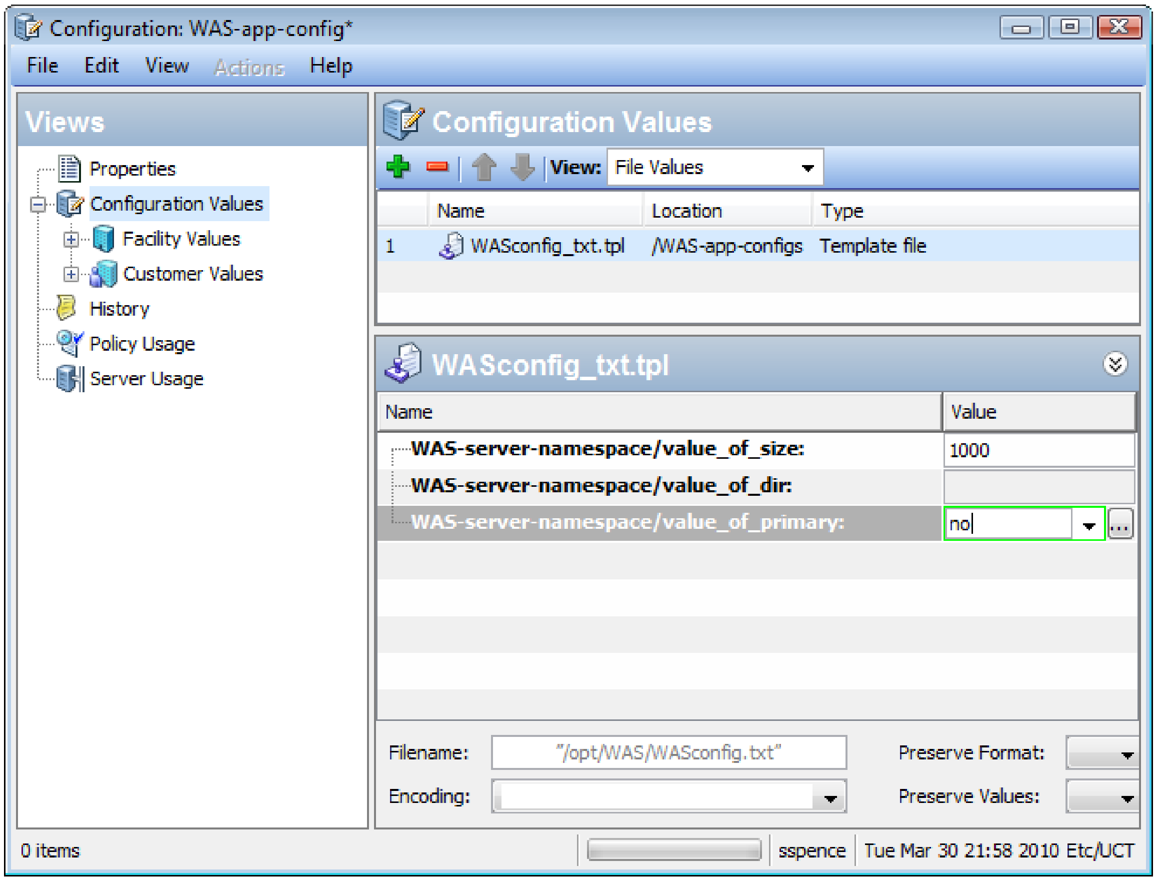
1 Open the application configuration object you created above.

2 Select the Configuration Values view.

3 Select the WAS-config-template.tpl template file.

4 Select File Values in the view drop-down list. This displays the default values for the template at the application level.

5 Set “value_of_size” to 1000 and “value_of_primary” to “no”, as shown below. Do not set a default value for “value_of_dir” because each server will need to set this value.



	Name	Location	Type
1	WASconfig_txt.tpl	/WAS-app-configs	Template file

Name	Value
WAS-server-namespace/value_of_size:	1000
WAS-server-namespace/value_of_dir:	
WAS-server-namespace/value_of_primary:	no

Filename: "/opt/WAS/WASconfig.txt"
Encoding:
Preserve Format:
Preserve Values:

6 Select File ➤ Save to save your **application level default values**.

7 Select File ➤ Close.

Set Server Level Default Values for RHEL001

The server RHEL001 needs to set `dir=/tmp/WAS_001` and `primary=yes` at the server level. It does not need to set size because it can use the value set at the application level.

To **set the server level default values** for the primary server RHEL001, perform the following steps.

1 Locate in the SA Client the RHEL001 server.

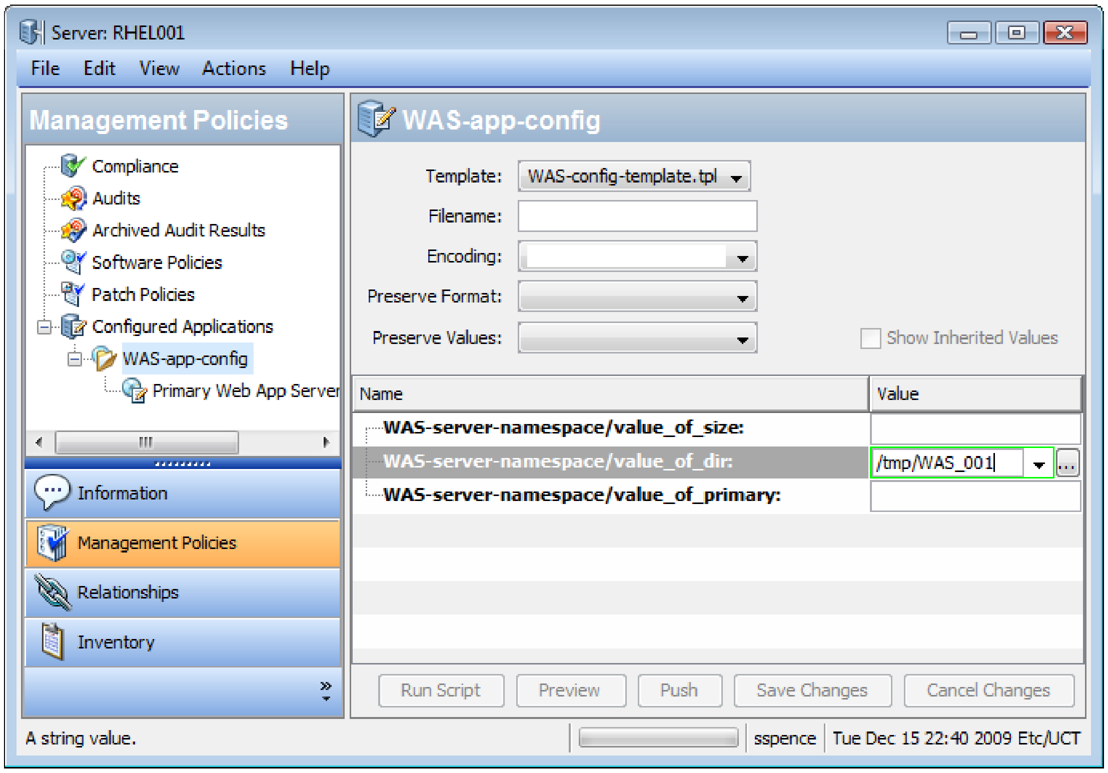
2 Select the RHEL001 server and select Actions ➤ Open.

3 Select the Management Policies tab.

4 Open the Configured Applications node to reveal the “WAS-app-config” application configuration object.

5 Select the “WAS-app-config” application configuration object attached to this server. This displays the default values set at the server level. Values set at the server level apply to all instances of the application configuration on the server unless overridden at the server instance level.

6 Set the server level default value for “value_of_dir” to “/tmp/WAS_001”, as shown below. Do not set a default value for “value_of_size” or “value_of_primary” because these values will be inherited from the application level.



Template: WAS-config-template.tpl
Filename:
Encoding:
Preserve Format:
Preserve Values:

Name	Value
WAS-server-namespace/value_of_size:	
WAS-server-namespace/value_of_dir:	/tmp/WAS_001
WAS-server-namespace/value_of_primary:	

A string value. sspence Tue Dec 15 22:40 2009 Etc/UCT

7 Select the Save Changes button or the File ➤ Save menu to save your **server level default values**.

8 Open the WAS-app-config node to reveal the application configuration instance “Primary Instance of WAS-app-config”.

9 Select the instance “Primary Instance of WAS-app-config”. This displays the **instance level default values**. This is the lowest value set level and overrides all other levels. Notice that no values have been defined at the instance level.

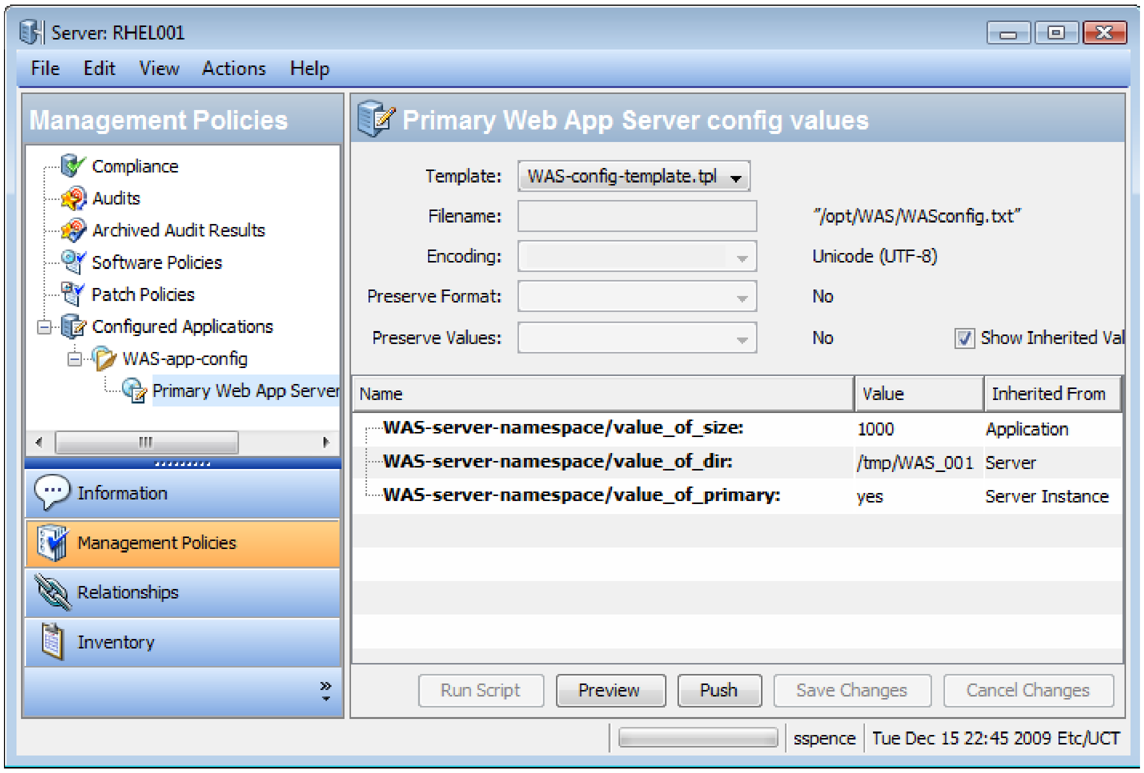
10 Select “Show Inherited Values” to show the values that will be inherited from the application level defaults and the server level defaults. Notice that “value_of_size” and “value_of_primary” are inherited from the application level and “value_of_dir” is inherited from the server level.

11 Uncheck “Show Inherited Values” so you can set the instance level default values.

12 Set “value_of_primary” to “yes”.

13 Select the Save Changes button or the File ➤ Save menu to save your **instance level default values**.

14 Select “Show Inherited Values” again to show the values that are inherited from the application level, the server level and the instance level. Notice that “value_of_size” is inherited from the application level, “value_of_dir” is inherited from the server level and “value_of_primary” is inherited from the instance level, as shown below.



Template: WAS-config-template.tpl
Filename: "/opt/WAS/WASconfig.txt"
Encoding: Unicode (UTF-8)
Preserve Format: No
Preserve Values: No
☑ Show Inherited Val

Name	Value	Inherited From
WAS-server-namespace/value_of_size:	1000	Application
WAS-server-namespace/value_of_dir:	/tmp/WAS_001	Server
WAS-server-namespace/value_of_primary:	yes	Server Instance

sspence | Tue Dec 15 22:45 2009 Etc/UCT

Set Server Level Default Values for RHEL008

The server RHEL008 needs to set `dir=/tmp/WAS_008` at the server level. It does not need to set size or primary because it can use the values set at the application level.

To **set the server level default values** for the secondary server RHEL008, perform the following steps.

1 Locate in the SA Client the RHEL008 server.

2 Select the RHEL008 server and select Actions ➤ Open.

3 Select the Management Policies tab.

4 Open the Configured Applications node to reveal the “WAS-app-config” application configuration object.

5 Select the “WAS-app-config” application configuration object attached to this server. This displays the default values set at the server level. Values set at the server level apply to all instances of the application configuration on the server unless overridden at the server instance level.

6 Set the server level default value for “value_of_dir” to “/tmp/WAS_008”. Do not set a default value for “value_of_size” or “value_of_primary” because these values will be inherited from the application level.

7 Select the Save Changes button or the File ➤ Save menu to save your **server level default values**.

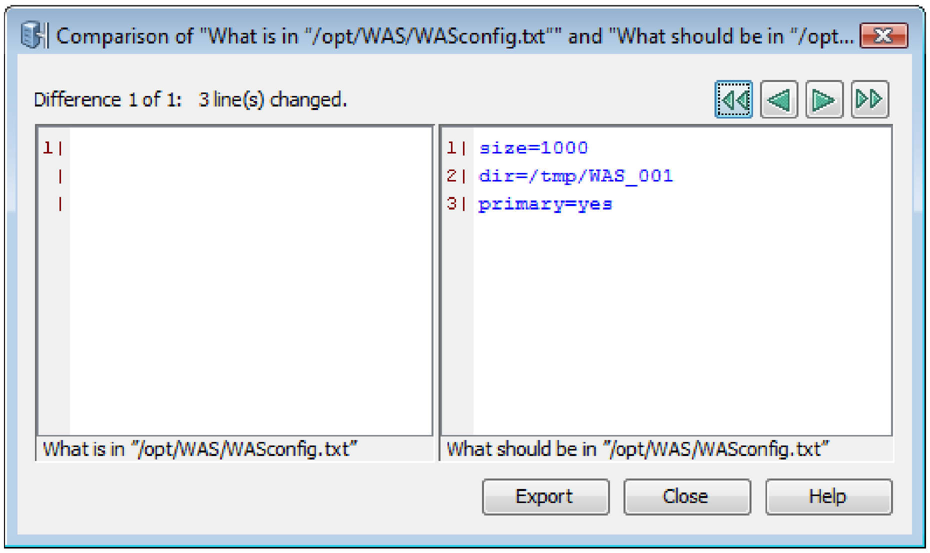
8 Open the WAS-app-config node to reveal the application configuration instance “Secondary Instance of WAS-app-config”.

9 Select the instance “Secondary Instance of WAS-app-config”. This displays the **instance level default values**. This is the lowest value set level and overrides all other levels. Notice that no values have been defined at the instance level.

10 Select “Show Inherited Values” to show the values that will be inherited from the application level defaults and the server level defaults. Notice that “value_of_size” and “value_of_primary” are inherited from the application level and “value_of_dir” is inherited from the server level.

7. Compare the actual configuration files with the configuration template

You can optionally compare the values specified in the application configuration to the actual values in the configuration file on the server by selecting the Preview button from the server screen. The following shows the comparison on RHEL001 when there is no configuration file on the server yet:


Comparison of "What is in "/opt/WAS/WASconfig.txt"" and "What should be in "/opt...

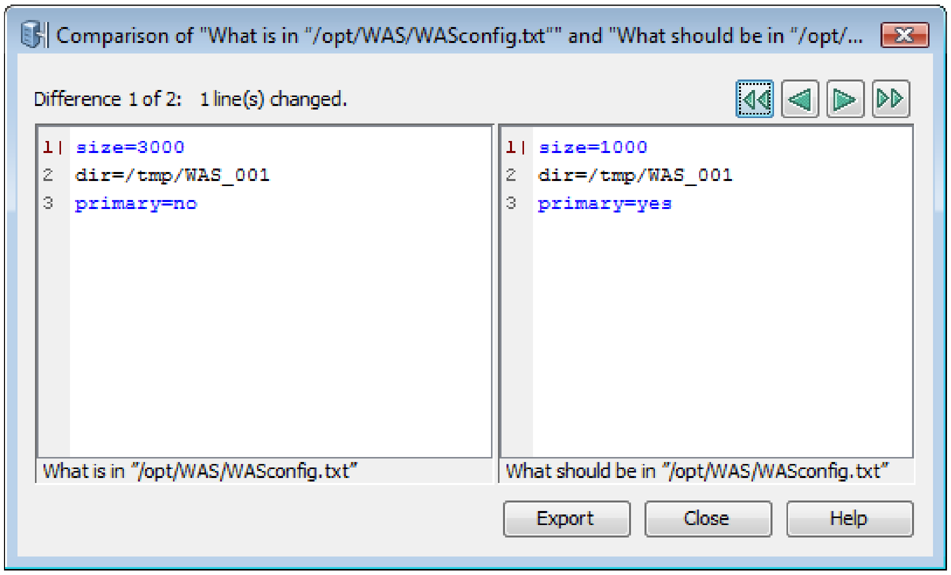
Difference 1 of 1: 3 line(s) changed.

```
size=1000
dir=/tmp/WAS_001
primary=yes
```

What is in "/opt/WAS/WASconfig.txt"

What should be in "/opt/WAS/WASconfig.txt"

The following shows the comparison when there is an existing configuration file on the server that differs from the values specified in the application configuration:


Comparison of "What is in "/opt/WAS/WASconfig.txt"" and "What should be in "/opt/...
Export
Close
Help

Difference 1 of 2: 1 line(s) changed.

What is in "/opt/WAS/WASconfig.txt"

```
size=3000
dir=/tmp/WAS_001
primary=no
```

What should be in "/opt/WAS/WASconfig.txt"

```
size=1000
dir=/tmp/WAS_001
primary=yes
```

8. Push configuration changes to the server

To push configuration changes to a server, perform the following steps:

1 Locate in the SA Client the RHEL001 server.

2 Select the RHEL001 server and select Actions ➤ Open.

3 Select the Management Policies tab.

4 Open the Configured Applications node to reveal the “WAS-app-config” application configuration object.

5 Open the “WAS-app-config” application configuration node to reveal the “Primary Instance of WAS-app-config” instance.

6 Select the “Primary Instance of WAS-appconfig” instance.

7 Select the Push button.

8 You can select the Start Job button to accept the job defaults for scheduling and notifications, or select Next.

9 In the Scheduling screen you can specify when you want the push configurations job to run. Select Next.

10 In the Notifications screen you can specify one or more people to receive an email message when the job succeeds or fails. You can also specify a ticket identifier. Select Next.

11 Select Start Job. SA generates the configuration file from the template and value set, pushes the resulting configuration file to the server and displays the results.

12 Select Close.

For a tutorial showing a more complex configuration file, see CML Tutorial 2 - Create a Template of a Web Server Configuration File on page 95.

8 CML Tutorial 2 - Create a Template of a Web Server Configuration File

This tutorial explains how to use the **Configuration Modeling Language** (CML) to make a configuration template based upon the Microsoft Internet Information Services (IIS) Web server configuration file named UrlScan.ini. You will use the CML language to create a template file based on this file so it can be managed on a managed server.

While this tutorial will not teach you everything about CML, creating a CML template from UrlScan.ini will help you gain a fundamental understanding of CML and the process of creating a configuration template from a configuration file.

A sample of the UrlScan.ini file is listed at Sample UrlScan.ini File on page 109. The complete CML file is listed at Complete url_scan_ini.tpl CML Template on page 113.

To complete this tutorial you should have the following.

- Documentation for UrlScan.ini. This is available with the Microsoft IIS documentation.
- The UrlScan.ini file.
- A text editor for creating a CML file.

1. Analyze the Native Configuration File and Documentation

Once you have identified an application configuration file you want to manage, the first thing to do is to analyze the native configuration file and its documentation. Make sure that you understand the purpose of the configuration file, all the elements in the file and the kinds of data the configuration file manages.

This tutorial uses the UrlScan.ini file. A sample is listed at Sample UrlScan.ini File on page 109. The file UrlScan.ini enables systems administrators to configure Microsoft Internet Information Services (IIS) web server. The UrlScan.ini file consists of several sections, such as [Options], [AllowVerbs], [DenyVerbs], [DenyHeaders], [AllowExtensions], and [DenyExtensions]. Each section allows the IIS administrator to set different configurations to either allow or disallow certain kinds of HTTP requests on the IIS server. These sections do not need to be in any specific order. However, the information inside each section must be ordered. For example, the [AllowVerbs] section must be followed by specific HTTP requests that are allowed to access the web site.

UrlScan.ini contains lists of strings, such as lists of verbs and file extensions, and options that take a Boolean value of “1” for True or “2” for False.

2. Create a CML Comment Block

A CML template is a simple text file named with the .tpl file extension. Use a text editor to create a new text file named Url_Scan_ini.tpl. The .tpl extension is the typical (but optional) file extension used by SA for CML templates.

Create a CML comment block at the top of the file with information about the template as follows:

```
@###############################################
# \system32\inetsrv\urlscan.ini (Windows)     #
# Version 1.0                                 #
# Joe Author (joe_author@your_company.com)    #
###############################################@
```

The CML comment tag uses the following syntax:

```
@# <one line comment>
```

Or

```
@## <comments spanning multiple lines>
    <comments spanning multiple lines> #@
```

3. Create CML Setup Instructions

The setup section instructs the parser how to interpret the CML file. The `namespace`, `filename-key` and `filename-default` instructions are required for all CML files. The other instructions shown below are optional. These instructions define white space handling, Boolean values, comment formats and ordering rules.

To create the basic setup section, enter the following information after the comment block in the CML template:

```
@!namespace=/security/@
@!filename-key="/test";filename-default="/c/UrlScan.ini"@
@!optional-whitespace@
@!boolean-yes-format="1";boolean-no-format="0"@
@!line-comment-is-semicolon@
@!unordered-lines@
```

Notice that each CML instruction tag starts with the characters “`@!`” and ends with the character “`@`”.

The following line combines two CML instructions on one line:

```
@!filename-key="/test";filename-default="/c/UrlScan.ini"@
```

This line could also be written as two separate lines as follows:

```
@!filename-key="/test"@
@!filename-default="/c/UrlScan.ini"@
```

Setup Instructions

Table 8 explains each setup instruction

Table 8 CML Template Setup Instructions

CML Tag	Description
`@!namespace=/security/@`	The `@!namespace` instruction defines the name space that will be used by this CML template. This defines where in the database the values used by this CML template will be stored. Each CML template should use its own name space so names do not collide with other template. In this example, the name space is `/security.` All value sets will be stored in this name space.
`@!filename-key="/files/urlscan_ini";filename-default="/c/urlscan.ini"@`	The `@!filename-key` instruction defines the location in the name space where the file name will stored. If the value starts with a "/", it defines a separate name space. If the value does not start with a "/", it will be appended to the name space defined by the `@!namespace` instruction. In this example, the default file name will be stored in the database under the name space "`/file/urlscan-ini`". The `@!filename-default` instruction defines the default path where the native configuration file will be saved on the server. This path can be changed using the SA Client. In this example, when the configuration file is pushed to a managed server, it will be placed in `/c/urlscan.ini`. Note that the path names use only forward slashes.

`@!optional-whitespace@`	This instruction indicates that white space is optional between items in the configuration file. For example, either of the following entries would be valid if this option is set:
	
	`Key = "value"`
	
	`Key="value"`

`@!boolean-yes-format="1";boolean-no-format="0"@`	This instruction defines the allowable Boolean values in the configuration file. In this case, true is indicated with the character `1`, and false is indicated with a `0`. Any other values for Booleans will not be allowed.
`@!line-comment-is-semicolon@`	Instructs the parser to ignore anything that follows a semicolon in the configuration file. This allows comments in the native configuration file using the semicolon before each comment.

Table 8 CML Template Setup Instructions (cont'd)

CML Tag	Description
`@!unordered-lines@`	Instructs the parser that the sections in the configuration file can be in any order. If you used `ordered-lines`, then the configuration file would have to conform to the order of the template.

4. Define the [Options] Section — Opening Blocks

Now you are ready to add CML instructions to the template. The first section of the UrlScan.ini file you will model in CML is the [Options] section, which contains several options for the configuration file.

In CML, if a section of information in a configuration file has more than one kind of data (data that needs to be read differently by the CML parser), you can open "blocks" to handle each section of information separately. Typically, you open a block in CML to define special parser rules for a section of the CML file. The [Options] section has two "blocks" of information: the title of the section which is just "[Options]" and all the options in that section. Since these blocks belong together, you will set them at different levels, the first block (the title of the section) at level one, and the second block (the contents of the section) at level two. Nesting the blocks in this manner keeps the sections within the block together when read by the parser.

1 To define the [Options] section, enter the following lines:

```
@1[;optional;ordered-lines@
[Options]
@2[;unordered-lines@
```

2 In the UrlScan.ini file the [Options] section contains a list of key-value pairs. Use the block tag ([ ) set at two levels because there are two kinds of data in this section: a heading and a list of key-value pairs. The first level block handles the text string "[Options]" while the second level block handles all of the key-value pairs in that section.

Table 9 explains how to open two block levels for the [Options] section.

Table 9 Marking Up the Start of the [Options] Section

CML Tag	Description

`@1[;optional;ordered-lines@`	The number 1 sets the first level of the multiline block.
	
	`[`
	The square bracket opens a new block.
	
	`optional`
	Indicates that this entire block is optional in the configuration file.
	
	`ordered-lines`
	Indicates that whatever follows this tag (the string [Options]) must come first in the native UrlScan.ini configuration file. In other words, the title [Options] must appear before the actual options.

Table 9 Marking Up the Start of the [Options] Section (cont'd)

CML Tag	Description
`[Options]`	The string that names the section in the native configuration file. This string will appear in the configuration file.

`@2[;unordered-lines@`	The number 2 sets the second level of the block.
	
	`[`
	The square bracket opens a new block, in this case a level 2 block, nested within the previous level 1 block.
	
	`unordered lines`
	Indicates that the lines that follow [Options] within the block can be in any order in the configuration file. That is, all the key-value pairs in the [Options] section can be in any order.

3 Next you will define all the options that can appear in the [Options] section of the configuration file. Most of these entries use the CML replace tag because they are simple key-value pairs that allow you to replace a single value. Table 10 explains the CML for each option.

Table 10 Marked Up Key-Value Pairs from UrlScan.ini [Options] Section

CML Tag	**Description**

`AllowDotInPath = @allow_dot_in_path;boolean@`	Note: All of the key-value pairs use some variation of the following syntax (unless otherwise indicated):
	`string literal = @source;type@`
	The string literal defines the actual option name that will appear in the configuration file. The source is the location in the database where the value will be stored in the value wet. The type is the type of data that will be stored in the value set.
	`@allow_dot_in_path`
	This string defines the name space path in which to store this value. In this example, the name space is relative, which means that it will be appended to the name space that you defined in the header of the template (@!namespace=/security/@) and will store the value in that name space location. That is, the value will be stored in the database at the key `/security/allow_dot_in_path.`
	You could also write this tag as follows:
	`AllowDotInPath = @/security/allow_dot_in_path;boolean@`
	`boolean`
	Since the key-value pair type is Boolean, the CML type: `boolean` is used. Note that since the header of this template defined the Boolean true value as 1, when the IIS administrator sets the value set, they would need to enter a 1 to allow dots in the path of the IIS server.

`AllowHighBitCharacters = @allow_high_bit_characters;boolean@`	Similar to the previous example, AllowHighBitCharacters is the option that appears in the configuration file, allow_high_bit_characters is the relative name space path, and boolean is the data type. This IIS option allows users to choose whether or not high bit characters are acceptable in a URL, indicated by a 1 for true and 0 for false in the configuration file.
`AllowLateScanning = @allow_late_scanning;boolean@`	Allows the IIS administrator to choose whether or not late scanning of a URL is acceptable. Defines a name space location to store the value. `boolean` indicates this key can be 1 for true and 0 for false in the configuration file.

Table 10 Marked Up Key-Value Pairs from UrlScan.ini [Options] Section (cont'd)

CML Tag	Description
`AlternateServerName = @alternate_servername@`	Defines a name space location where an alternate server name can be stored when entered by the user or read in from a configuration file. Since no type is specified, the default is a string type.
`EnableLogging = @enable_logging;boolean@`	Allows users to turn on logging, indicated by 1 for true and 0 for false in the configuration file.
`LoggingDirectory = @logging_directory;dir@`	Allows users to choose a directory to store log files, if logging has been turned on. The type `dir` indicates a directory.
`LogLongURLs = @log_long_urls;boolean@`	Allows users to choose whether or not to log URLs that access the server, specified by 1 for true and 0 for false in the configuration file.
`NormalizeUrlBeforeScan = @normalize_url_before_scan;boolean@`	Allows users to choose whether or not to normalize the URL before it is read by the server, indicated by 1 for true and 0 for false in the configuration file.
`PerDayLogging = @per_day_logging;boolean@`	Allows users to choose to turn on per day logging, indicated by 1 for true and 0 for false in the configuration file.
`PerProcessLogging = @per_process_logging;boolean@`	Allows users to turn on or off per process logging, indicated by 1 for true and 0 for false in the configuration file.

`RejectResponseUrl = @reject_response_url;string;r'(HTTP_URLSCAN_STATUS_HEADER)|(HTTP_URLSCAN_ORIGINAL_VERB)|(HTTP_URLSCAN_ORIGINAL_URL)';optional@`	Syntax:
	`string literal = @source;type;r'regular expression';option@`
	`reject response`
	String literal that defines the path where the strings will be stored in the name space.
	`string`
	Indicates that the data type for the reject URL request is a string.
	`r'`
	This is a string range specifier that introduces a regular expression. In this case, a range of string literals.
	`(HTTP_URLSCAN_STATUS_HEADER)|(HTTP_URLSCAN_ORIGINAL_VERB)|(HTTP_URLSCAN_ORIGINAL_URL)'`
	The string literals (rejected URL responses) to be read by the parser: the status header, original verb, and original URL.
	`optional`
	Indicates that this value is optional. That is, the RejectResponseUrl option may be omitted from the UrlScan.ini file.

Table 10 Marked Up Key-Value Pairs from UrlScan.ini [Options] Section (cont'd)

CML Tag	Description
`RemoveServerHeader = @remove_server_header;boolean@`	Allows users to turn on or off the RemoveServerHeading feature. When activated (set to 1), the reject response sent to the client will remove the server header in the message. This setting is indicated by 1 for true and 0 for false in the configuration file.
`UseAllowVerbs = @use_allow_verbs;boolean@`	Allows users to turn on or off the UseAllowVerbs feature. When activated (set to 1), the server will reject any request to the server that contains an HTTP verb that is not explicitly listed in the AllowVerbs section of the UrlScan.ini file. Indicated by 1 for true and 0 for false in the configuration file.
`UseAllowExtensions = @use_allow_extensions;boolean@`	Allows users to turn on or off the UseAllowExtension feature. When activated (set to 1), the server will reject any request to the server that contains a file extension that is not explicitly listed in the AllowExtension section of the UrlScan.ini file. Indicated by 1 for true and 0 for false in the configuration file.
`UseFastPathReject = @use_fast_path_reject;boolean@`	Allows users to turn on or off the UseFastPathReject feature. When activated (set to 1), the server ignores the RejectResponseUrl option and returns a short 404 response to the client when a URL is rejected. Indicated by 1 for true and 0 for false in the configuration file.
`VerifyNormalization = @verify_normalization;boolean@`	Allows users to turn on or off normalization of all URLs scanned by UrlScan.ini. When activated (set to 1), the URL is normalized before being scanned. Indicated by 1 for true and 0 for false in the configuration file.

5. Define the [AllowExtensions] Section - Closing a Block by Opening a New Block

Now that you have defined all of the options in the [Options] section of the UrlScan.ini file, you are ready to start defining the next section, [AllowExtensions]. Remember that to start the [Options] section you had to open a two-level block to account for two levels of information — the title of the [Options] section and its contents.

Before you can start defining the [AllowExtensions] section, you need to close the previous section by closing the CML block. With CML, you can close a block explicitly with the "]" tag, or by opening a new block at a higher level (specified by a lower number) or at an equal level. In this task, you will open the new block for the [AllowExtensions] the same way you opened a block for the [Options] section, by starting a new level 1 block. This automatically closes the blocks opened by the [Options] section.

To open a new block and define the [AllowExtensions] section:

1 After the last line of the [Options] section, enter the following to open the new block for the [AllowExtensions] section:

```
@1[;optional;ordered-lines@
[AllowExtensions]
@2[;unordered-lines@
```

Table 11 explains how opening a new two level block closes the previous block.

Table 11 Starting a New Block for the [AllowExtensions] Section

CML Tag	Description

`@1[;optional;ordered-lines@`	The number 1 opens a new level one block. Because it is a number 1 level block, which is at a higher level than the previous block (a level two block for the key-value pairs in the [Options] section) and equal to the level 1 block before that, it will close the two blocks that came before it.
	
	Note that you can explicitly close a block by using the close block tag. For example:
	`@2]@`
	
	`[`
	CML block tag that opens a new block.
	
	`optional`
	Indicates that this entire block is optional and not required to be in the configuration file.
	
	`ordered-lines`
	Indicates that whatever follows this tag (the string [AllowExtensions] has to come first in the native UrlScan.ini configuration file. In other words, you could not list all the options in the native file and then the title. [AllowExtensions] has to come first. In CML, the ordered-line element determines this order.

`[Options]`	The literal string that names the section in the native configuration file.

`@2[;unordered-lines@`	The number 2 sets the second level of the block.
	
	`[`
	
	CML block symbol that opens a new block.
	
	`unordered lines`
	Indicates that all the lines that follow [AllowExtensions] within the block can be in any order in the configuration file. That is, all the key-value pairs in the [AllowExtensions] section can be in any order.

2 Next, because the [AllowExtensions] section of the UrlScan.ini file can contain any list of file extensions entered by the user, you will use a CML loop and loop target tag to instruct the parser to read the information in this section one line at a time. Immediately after the last `@2[;unordered-lines@` text from the last step, enter the following text:

```
@*allow_extension;unordered-string-set@
.@.@
```

Table 12 explains the how the loop and loop target CML tags work:

Table 12 Loop and Loop Target CML Tags

CML Tag	Description

`@*allow_extension;unordered-string-set@`	Syntax
	`@<level><tag type><name>;<data type>;<options>@`
	The loop tag (`*`) will loop and read over the unordered string set listed in the [AllowExtensions] section.
	`allow_extension`
	String that defines the path where the strings will be stored in the the name space.
	`unordered-string-set`
	Indicates that the list of strings do not have to be in any specific order.

`.@.@`	First (`.`) In this section, this unordered string set that the parser reads is a list of file extensions listed in the [AllowExtensions] section that start with a (`.`) character. `@.@` Loop target tag (`.`) instructs the parser to read everything in this list that starts with a period character.

3 Save the file.

6. Define the [DenyExtensions] Section

Next you define the [DenyExtensions] section of the UrlScan.ini file the same way you defined the [AllowExtensions] section. You will open a new level one block, which closes the previous block from the [AllowExtensions] section. Then you will open a level two block from which you will instruct the parser to read an unordered list of all file extensions beginning with a (`.`) that you want to block using UrlScan.ini.

The CML for the [DenyExtensions] section looks like this:

```
@1[;optional;ordered-lines@
[DenyExtensions]
@2[;unordered-lines@
```

```
@*deny_extension;unordered-string-set@
.@.@
```

7. Define the [AllowVerbs] and [DenyVerbs] Sections

The next two sections of the UrlScan.ini file follow the same CML you used for [DenyExtensions] in the previous sections. You open a first level block to close the previous block, which will also parse the following text as an ordered line.

Then you open a second level block that reads the following list of unordered strings — in other words, a list of verbs. In these two sections, the string instructs the parser to read the list of verbs you want to allow into your web site and a list of verbs you want to deny access to your web site.

The CML for both of these sections is as follows:

```
@1[;optional;ordered-lines@
[AllowVerbs]
@2[;unordered-lines@
@*allow_verb;unordered-string-set@
@.@

@1[;optional;ordered-lines@
[DenyVerbs]
@2[;unordered-lines@
@*deny_verb;unordered-string-set@
@.@
```

8. Define the [DenyHeaders] Section

Next you define the [DenyHeaders] section of the UrlScan.ini file, which allows you to configure IIS to deny specific HTTP request headers.

This section is similar to the previous sections in that you open two blocks for strings. However, you will separate the list of HTTP headers listed in the UrlScan.ini file by a colon, using a CML sequence delimiter. Since HTTP request headers contain a colon (:), you need to use a sequence delimiter to tell the parser to read each line in the section so when it encounters a colon (:), it will move on to the next entry.

For example, the list of HTTP headers to be denied listed in the UrlScan.ini file might read:

Translate:
If:
Lock-Token:

Because each header request listed in the configuration file ends with a (:), you need to instruct the parser to recognize the (:) as the end of an entry.

1 To define the [DenyHeaders] section, after the last line of the [DenyVerbs] section, enter the following text to open the new block for the [DenyHeaders] section:

```
@1[;optional;ordered-lines@
[DenyHeaders]
@2[;unordered-lines@
```

As in previous sections, these tags open a level one block to be read as an ordered line, then they open a second level block to be read as unordered lines.

2 Next enter the following CML loop and loop target tags to instruct the parser to read through the list of header requests:

```
@*deny_header;unordered-string-set;;sequence-delimiter=":"@
@.@:
```

Loop and Loop Target Tags for the [DenyHeaders] SectionTable 13 describes the syntax of these two tags.

Table 13 Loop and Loop Target Tags for the [DenyHeaders] Section

CML Tag	Description

`@*deny_header;unordered-string-set;;sequence-delimiter=":"@`	`*`
	Indicates a loop CML tag that will read through the list of strings.
	`deny_header`
	String literal that defines the path where the strings will be stored in name spacename space.
	`unordered-string-set`
	Indicates that the list of strings can be listed in any order.
	`;`
	The first semicolon separates the two sections of the tag.
	`;`
	The second semicolon allows you to enter the following colon (:) sequence delimeter without it being interpreted as a range.
	`sequence-delimiter=":"`
	Instructs the parser to read a colon (:) as part of the string and the point at which to move on to the next entry.

`@.@`	Loop target tag instructs the parser to store these values into the `deny_header` name space location. For example: /security/deny_header.
`:`	The final colon (:) tells the parser that each item in this list will be followed by a colon. That is, this character will be stored as a part of the entry for a denied header.

3 Save the file.

9. Define the [DenyURLSequences] Section

Define the [DenyUrlSequence] similar to the [DenyHeader] section. Open two blocks that will be read for order and unordered strings. However, for this section you will separate the list of URL sequences in the template with a field delimiter. The field delimiter used here will be an end of line element which instructs the parser stop reading an entry when it encounters the end of a line.

To define the [DenyUrlSequence] section:

1 After the last line of the [DenyUrlSequence] section, enter the following text to open the new block for the [DenyUrlSequence] section:

```
@1[;optional;ordered-lines@
[DenyUrlSequence]
@2[;unordered-lines@
```

As in previous sections, these tags open a level one block to be read as an ordered line, then they open a second level block to be read as unordered lines.

2 Next type the following CML loop and loop target tags to instruct the parser to read through the list of URL sequences to be denied:

```
@*deny_url_sequence;unordered-string-set;;field-delimiter-is-eol@
@.@
```

Table 14 describes the syntax of these tags.

Table 14 Loop and Loop Target Tags for the [DenyUrlSequence] Section

CML Tag	Description

`@*deny_url_sequence;unordered-string-set;;field-delimiter-is-eol@`	`*`
	Indicates a loop CML tag that reads through the list of strings.
	
	`deny_url_sequence`
	String literal that defines the path where the string will be stored in the name space.
	
	`unordered-string-set`
	Indicates that the list of strings can be listed in any order.
	
	`;`
	The first semicolon separates the two sections of the tag.
	
	`;`
	The second semicolon allows you to enter the following colon (:) sequence delimeter without it being interpreted as a range.
	
	`sequence-delimiter=":"`
	Instructs the parser to read a colon (:) as part of the string and the point at which to move on to the next entry.

`@.@`	Loop target tag instructs the parser to store these values into the `deny_url_sequence` name space location. For example: /security/deny_url_sequence.

3 Save the file.

10. Define the [RequestLimits] Section

Defining the [RequestsLimits] is very similar to the way you defined the [DenyUrlSequence] section. Open two blocks that will be read for order and unordered strings. But for this section, after you open both blocks, you will use the CML replace tag to define three key-value pairs.

To define the [RequestsLimits] section:

1 After the last line of the [RequestsLimits] section, enter the following text to open the new block for the [RequestsLimits] section:

```
@1[;optional;ordered-lines@
[RequestsLimits]
@2[;unordered-lines@
```

As in previous sections, these tags open a level one block to be read as an ordered line, then open a second level block to be read as unordered lines. Recall that by starting the new first level block, you are closing the previous second level block from the {DenyUrlSequence] section.

2 Next type the following CML replace tags to define the three kay value pairs found in the [RequestsLimits] section:

```
MaxAllowedContentLength = @max_allowed_content_length;int@
MaxUrl = @max_url;int@
MaxQueryString = @max_query_string;int@
@1]@
```

Table 15 describes the syntax of these tags.

Table 15 Loop and Loop Target Tags for the [DenyUrlSequence] Section

CML Tag	Description
`MaxAllowedContentLength = @max_allowed_content_length;int@`	`MaxAllowedContentLength` Request limit parameter string from the configuration file. `max_allowed_content_length` String literal that defines the path where the value will be stored in the name space. `int` Indicates that the value to be stored is an integer.
`MaxUrl = @max_url;int@`	`MaxUrl` Request limit parameter string from the configuration file. `max_url` String literal that defines the path where the value will be stored in the name space. `int` Indicates that the value to be stored is an integer.

Table 15 Loop and Loop Target Tags for the [DenyUrlSequence] Section (cont'd)

CML Tag	Description

`MaxQueryString = @max_query_string;int@`	`MaxQueryString`
	Request limit parameter string from the configuration file.
	`max_query_string`
	String literal that defines the path where the value will be stored in the name space.
	`int`
	Indicates that the value to be stored is an integer.

`@1]@`	This level one block tag closes the block.

3 Save the File

11. Place the Template in an Application Configuration

Once you have created the CML template for UrlScan.ini and saved it as url_scan_ini.tpl, you are now ready to do the following tasks:

- Import the template into the SA Client and validate the CML syntax. See Importing and Validating a Template File on page 18.
- Add the template to an Application Configuration. See Adding or Removing Templates from an Application Configuration on page 19.
- Attach the Application Configuration to a server. See Attaching an Application Configuration to a Server or Device Group on page 23.
- Test your template by making changes and pushing them to a server. See Pushing Application Configurations on page 26.

These steps are described in CML Tutorial 1 - Create an Application Configuration for a Simple Web App Server on page 85.

Sample UrlScan.ini File

Below is a sample UrlScan.ini file.

```
[Options]
UseAllowVerbs=1                ; If 1, use [AllowVerbs] section, else use the
                               ; [DenyVerbs] section.   The default is 1.

UseAllowExtensions=0           ; If 1, use [AllowExtensions] section, else
                               ; use the [DenyExtensions] section. The
                               ; default is 0.

NormalizeUrlBeforeScan=1       ; If 1, canonicalize URL before processing.
                               ; The default is 1.  Note that setting this
                               ; to 0 will make checks based on extensions,
                               ; and the URL unreliable and is therefore not
                               ; recommend other than for testing.

VerifyNormalization=1         ; If 1, canonicalize URL twice and reject
                               ; request if a change occurs.  The default
                               ; is 1.

AllowHighBitCharacters=0      ; If 1, allow high bit (ie. UTF8 or MBCS)
                               ; characters in URL.  The default is 0.

AllowDotInPath=0              ; If 1, allow dots that are not file
                               ; extensions. The default is 0. Note that
                               ; setting this property to 1 will make checks
                               ; based on extensions unreliable and is
                               ; therefore not recommended other than for
                               ; testing.

RemoveServerHeader=1          ; If 1, remove the 'Server' header from
                               ; response.  The default is 0.

EnableLogging=1               ; If 1, log UrlScan activity.  The
                               ; default is 1.  Changes to this property
                               ; will not take effect until UrlScan is
                               ; restarted.

PerProcessLogging=0           ; This property is deprecated for UrlScan
                               ; 3.0 and later.  UrlScan 3.0 and later can
                               ; safely log output from multiple processes
                               ; to the same log file.  Changes to this
                               ; property will not take effect until
                               ; UrlScan is restarted.

AllowLateScanning=0           ; If 1, then UrlScan will load as a low
                               ; priority filter.  The default is 0.  Note
                               ; that this setting should only be used in
                               ; the case where there another installed
                               ; filter is modifying the URL and you wish
                               ; to have UrlScan apply its rules to the
                               ; rewritten URL.  Changes to this property
                               ; will not take effect until UrlScan is
                               ; restarted.

PerDayLogging=1               ; If 1, UrlScan will produce a new log each
                               ; day with activity in the form
                               ; 'UrlScan.010101.log'. If 0, UrlScan will
                               ; log activity to urlscan.log.  The default
                               ; is 1.  Changes to this setting will not
                               ; take effect until UrlScan is restarted.

UseFastPathReject=0           ; If 1, then UrlScan will not use the
                               ; RejectResponseUrl.  On IIS versions less
                               ; than 6.0, this will also prevent IIS
                               ; from writing rejected requests to the
                               ; W3SVC log.  UrlScan will log rejected
                               ; requests regardless of this setting.  The
```

```
                               ; default is 0.

LogLongUrls=0                  ; This property is deprecated for UrlScan 3.0
                               ; and later. UrlScan 3.0 and later will
                               ; always include the complete URL in its log
                               ; file.

UnescapeQueryString=1          ; If 1, UrlScan will perform two passes on
                               ; each query string scan, once with the raw
                               ; query string and once after unescaping it.
                               ; If 0, UrlScan will only look at the raw
                               ; query string as sent by the client.  The
                               ; default is 1. Note that if this property is
                               ; set to 0, then checks based on the query
                               ; string will be unreliable.

RejectResponseUrl=

LoggingDirectory=Logs

[AllowVerbs]

;
; The verbs (aka HTTP methods) listed here are those commonly
; processed by a typical IIS server.
;
; Note that these entries are effective if "UseAllowVerbs=1"
; is set in the [Options] section above.
;

GET
HEAD
POST

[DenyVerbs]

;
; The verbs (aka HTTP methods) listed here are used for publishing
; content to an IIS server via WebDAV.
;
; Note that these entries are effective if "UseAllowVerbs=0"
; is set in the [Options] section above.
;

PROPFIND
PROPPATCH
MKCOL
DELETE
PUT
COPY
MOVE
LOCK
UNLOCK
OPTIONS
SEARCH
```

```
[DenyHeaders]

;
; The following request headers alter processing of a
; request by causing the server to process the request
; as if it were intended to be a WebDAV request, instead
; of a request to retrieve a resource.
;

Translate:
If:
Lock-Token:
Transfer-Encoding:

[AllowExtensions]

;
; Extensions listed here are commonly used on a typical IIS server.
;
; Note that these entries are effective if "UseAllowExtensions=1"
; is set in the [Options] section above.
;

.htm
.html
.txt
.jpg
.jpeg
.gif

[DenyExtensions]

;
; Extensions listed here either run code directly on the server,
; are processed as scripts, or are static files that are
; generally not intended to be served out.
;
; Note that these entries are effective if "UseAllowExtensions=0"
; is set in the [Options] section above.
;
; Also note that ASP scripts are denied with the below
; settings.  If you wish to enable ASP, remove the
; following extensions from this list:
;    .asp
;    .cer
;    .cdx
;    .asa
;

; Deny executables that could run on the server
.exe
.bat
.cmd
.com
```

```
; Deny infrequently used scripts
.htw     ; Maps to webhits.dll, part of Index Server
.ida     ; Maps to idq.dll, part of Index Server
.idq     ; Maps to idq.dll, part of Index Server
.htr     ; Maps to ism.dll, a legacy administrative tool
.idc     ; Maps to httpodbc.dll, a legacy database access tool
.shtm    ; Maps to ssinc.dll, for Server Side Includes
.shtml   ; Maps to ssinc.dll, for Server Side Includes
.stm     ; Maps to ssinc.dll, for Server Side Includes
.printer ; Maps to msw3prt.dll, for Internet Printing Services

; Deny various static files
.ini     ; Configuration files
.log     ; Log files
.pol     ; Policy files
.dat     ; Configuration files
.config  ; Configuration files

[DenyUrlSequences]
;
; If any character sequences listed here appear in the URL for
; any request, that request will be rejected.
;

..  ; Don't allow directory traversals
./  ; Don't allow trailing dot on a directory name
\   ; Don't allow backslashes in URL
:   ; Don't allow alternate stream access
%   ; Don't allow escaping after normalization
&   ; Don't allow multiple CGI processes to run on a single request
```

Complete url_scan_ini.tpl CML Template

Below is the complete url_Scan_ini.tpl template.

```
@##############################################
# \system32\inetsrv\urlscan.ini (Windows)    #
# Version 1.0                                #
# Joe Author (joe_author@your_company.com)   #
##############################################@

@!namespace=/security/@
@!filename-key="/test";filename-default="/c/UrlScan.ini"@
@!optional-whitespace@
@!boolean-yes-format="1";boolean-no-format="0"@
@!line-comment-is-semicolon@
@!unordered-lines@

@############################################
```

```
# Begin data                               #
###########################################@

@1[;optional;ordered-lines@
[Options]
@2[;unordered-lines@
AllowDotInPath = @allow_dot_in_path;boolean@
AllowHighBitCharacters = @allow_high_bit_characters;boolean@
AllowLateScanning = @allow_late_scanning;boolean@
AlternateServerName = @alternate_servername@
EnableLogging = @enable_logging;boolean@
LoggingDirectory = @logging_directory;dir@
LogLongURLs = @log_long_urls;boolean@
NormalizeUrlBeforeScan = @normalize_url_before_scan;boolean@
PerDayLogging = @per_day_logging;boolean@
PerProcessLogging = @per_process_logging;boolean@
RejectResponseUrl =
@reject_response_url;string;r'(HTTP_URLSCAN_STATUS_HEADER)|(HTTP_URLSCAN
_ORIGINAL_VERB)|(HTTP_URLSCAN_ORIGINAL_URL)';optional@
RemoveServerHeader = @remove_server_header;boolean@
UnescapeQueryString = @unescape_query_string;boolean@
UseAllowVerbs = @use_allow_verbs;boolean@
UseAllowExtensions = @use_allow_extensions;boolean@
UseFastPathReject = @use_fast_path_reject;boolean@
VerifyNormalization = @verify_normalization;boolean@

@1[;optional;ordered-lines@
[AllowExtensions]
@2[;unordered-lines@
@*allow_extension;unordered-string-set@
.@.@

@1[;optional;ordered-lines@
[DenyExtensions]
@2[;unordered-lines@
@*deny_extension;unordered-string-set@
.@.@

@1[;optional;ordered-lines@
[AllowVerbs]
@2[;unordered-lines@
@*allow_verb;unordered-string-set@
@.@

@1[;optional;ordered-lines@
[DenyVerbs]
@2[;unordered-lines@
@*deny_verb;unordered-string-set@
@.@

@1[;optional;ordered-lines@
[DenyHeaders]
@2[;unordered-lines@
@*deny_header;unordered-string-set;;sequence-delimiter=":"@
@.@:
```

```
@1[;optional;ordered-lines@
[DenyURLSequences]
@2[;unordered-lines@
@*deny_url_sequence;unordered-string-set;;field-delimiter-is-eol@
@.@

@1[;optional;ordered-lines@
[RequestLimits]
@2[;unordered-lines@
MaxAllowedContentLength = @max_allowed_content_length;int@
MaxUrl = @max_url;int@
MaxQueryString = @max_query_string;int@
@1]@
```

9 CML Primer

This chapter introduces the **Configuration Modeling Language**, CML. For complete details on CML see CML Reference on page 129. See also CML Tutorial 1 - Create an Application Configuration for a Simple Web App Server on page 85 and CML Tutorial 2 - Create a Template of a Web Server Configuration File on page 95.

SA manages configuration files by creating a **configuration template** that it uses to:

- Model the syntax of the configuration file.
- Extract the values from the configuration file and store them as a **value set** in the SA database. Once those values are stored, you can use the SA Client to manage those values.
- Create a new configuration file from the value set.
- Push the new configuration file to your servers.
- Audit the configuration files on your servers to ensure compliance.

Terminology

- **Configuration File** - The file to be managed by SA.
- **Value Set** - The data values from configuration files that can vary from server to server. Values in a value set are stored in the SA database in “key = value” format.
- **Name Space** - The structure of how value sets are stored in the SA database.
- **Configuration Template** - A model of your configuration file written in CML.
- **Configuration Modeling Language (CML)** - The set of instruction tags that are used to model a configuration file in a configuration template.
- **Instruction** or **Tag** - The key words and characters that define the action to be taken. All instructions start and end with the “@” character. The terms “instruction” and “tag” can be used interchangeably.
- **Application Configuration Object** - A container of configuration templates that, when combined with a value set, generates a configuration file that is then “pushed” to your managed servers. This can also contain scripts that are executed as part of the push operation.

CML Basic Concepts

CML, Configuration Modeling Language, models the syntax of a configuration file. You use CML to create a **configuration template**, which is a model of the target configuration file. To do this, it is usually best to obtain the documentation for the target configuration file so you can understand the valid values and ranges in the configuration file and determine the best way to model it.

Configuration templates work best when the entire configuration file is modeled by CML. However, it is possible to only write CML for some of the lines in a configuration file. This is called a partial template and is discussed at Partial Templates on page 128.

Required CML Instruction Tags

The following three CML instructions (also called CML tags) are required in any configuration template.

- `namespace` defines the key where the value set data is stored in the SA database. See The namespace Tag on page 118.
- `filename-key` defines the key where the target configuration file name is stored in the SA database. See The filename-key Tag on page 119.
- `filename-default` specifies the default name of the target configuration file. See The filename-default Tag on page 120.

All other tags are optional and are used to model the contents of the specific configuration file. For complete details on CML, see CML Reference on page 129. For details on these three required tags, see CML Global Option Attributes on page 151.

The namespace Tag

The `namespace` instruction defines the key where value sets are stored in the SA database.

Syntax

```
@!namespace=<path>@
```

where <*path*> is a string similar to a directory path that defines the key where value sets are stored in the SA database.

Example

```
@!namespace=/example/namespace/@
```

This example sets the base name space for all other instructions in this template to "/example/namespace/". That is, all values in the value set will be stored at the key "/example/namespace/" unless specified otherwise.

Description

The namespace instruction specifies the key in the key/value mapping for the values in the value set. It is an arbitrary string and can be anything except a number. This instruction determines the key where the values in the value set are stored in the SA database. The Replace tag (and other tags) in the configuration file use the name space key to obtain values from the SA database.

The name space value can be relative or absolute, similar to relative or absolute path names.

- Relative names get appended to the value specified in the namespace instruction tag. For example, any value matching the following tag:

  ```
  @testval@
  ```

  would get stored in the value set under the key "/examples/namespace/testval".
- Absolute names are the full path name starting with a "/" character. These names do not use the value specified in the namespace instruction. For example, any value matching the following tag:

  ```
  @/testval@
  ```

  would get stored in the value set under the key "/testval".

Note that any named tag that is part of a loop requires a dot "." in front of the name, and that tags name space gets appended to the current loop's name space. For example:

```
@.testval@
```

The filename-key Tag

The filename-key instruction specifies the key where the target configuration file name gets stored in the SA database.

Syntax

`@!filename-key=<`*`path`*`>@'`

where <*path*> is an arbitrary string similar to a directory path that defines the key where configuration file name is stored in the SA database.

Examples

```
@!filename-key=/files/example@
```

This example specifies that the file name will be stored in the SA database with the key "/files/ example". Note that because the <path> value starts with a "/" character, it is an absolute path.

```
@!filename-key=files/example@
```

This example specifies a relative path because the value does not start with a "/" character. If it were combined with the previous namespace example, the configuration file name would be stored at /example/namespace/files/example".

Description

The filename-key instruction specifies the key that will be used to store the target configuration file name in the SA database. For example, when you set the “Filename” field for a template in the SA Client, that file name will be stored under the key “/files/example” in the value set. Note that the filename-key is not a file system path, but just a key that can be written similar to a path.

This can be very handy for pre and post scripts that need to know the name of the configuration file before or after it has been pushed to the target servers. For example, if you have a post script that needs to add a line to the end of the configuration file after it has been pushed, it might look something like this:

```
echo “#end of the file” >> @/files/example@
```

The filename-default Tag

Syntax

```
@!filename-default=<file>@
```

where <*file*> is the directory path and file name of the target configuration file in the file system of your managed servers. This is where the generated configuration file will be pushed on to your managed servers.

Example

```
@!filename-default=/etc/hosts@
```

This example specifies that the target configuration file is /etc/hosts. This value is stored in the SA database with the key specified in the Filename-key instruction.

Description

The filename-default instruction specifies the standard file system path of the target configuration file. This value is the default file name and directory and is stored under the key specified by the filename-key instruction. It is the default value in the "Filename" field in the SA Client.

Combining Tags on One Line

You can combine multiple instruction tags into one instruction by separating the instructions by semicolons and only using a single exclamation point at the beginning as follows:

```
@!namespace=/example/namespace/;filename-key=/files/example;
filename-default=/etc/example@
```

This can be handy, since this one line included in any file makes it a valid CML template, assuming of course, that all other CML tags are correctly formed.

Use Case 1 - Simple Key=Value Configuration File

The simplest type of configuration file has one or more Key = Value entries as in the following example:

```
Port = 1280
IPAddress = 192.168.0.1
ServerName = server01
```

In this configuration file, types and descriptions are easy to figure out.

Using the Replace Instruction

To write a template for this configuration file, use a CML tag to represent where the value exists and where it will be stored in the value set name space. The tag for this is the Replace tag.

The replace tag is composed of several fields and, as with all tags in the CML language, it begins and ends with the "@" symbol and the fields are separated by semicolons. The form looks something like this:

@ <*name*> ; [*type*] ; [*range*] ; [*option*] ; [*option*] … @

Of all the fields, only the <*name*> field is required. So the most basic CML that could represent the configuration file above would be:

```
@!namespace=/example/namespace/@
@!filename-key=/files/example@
@!filename-default=/etc/example@
Port = @port@
IPAddress = @ipaddress@
ServerName = @servername@
```

This specifies that the value for the port number will be stored in the SA database at the key "/example/namespace/port". The IP address will be stored at the key "/example/namespace/ipaddress" and the server name at the key "/example/namespace/servername".

This would technically work, but you would be missing a lot of the field validation and error checking available that prevents entering invalid data such as "someport" for the port, for example.

The <name> Field in the Replace Instruction Tag

If the <*name*> field is relative (that is, it does not start with a "/" or a ".") it gets appended to the current name space and becomes part of the key used to store the value read from the SA database by this tag.

If the name is absolute (that is, it starts with a "/") it is the entire key and the value gets stored under this key.

Finally, if the name starts with a dot ".", it will be appended to the name space of whatever loop it is a part of. With very few exceptions, every tag inside a loop should start with a dot, ".".

The <type> Field in the Replace Instruction Tag

The <*type*> field lets you assign certain predefined ranges and error checking to different values, based on well-known types. For the full list of types, see CML Type Attributes on page 144. For this configuration file, use the predefined types “port,” “ip” and “hostname” for the separate entries, as follows:

```
@!namespace=/example/namespace/@
@!filename-key=/files/example@
@!filename-default=/etc/example@
Port = @port;port@
IPAddress = @ipaddress;ip@
ServerName = @servername;hostname@
```

Adding these types restricts the values and provides validation and error checking.

You can also use the replace tag to represent a sequence of repeating values by prepending “ordered-” or “unordered-” and appending “-set” or “-list”. More on this in the next example.

The default for this field is “string”, which will match anything.

The <range> Field in the Replace Instruction Tag

The <*range*> field allows you to set the allowable range for the values. You can set either integer ranges or string ranges. Integer ranges are valid for any type that consists of strictly integers, string ranges are valid for all other types.

Ranges can be combined with logical OR by using a comma, “,”. Ranges can be combined with logical AND by using the ampersand character, “&”. The “!” character negates the range.

Keep in mind that the specified ranges will be used when reading in a configuration file as well as when accepting values from the SA Client. If you have a configuration file that has a value outside of the ranges you set in the template, then an error will be given when parsing that file. Specify the valid ranges based on the documentation for the configuration file.

Integer Ranges

Integer ranges can only use the < and the = symbols to specify “less than” or “greater than” ranges. Specify the position of the number used in the comparison as follows:

Table 16 Specifying Integer Ranges

Range Condition	Symbols to Use
Greater than	n<
Greater than or equal	n<=
Less than	<n
Less than or equal	<=n
Equal	=n

For example, if the ports in the configuration file can only be between 1024 and 2048 inclusive, you would add the ranges to the tag like this:

```
Port = @port;port;1024<=,<=2048@
```

String Ranges

String ranges can be a list of valid strings surrounded by quotes and a list of regular expressions starting with the characters r" and ending with a quote. For example, if the ServerName field can only be anything starting with the word "server", you would want to add the ranges to the servername tag like this:

```
ServerName = @servername;hostname;r"server.*"@
```

The [option] Fields in the Replace Instruction Tag

You may append as many options as you need to the tag. Everything after the third semicolon is considered an option and every option is separated by semicolons.

For example, if the IPAddress line in the configuration file is optional and not required to make the configuration file valid, and the IP address could stop at a forward-slash, you could add the option like this:

```
IPAddress = @ipaddress;ip;;optional;delimiter="/"@
```

This would match the following entry:

```
IPAddress = 192.168.0.1
```

It would also match the following entry:

```
IPAddress = 192.168.0.2/
```

Notice that the range field is left empty. Any fields you want to leave as default must still be represented if you want to fill in any later fields. For instance, the following two lines are valid:

```
@ipaddress@

@ipaddress;;;optional@
```

However, the following is not valid because the field "optional" will be interpreted as the <type> field rather than as an option and will result in an error.

```
@ipaddress;optional@
```

The Final CML Template

After all the types, options and ranges are set, the CML template should look something like this:

```
@!namespace=/example/namespace/@
@!filename-key=/files/example@
@!filename-default=/etc/example@
Port = @port;port;1024<=,<=2048@
IPAddress = @ipaddress;ip;;optional;delimiter="/"@
ServerName = @servername;hostname;r"server.*"@
```

The Resulting Value Set

Using the above configuration template to read in the example target configuration file from above will result in the following value set stored in the SA database:

```
/example/namespace/port = 1280
/example/namespace/ip = 192.168.0.1
/example/namespace/servername = server01
```

As you can see, the keys in the name space are a combination of the template's name space (/example/namespace) and each individual tag's name, since they all use relative names.

Use Case 2 - Repeating Values in the Configuration File

It is possible you will encounter a configuration file that will be nothing but a list of values, for example a file that contains only a list of user names that have write access to a directory. The format of this file could look something like this:

```
admin;
user1;
user2;
```

The repeating lines in this file call for more than the replace tag described in the previous example. The most basic CML that could match this configuration file is the following loop instruction:

```
@!namespace=/wuserlist/namespace/@
@!filename-key=/wuserlistfile/example@
@!filename-default=/etc/wusers.txt@
@*users@
@.@
```

Using the Loop Instruction Tag

The Loop instruction is used when a set of values may appear multiple times in a configuration file. The default behavior of the loop instruction is to loop over the line of CML directly after it, though this can be modified to loop over multiple lines or within a single line.

The form looks something like this:

@ [*group level*] * <*name*> ; ["ordered" | "unordered"] - [*type*] - ["set" | "list"] ; [*range*] ; [*option*] ; [*option*] … @

You'll notice some differences and similarities between this tag and the Replace tag in the previous section. The following sections describe each of the options on the Loop tag.

The <group level> Field

For this example configuration file, group level is not important and it will be discussed later. For now just know that it is used to determine what specific section to loop over, whether within a single line or over multiple lines.

For this example, leave the group level blank which indicates that this loop tag will only iterate over the CML line directly below it.

The Instruction Type Specifier Field, *

The "*" is an instruction type specifier. It signifies what type of instruction this is, in this case, a Loop instruction. The Replace instruction is the default instruction type since it is so common, therefore it does not have an instruction type specifier.

The <name> Field

The same <*name*> field rules apply to this tag as to the replace tag. To review, see The <name> Field in the Replace Instruction Tag on page 121. Any named tag that is part of this loop will need a "." (dot) in front of the name, and that tag's namespace gets appended to this loop's namespace. Likewise, if this loop were a part of another loop, it would need a "." in front of the name.

This example will use the name "users" as follows:

```
@*users@
```

The [type] Field

The [*type*] field for the Loop tag is different from the Replace tag's [*type*] field in two major ways. (To review, see The <type> Field in the Replace Instruction Tag on page 122.)

- Ordered vs. Unordered and Set vs. List

  The basic types include all the same types as the Replace tag, but since this will be a repeating sequence of values, information about the sequence needs to be specified. This information is included in this modified [type] field by prepending "ordered" or "unordered", followed by a dash, and appending a dash followed by "set" or "list" to the type.

  — Prepending "ordered" specifies that the order of the values will be preserved.

  — Prepending "unordered" specifies that the values can be in any order.

  — Appending "set" specifies that the values must be unique.

  — Appending "list" specifies that the values can be repeating.

  While this is optional for the Replace tag, the ordered or unordered option and set or list option is required for the Loop tag.

  For this example, the data is unordered so "set" should be appended, as there would be no reason to order an access list or to have repeating values.

- Namespace Type

  This type is unique to the Loop tag. If the section you are going to be iterating over contains more than one value, then you need to use the name space type.

The default type for this tag is "unordered-string-list".

For this example the default value would work, but it would be better to specify the type to be "user". Assuming an unordered set, the resulting tag would look like this:

```
@*users;unordered-user-set@
```

The [range] Field

Ranges are the same as in the Replace tag for every type except the name space type. Since the name space type iterates over several different tags that may have their own ranges, no range should be used.

Since this example uses the “user” type, it can also use a range. For example, if the documentation for this configuration file were to say that “root” is not a valid user, you could set the range to be valid for anything except root as follows:

```
@*users;unordered-user-set;!"root;"@
```

The [option] Field

The Option field is the same as the Replace tag's option field. To review, see The [option] Fields in the Replace Instruction Tag on page 123.

For this example, we can eliminate the “;” from the user names by setting semicolon as the field delimiter and including it in the line we are iterating over to eliminate it from the value that is read in, as follows:

```
@*users;unordered-user-set;!"root";field-delimiter-is-semicolon@
@.@;
```

The Loop Target Tag

The loop target tag looks like the following:

@.@

Its only purpose is to signify the position of the loop value, which is where the data will be placed in the resulting configuration file.

The Final CML

After all the types, options and ranges are set, the CML template should look something like this:

```
@!namespace=/wuserlist/namespace/@
@!filename-key=/wuserlistfile/example@
@!filename-default=/etc/wusers.txt@
@*users;unordered-user-set;!"root";field-delimiter-is-semicolon@
@.@;
```

The Resulting Value Set

Every value that gets read in will need to be stored in the value set under a unique key. To handle sequences, CML will append a unique number on to the name space starting at 1 and incrementing for each additional iteration.

The resulting value set for the example configuration file using the CML above will look like the following:

```
/example/namespace/users/1 = admin
/example/namespace/users/2 = user1
/example/namespace/users/3 = user2
```

Use Case 3 - Complex Repeating Values in the Configuration File

This example models the /etc/hosts file, which is a list of IP addresses followed by a list of host names like this:

```
127.0.0.1 localhost
192.168.0.1 server1 server1.domain.com
192.168.0.2 server2 server2.domain.com
```

This example is similar to the previous example, except that this example iterates over a more complex line. The best way to think about this is by first modeling the single instance of the line using CML Replace instructions like this:

```
@ip-addr;ip@ @sname;unordered-hostname-set@
```

This line defines two replace instructions, one for the IP address and one for the server name.

Notice that the “ip-addr” replace tag specifies the data type as “ip” because it must be an IP address.

The “sname” replace tag is typed as an “unordered-hostname-set”. This means that it can match a list of host names and it will store them along with the corresponding IP address. This is similar to how the Loop tag works and the values get stored in the same way.

This CML is for one iteration. The next step encloses it in a loop. To do this, use the name space loop, since it is iterating over more than one value on each line, and prepend a “.” to the names of the tags in the loop, as follows:

```
@*entries;unordered-namespace-set@
@.ip-addr;ip@ @.sname;unordered-hostname-set@
```

The Loop tag (indicated by the @* characters) defines a loop. The “unordered-hostname-set” indicates that the data are host names, the host names can be in any order, and the values must be unique.

The “.” characters added before the “ip-addr” and “sname” strings in the Replace instruction indicate that these are the target of the Loop instruction.

The Final CML

Adding the above loop and replace CML to the required name space and file lines gives the following.

```
@!namespace=/example/namespace/@
@!filename-key=/files/example@
@!filename-default=/etc/hosts@
@*entries;unordered-namespace-set@
@.ip-addr;ip@ @sname;unordered-hostname-set@
```

The Resulting Value Set

Below are the values that will be stored in the SA database if the entries in the sample file are read in using the SA Client.

```
/example/namespace/entries1/ip-addr = 127.0.0.1
/example/namespace/entries1/sname/1 = localhost
/example/namespace/entries2/ip-addr = 192.168.0.1
/example/namespace/entries2/sname/1 = server1
/example/namespace/entries2/sname/2 = server1.domain.com
/example/namespace/entries3/ip-addr = 192.168.0.2
/example/namespace/entries3/sname/1 = server2
/example/namespace/entries3/sname/2 = server2.domain.com
```

Partial Templates

While it is best to model the entire configuration file, you can use partial templates to model only part of a configuration file. To create a partial template, you must have a copy of the full configuration file on a server for the template to read. And you must use the Preserve Format option to preserve the rest of the file.

The following shows a simple configuration file.

```
UserName = alice
Password = pass
HomeDir = /home/alice
```

To manage only the home directory line, use the `@!partial-template` instruction and model only the line you want to manage. The template would look like this:

```
@!namespace=/example/@
@!filename-key=/files/example@
@!filename-default=/usr/example@
@!partial-template@
HomeDir = @homedir;dir@
```

For more information on the Preserve Format setting, see Setting Fields in the Value Set Editor on page 40. See also The @!full-template and @!partial-template Attributes on page 152.

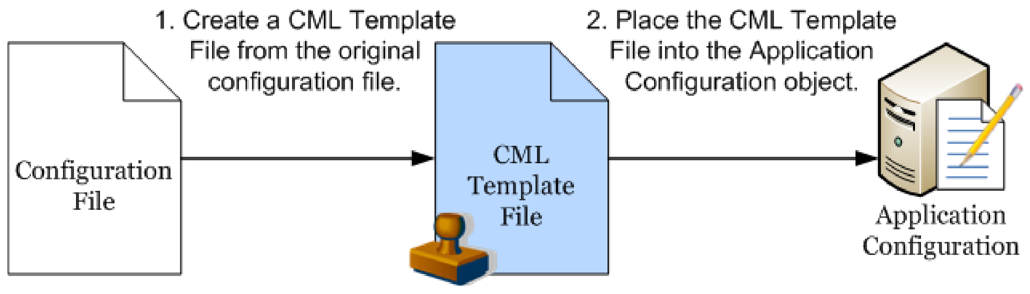
10 CML Reference

The **Configuration Modeling Language** (CML) is used to create a **template** of a **configuration file** so it can be managed from SA. A CML template is a separate file you create that models the format of the configuration file so the variable values in the configuration file can be set to different values for different sets of servers.

The template file contains data, directives and definitions so that an actual configuration file can be generated from the template and a set of values.

CML defines a two-way transform: it specifies how to move values from a configuration file to a value set in the SA database, and it specifies how data from a value set is merged with the template to create a properly formatted configuration file that can be pushed to a managed server.

You also write scripts using CML that are run when pushing configurations to managed servers. For more information, see About Running Scripts with Application Configurations on page 49.


1. Create a CML Template File from the original configuration file.
2. Place the CML Template File into the Application Configuration object.
Configuration File
CML Template File
Application Configuration

XML Configuration Files

SA can also manage XML configuration files. For more information on using XML configuration templates, see Managing XML Configuration Files on page 61.

About Configuration Templates

A configuration template is a “templatized” version of an actual configuration file whose values have been turned into variables. Using the SA Client, you can define a template’s value sets, save them to the SA database, and then propagate those values to a real configuration file on a managed server.

Value sets are stored on the SA database. Storing all values in the SA database allows you to manage configuration values from a central location and ensures configuration consistency across applications in your data center.

Once the template version of the configuration file has been created and added to an application configuration object and you have created a value set for the template, you can push those values to configuration files on managed servers.

CML Overview

A configuration template consists of a series of CML tags. Each tag represents either an instruction to the CML parser how to interpret the text in the configuration file or a placeholder that identifies the location of a value in the configuration file and how to map it into a value set.

Keep in mind that the configuration template contains no values. It only defines how values are moved between the value set in the SA database and the configuration file instances on the managed servers. For more information, see About Value Sets on page 36.

Template files containing CML are typically named with ".tpl" as the file extension, but this file extension is not required.

Structure of CML Tags

The basic building blocks of a CML tag is as follows:

```
@{level}{tagtype}{source};{type};{range};{option};...;{option}@
```

The following rules apply to all CML tag:

- All CML tags start and ends with the @ symbol.
- You cannot use any white space between tags.
- Semicolons (;) mark placeholders for omitted attributes. For example, the following shows two omitted attributes between the @name attribute and the optional@ attribute:

  ```
  @name;;;optional@
  ```

- If attributes to the right of a semicolon are empty, then semicolons are optional. For example:

  ```
  @name@
  ```

- {**level**} - The block level is an integer that specifies the nesting level of the block. The level also determines whether the block spans multiple lines or is part of a single line. If the level is between 1 and 99 it is a multi-line block. If it is above 101, it is a block within a line. Each block open tag closes all previous blocks that have an equal or greater level.

  Do not use level 100 because level 100 is reserved.

- {**tagtype**} - CML defines the tag types listed below. Each type is an instruction to the CML parser. For complete details, see CML Tag Types on page 134.
  - — Comment Tags: @# and @## ... #@ - Define comments in the template.
  - — Replace Tag: @ - Defines how to replace a variable with a value from the value set.
  - — Instruction Tag: @! - Gives an instruction to the CML parser.
  - — Block Tags: @[@...@]@ - Create a new scope.
  - — Loop Tag: @* - Creates a loop over multiple similar values.
  - — Loop Target Tag: @. - Ends a loop.

— Conditional Tag: @?

— DTD Tag: @~ - Defines

- {**source**} - Defines a key where the value is stored in the value set. Absolute path names start with a "/" character. Relative path names do not start with a "/" and are concatenated to the name space key value defined by the `@!namespace` instruction.
- {**type**} - Defines the data type of the value required by the configuration file and the corresponding value in the value set. For example, int for integer, string, boolean, IP-address, and so forth. You can also specify ordered and unordered lists and sets.
- {**range**} - Defines additional restrictions on data values for better error checking.
- {**option**} - Defines additional parameters you can specify to modify the behavior of the CML tag.

Required CML Tags

Every CML file must define its name space and the default file name of the configuration file the template models using the following CML tags:

- `@!namespace` defines the name space for the template. All values in the value set used by the template will be stored in the SA database at the key defined by the `@!namespace` tag. For more information, see Defining the Namespace with the @!namespace CML Tag below.
- `@!filename-key` defines a specific key where the default file name will be stored in the SA database. This key can either be a separate name space or it can be appended to the name space defined by the `@!namespace` tag. For more information, see Defining the Default Configuration File Name with the @!filename-key and @!filename-default CML Tags on page 132.
- `@!filename-default` defines the directory and name of the configuration file being modeled by the template. This value can be modified by the value set. For more information, see Defining the Default Configuration File Name with the @!filename-key and @!filename-default CML Tags on page 132.

Defining the Namespace with the @!namespace CML Tag

The name space in a CML template file defines a unique key value where data is stored in the database. The name space value is represented as a path name and looks like a directory path name in a file system, or a URI in a web browser's location bar. Use the `namespace` tag to define the name space.

The path names for individual values can be either absolute or relative. An absolute path name start with "/" and is the complete representation of the location of the value in the value set. A path name that does not start with a "/" is a relative path name; its value will be appended to the current value of the name space.

The `namespace` tag is required.

➤ All key names in CML templates must be ASCII. Other fields and text can be either ASCII or non-ASCII text.

Below is an example of a `namespace` tag in a CML template:

```
@!namespace=/security/@
```

Defining the Default Configuration File Name with the @!filename-key and @!filename-default CML Tags

Each template must define the default configuration file name that will be used when pushing the generated configuration file to a server. This file name can be overridden by the value sets. Use the `filename-default` tag to define the default file name.

You must also specify a unique key where the default file name will be stored in the SA database. This key can be combined with the name space to generate a unique storage location for the default file name. The key defines a name space is represented as a path name. Use the `filename-key` tag to define the key value.

The `filename-default` and `filename-key` tags are required.

All key names in CML templates must be ASCII. Other fields and text can be either ASCII or non-ASCII text.

The following example CML specifies that they key value "/files/hosts" will be used to store the default file name in the SA database. It also specifies that the default file name for the generated configuration file will be "/etc/hosts".

```
@!filename-key="/files/hosts"@
@!filename-default="/etc/hosts"@
```

You can also combine CML tags on one line as follows:

```
@!filename-key="/files/hosts";filename-default="/etc/hosts"@
```

Example CML Template for /etc/hosts

The following is an example of a CML template that models a typical /etc/hosts file.

```
@##############################################
#                                             #
# /etc/hosts (multiplatform)                  #
# Version 2.0                                 #
# Joe Author (joe_author@your_company.com)    #
#                                             #
##############################################@
@!namespace=/system/dns/@
@!filename-key="/files/hosts";filename-default="/etc/hosts"@
@!unordered-lines;missing-values-are-error@
@!relaxed-whitespace@
@!sequence-delimiter-is-whitespace@
@!line-comment="#"@
@~host/.ip
type = ip
printable = IP address
description = This is an IP address
@
@~host/.hostnames
type = unordered-hostname-set
printable = Hostnames
description = A set of hostnames
@
@1*host;unordered-namespace-set;;sequence-append@
@.ip@.hostnames@

@1]@
```

CML Tag Types

The following are the main CML tags. These are described in detail below.

- Comment Tag: @# and @##
- Replace Tag: @
- Instruction Tags: @!
- Block (or Group) Tag: @[@...@]@
- Loop Tag: @*
- Loop Target Tag: @.
- Conditional Tag: @?
- DTD Tag: @~

Comment Tag: @# and @##

This tag defines a comment in the CML template file. You can define one line comments or multiple line comments.

Syntax

```
@#   <one line comment>
```

Or:

```
@## <comments spanning multiple lines>
    <comments spanning multiple lines>
    <comments spanning multiple lines> #@
```

Description

The comment tag can be used to insert comments anywhere in your CML file.

As a best practice, use the comment tag at the beginning of a CML template to create a header describing the template, such as the name of the template, the configuration file the template is based on, the purpose of the template, the author, the date, and so on.

Attributes

None.

Examples

The following is a one line comment:

```
@# This comment ends at the end of this line.
```

The following is a multiple line comment:

```
@##
   This comment spans
   multiple lines.
#@
```

The following is also a multiple line comment:

```
@##########################################################
# /etc/hosts (multiplatform)                              #
# $Id: hosts.tpl 8650 2006-06-05 05:28:03Z joe_author $ #
##########################################################@
```

Replace Tag: @

This tag replaces text in the template file with a value from the value set.

Syntax

```
@{source}[;[{type}][;[{range}[;{option}][;{option}]...]]]]@
```

Description

The replace tag replaces the tag in a CML line with the data from the specified location in the name space. It is an indicator that the text in this location is data, and it also specifies details about how that data should be stored and validated. The source name is the index key where the data is found in the value set. The other fields of the replacement tag specify details about how the data should be stored and validated.

The replace tag is the only tag that is not indicated by a special character following the "@" character. The only required element in a replace tag is the source. All other elements are optional.

Attributes

- **Source**: The source attribute is the database key used to store and access the value in the value set. If the source attribute is relative (that is, it does not start with a "/" or a ".") it gets appended to the current name space and becomes part of the key used to store the value read by this tag. If the name is absolute (that is, it starts with a "/") it is the key, and the value gets stored under this key.

  The only required element in a replace tag is the source. All other elements are optional. If the name starts with a ".", it will be appended to the name space of the loop it is a part of. Tags inside a loop typically start with a ".".

- **Type**: The type attribute specifies the type of the replace tag, which applies certain predefined restrictions and error checking to different values. The default type for replace tags is "string".

  The available types are described at CML Type Attributes on page 144.

- **Range**: The range attribute allows you to set the range of valued values. (Keep in mind that all ranges will be used when reading in a file as well as when accepting values from a user.) If you have a configuration file that has a value outside of the specified ranges, then an error will occur when parsing that file.

  Ranges are described at CML Range Attributes on page 149.

- **Options**: The option attributes modify the behavior of the tag. Multiple options can be appended to the end of most tags, separated by semicolons. Everything after the third semicolon is considered an option. Options can also be used as instruction tags.

  Options are described at CML Global Option Attributes on page 151 and CML Regular Option Attributes on page 153.

Example 1

```
Title=@main_title@
```

In this example, `main_title` will extract the string that follows "`Title=`" text in the configuration file, and store it at key location /main_title in the value set.

Or if you are performing a push, `main_title` will extract the value stored from location `/main_title` from the value set, and push it after the string "`Title=`" text in the configuration file.

Example 2

```
Port = @port;port;1024<=,<=2048@
IPAddress = @ipaddress;ip;;optional;delimiter="/"@
ServerName = @servername;hostname;"localhost",r"server.*"@
```

Instruction Tags: @!

This tag specifies parser actions. For a list of available instructions, see CML Global Option Attributes on page 151 and CML Regular Option Attributes on page 153.

Syntax

```
@!{option}[[;{option}]...]@
```

Description

The instruction tag sets options that will be used at parse time. For example, defining the name space, whether a list is sorted, ordered, or unordered, how the parser should interpret white space, acceptable delimiters, defining comment characters, and so on.

The only attributes used by an instruction tag are options. One or more option can appear in one instruction tag. Multiple options are separated by semicolons. To understand how any particular instruction tag affects the parser, refer to the descriptions of the embedded options.

Attribute

Only option attributes are used with an instruction tag.

- **Options**: The option attributes in an instruction tag define the behavior of the tag. Multiple options can be appended to the end of most tags, separated by semicolons. Many options are toggles of other options. When an option from one of these toggling groups appears in a block, no other option from that group should appear in the same block.

  Options are described at CML Global Option Attributes on page 151 and CML Regular Option Attributes on page 153.

Example 1

The following instruction tag tells the CML parser that white space in the template will be matched by any combination of tabs and spaces.

```
@!relaxed-whitespace@
```

Example 2

The two options in the following instruction tag tell the CML parser the relative order of lines in the configuration file is not important to mapping values from those lines with the value set and it is not an error if values in the value set are not matched by text in the configuration file.

```
@!unordered-lines;missing-values-are-null@
```

Example 3

```
@!namespace=/test/@ @!filename-key="/test";filename-default="/tmp/test.txt"@
@!optional-whitespace@
@!boolean-yes-format="1";boolean-no-format="0"@ @!line-comment-is-semicolon@
@!unordered-lines@
```

Block (or Group) Tag: @[@...@]@

The Block tag is sometimes also referred to as the Group tag. This tag creates a block or group of related tags and lets you nest groups of related tags.

Syntax

The block tag can have either single line syntax or multiple line syntax.

Single line syntax for the block tag is as follows:

```
@[{level}][ [;{option}[;{option}]...]@ {set of CML tags} @[{level}]]@
```

Multiple line syntax for the block tag is as follows:

```
@[{level}][ [;{option}[;{option}]...]@
{set of CML tags}
@[{level}]]@
```

The level is an integer that determines whether the block spans multiple lines or is part of a single line. If the level is between 1 and 99 it is a multiline block. If it is greater than 101, it is a block within a line. Do not use level 100 because it is reserved.

Each block can be ended explicitly or implicitly. To end a block explicitly, use an end block tag with the level number. For example, the following tag explicitly closes a level 3 block: `@3]@`.

To end a block implicitly, use an end block tag with a lower level number to end an enclosing block, or define a new block with a lower level. Each block open tag will close all previous blocks that have an equal or greater level.

Description

The block tag allows you to group related tags and nest groups of tags. With a block, you can define separate parsing rules for each section of a configuration file.

You can nest blocks within other blocks using a higher number for the level. Any subsequent tag with a level value will close all open levels of equal or great value. The block close tag, `@]@`, is not required.

The opening block tag can include option attributes. Those attributes only affect the tags inside the block at the level declared by the opening tag. Contrast that with instruction tags that appear inside the block: those instruction tags affect the behavior of the current level and any nested blocks.

By using blocks, you can specify unique options for each separate section of the configuration file. For example, you might have a section of a configuration file where the values for True and False are defined as "1" and "0", respectively. In another section in the same file, you could define values for True and False as "T" and "F". Use the block tag to separate the two different ways of defining True and False.

Another example could be if in one section of a configuration file a specific number of spaces are important, while in another section any number of spaces is acceptable. You can use the block tag to indicate where the number of spaces differ.

Attributes

No name, type, or range attributes are used with block tags.

- **Level**: The block level is an integer that specifies the nesting level of the block. The level also determines whether the block spans multiple lines or is part of a single line. If the level is between 1 and 99 it is a multi-line block. If it is above 101, it is a block within a line. Each block open tag closes all previous blocks that have an equal or greater level.

Do not use level 100 because level 100 is reserved.

- **Options**: The option attributes modify the behavior of the CML tags in the block. Instruction tags within the block affect the behavior of CML tags in the current block and in nested blocks. Multiple options can be appended to the end of most tags, separated by semicolons. Options can also be used as instruction tags.

  Options are described at CML Global Option Attributes on page 151 and CML Regular Option Attributes on page 153.

Example 1

The following example creates two blocks, one block nested within the other. The first line defines the first block, which is the outer block. The fourth line defines the second block, which is the inner block nested within the first block. The second to last line closes the inner block. This line is optional. The last line closes the outer block. If the second to last line were omitted, the last line would close both blocks.

```
@1[@
@!ordered-lines@
[SectionOne]
@2[@
@!unordered-lines@
optionA = @section_one/option_a@
optionB = @section_one/option_b@
@2]@
@1]@
```

Example 2

This example models two sections named [Options] and [AllowVerbs] in a Windows `UrlScan.ini` file. Both sections in this file contain a list of key-value pairs.

To define the first section (lines 1 through 3), you can use the block tag (`[`) set at two levels because there are two kinds of data in this section: a fixed heading followed by a list of key-value pairs. The first level block handles the text string "[Options]" while the second level block handles all of the key-value pairs in that section.

The second section (lines 4 through 6) defines the [AllowVerbs] section. Notice that the first section is not explicitly closed with the `@2]@` and `@1]@` tags as in the previous example because opening the next level 1 section (line 4) implicitly closes the previous sections.

```
@1[;optional;ordered-lines@
[Options]
@2[;unordered-lines@
@1[;optional;ordered-lines@
[AllowVerbs]
@2[;unordered-lines@
```

Loop Tag: @*

This tag defines a processing loop. See also the Loop Target Tag: @. on page 140.

Syntax

```
@[{level}]*{source}[;[{type}][;[{range}[;{option}[;{option}]...]]]] @
{target}
```

Description

The Loop tag is used when a set of values may appear multiple times in a configuration file. The default behavior of the loop tag is to iterate over the line of CML directly after it, though this can be modified to iterate over multiple lines or within a single line.

Loops are a form of Group tag, see the Group tag for more information.

The Loop tag allows sequences (lists and sets) to be enumerated. The block associated with a loop element will be processed for each incident of that block in an input file, and will be generated in an output file for each incidence of that data in a value set.

The group associated with a loop element will cause a new element to be stored in the value set for each incident of that group in a configuration file, or each incidence of that data in a value set will push a value to the configuration file. The source attribute is the index key used to map values in the value set.

Attributes

- **Level**: The group level is an integer that determines whether the group spans multiple lines or is part of a single line. If the level is between 1 and 99 it is a multi-line group. If it is above 101, it is a group within a line. Level 100 is reserved for internal purposes. Each group open tag will close all previous groups that have an equal or greater level.
- **Source**: The source attribute is the database key used to access the value in the value set. If the source attribute is relative (that is, it does not start with a "/" or a ".") it gets appended to the current name space and becomes part of the key used to store the value read in by this tag. If the name is absolute (that is, it starts with a "/") it is the key, and the value gets stored under this key. The only required element in a replace tag is the source; everything else is optional. If the source name starts with a ".", it is to be appended to the name space of whatever loop it is a part of. Tags inside loops typically start with a ".".

- **Type**: The type attribute specifies the type of the replace tag, which applies certain predefined restrictions and error checking to different values. The default type for replace tags is “string”.

  For the full list of types see CML Type Attributes on page 144.

  You can prepend “ordered-” or “unordered-” to the type. And you can append “-set” or “-list” to the type.

  — Prepending “ordered-” specifies that the values must be in order.

  — Prepending “unordered-” specifies that the values can be in any order.

  — Appending “-set” specifies that the values must be unique.

  — Appending “-list” specifies that the values can be repeating.

- **Range**: The range attribute allows you to set the range for the values. Keep in mind that all ranges will be used when reading in a file as well as when accepting values from a user. If you have a configuration file that has a value outside of the specified ranges, then an error will occur when parsing that file.

  Ranges are described at CML Range Attributes on page 149.

- **Options**: The options attributes modify the behavior of the tag. Multiple options can be appended to the end most tags, separated by semicolons. Everything after the third semicolon is considered an option. Options can also be used as instruction tag.

  Options are described at CML Global Option Attributes on page 151 and CML Regular Option Attributes on page 153.

Example 1

The asterisk character indicates a loop tag. For example:

```
@1*includegroup;ordered-namespace-set;;optional@
#BEGIN_ALTERNATE
@*.include@
#INCLUDE @.@
#END_ALTERNATE
@1]@
```

Example 2

```
@*users;unordered-user-set;!”root”;field-delimiter-is-semicolon@
@.@;
```

Loop Target Tag: @.

The loop target tag defines an iteration for the Loop tag. See Loop Tag: @* on page 139.

Syntax

```
@.[{source}[;[{type}][;[{range}[;{option}[;{option}]...]]]]]@
```

Description

The loop target tag indicates the placeholder for a value in a loop. If you consider that the loop tag indicates the beginning of a loop, and is therefore similar to a group tag, the loop target tag is quite similar to a replace tag.

When encountered in a group, with each loop iteration, this tag simply maps the text at current position in the configuration file with the current value in value set. If the optional source attribute is used, the source is appended to the name space created by the loop.

Attributes

None.

Example

The loop target tag is indicated by a period following the "@" character. For example:

```
@*keys;unordered-namespace-set@
@.key@ = @.value@
```

Conditional Tag: @?

This tag defines a condition.

Syntax

```
@[{level}]?{source}@{text}
```

Description

The conditional tag maps whether or not the text exists in the configuration file with a Boolean value in the name space. When reading a target configuration file, if the text matches, the name space value gets true, otherwise false. When writing to a configuration file, if the name space value is true then the configuration files gets the text; otherwise, no text is written.

This is one of the few tags in which something outside of the tag is actually the value. The main use of this tag is to store a Boolean true value in a location in the name space if the text after the tag exists.

Attributes

No type, range or option attributes are used with conditional tags.

- **Level**: The level is an integer that determines whether the group spans multiple lines or is part of a single line. If the level is between 1 and 99 it is a multi-line group, if it is above 101, it is a group within a line. Level 100 is reserved for internal purposes. Each group open tag will close all previous groups that have an equal or greater level.
- **Source**: The source attribute is the database key used to access the Boolean value. If the source attribute is relative (does not start with a "/" or a ".") it gets appended to the current name space and becomes part of the key used to store the value read in by this tag. If the name is absolute (starts with a "/") it *is* the key, and the value gets stored under this key.

If the source name starts with a “.”, it is to be appended to the name space of whatever loop it is a part of. Typically a tag inside a loop should start with a “.”.

Example 1

The conditional tag is indicated by a question mark symbol (?). For example:

```
@?debug@options debug
```

In this example, if you were importing a configuration file into a configuration template, and if the text “options debug” exists in the configuration file, then the value at key `/debug` will be set to true.

If you were going to push the application configuration, if the value stored at key `/debug` is true, then the text “options debug” will be pushed to the configuration file.

Example 2

For example, if a configuration file specified that an application were to be threaded based on the existence of the key word “threaded” in the configuration file, the CML would look like this:

```
@?is_threaded@threaded
```

This sets the value at the name space key `/is_threaded` to true if the value “threaded” is in the configuration file and to false if the value “threaded” is not in the configuration file.

DTD Tag: @~

This tag defines a DTD.

Syntax

```
@~{source}
[type = {type}]
[description = {description}]
[printable = {printable}]
[range = {range}]
[{option}
...]
@
```

Description

CML supports Document Type Definition (DTD) tags that can be used to pre-define attributes for other CML tags. DTD’s can be used to make the actual functional part of the CML template a little cleaner by storing all of the characteristics of the tag in another location and just referencing the tag itself by name.

DTD definitions can be used to define any tag that has a source attribute; for example loop tags, loop target tags, replace tags, but not tags like instruction tags or group tags (which do not have a source attribute).

Another advantage of using DTD tags in CML is the ability to define 'PRINTABLE' and 'DESCRIPTION' values. The 'PRINTABLE' and 'DESCRIPTION' values give the user some feedback regarding the intended purpose of the field. The string value of the DESCRIPTION attribute is displayed when the mouse cursor rolls over the field in the value set editor screen. The string value of the Printable attribute will replace the path name in the value set editor with a easier-to-read field label.

DTD tags in CML are also inherently multi-line tags. All but the first and last line can be in any order, and all the elements here relate to the fields in a tag, except for printable and description, as those two are valid only for DTD defined tags.

For more information on using DTD tags in your configuration templates, see Using DTD Tags in CML on page 162. For XML templates, see Customizing XML DTD Element Display on page 65.

Attributes

No level attribute is used with a DTD tag. The only required attribute in a DTD tag is the source; everything else is optional. However, a DTD tag with only the name defined does nothing useful.

- **Source**: The source attribute is the database key used to access the value. If the source attribute is relative (does not start with a "/" or a ".") it gets appended to the current name space and becomes part of the key used to store the value read in by this tag. If the name is absolute (starts with a "/") it *is* the key, and the value gets stored under this key. If the source name starts with a ".", it is to be appended to the name space of whatever loop it is a part of. Typically a tag inside a loop should start with a ".".
- **Type**: The type attribute assigns certain predefined restrictions and error checking to different values, based on well-known types. The default type for replace tags is "string", which will match more or less anything.

  The full list of types is available at CML Type Attributes on page 144 of this document.
- **DESCRIPTION**: The value of the description attribute is a string that is a brief description of what kind of value this tag represents. This attribute will be displayed as mouse-over text in the SA Client value set editor.
- **PRINTABLE**: The value of the printable attribute is a string that is just a clean name for the variable. It will be displayed in the SA Client value set editor as the name for the attribute.
- **Range**: The range attribute allows you to set the range for the values. You need to keep in mind that all ranges will be used when reading in a file as well as when accepting values from a user. If you have a configuration file that has a value outside of the ranges you set in the template, then an exception will probably get thrown when parsing that file. It is best to use ranges that are correct based on the documentation for the configuration file.

  Ranges are described fully at CML Range Attributes on page 149.
- **Options**: The option attributes serve to modify or affect the behavior of the tag. Multiple options can be appended to the end most tags, separated by semicolons. You may append as many options as you need to the tag, everything after the third semicolon is considered an option, and every option is separated by semicolons. Options can also be used as instruction tag.

  Options are full described at CML Global Option Attributes on page 151 and CML Regular Option Attributes on page 153.

Example

```
@~port
type = port
```

```
range = 1024<=,<=2048
printable = Port
description = The port used for this application. It
should be a port number between 1024 and 2048
@
```

CML Type Attributes

CML attributes define and control the semantics of a CML tag. This section defines the types you can use in a CML template. Note that some types can be modified to represent a sequence of repeating values by appending “-set” or “-list” to the type. Some types can be modified to ignore the order of a sequence of repeating values by prepending “ordered-“ or “unordered-“ to the type.

The int Type

Int is a numeric type.

Syntax

```
@[{level}]{tag-type}[[{source}][;int][;[{range}][;{option}[;{option}]...]]]]@
```

Description

An Integer value …, -2, -1, 0, 1, 2, … (Z).

The decimal Type

Decimal is a numeric type.

Syntax

```
@[{level}]{tag-type}[[{source}][;decimal][;[{range}][;{option}[;{option}]...]
]]]@
```

Description

Decimal number.

The guid Type

Guid is a numeric type.

Syntax

```
@[{level}]{tag-type}[[{source}][;guid][;[{range}][;{option}[;{option}]...]]]]
@
```

Description

Globally Unique Identifier (GUID), 128-bit id.

The string Type

String is a non-numeric type.

Syntax:

```
@[{level}]{tag-type}[[{source}][;string][;[{range}][;{option}[;{option}]...]]
]]@
```

Description

String is the default type for all values if no other type is explicitly specified.

The quotedstring Type

Quotedstring is a non-numeric type.

Syntax

```
@[{level}]{tag-type}[[{source}][;quotedstring][;[{range}][;{option}[;{option}
]...]]]]@
```

Description

Quoted string.

The boolean Type

Boolean is a non-numeric type.

Syntax

```
@[{level}]{tag-type}[[{source}][;boolean][;[{range}][;{option}[;{option}]...]
]]]@
```

Description

Boolean.

The duration Type

Duration is a non-numeric type.

Syntax

```
@[{level}]{tag-type}[[{source}][;duration][;[{range}][;{option}[;{option}]...
]]]]@
```

Description

Duration.

The ipv6 Type

Ipv6 is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;ipv6][;[{range}][;{option}[;{option}]...]]]]
@
```

Description

IP v6 Address.

The ipv4 Type

Ipv4 is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;ipv4][;[{range}][;{option}[;{option}]...]]]]
@
```

Description

IP v4 Address.

The ip Type

Ip is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;ip][;[{range}][;{option}[;{option}]...]]]]@
```

Description

IP Address (ipv4 and ipv6).

The hostname Type

Hostname is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;hostname][;[{range}][;{option}[;{option}]...
]]]]@
```

Description

The name of a host server.

The host Type

Host is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;host][;[{range}][;{option}[;{option}]...]]]]
@
```

Description

Host IP Address or Hostname.

The network Type

Network is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;network][;[{range}][;{option}[;{option}]...]
]]]@
```

Description

IP v4 Network.

The port Type

Port is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;port][;[{range}][;{option}[;{option}]...]]]]
@
```

Description

TCP or UDP Port.

The user Type

User is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;user][;[{range}][;{option}[;{option}]...]]]]
@
```

Description

Username.

The group Type

Group is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;group][;[{range}][;{option}[;{option}]...]]]
]@
```

Description

Group name.

file – system specific type

Syntax

```
@[{level}]{tag-type}[[{source}][;file][;[{range}][;{option}[;{option}]...]]]]
@
```

Description

File name.

The dir Type

Dir is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;dir][;[{range}][;{option}[;{option}]...]]]]@
```

Description

Directory path name.

The email Type

Email is a system-specific type.

Syntax

```
@[{level}]{tag-type}[[{source}][;email][;[{range}][;{option}[;{option}]...]]]
]@
```

Description

Email Address.

CML Range Attributes

CML attributes define and control the semantics of a CML tag. This section defines the possible range attributes you can use in a CML template. For a given a CML type, range attributes allow you to define and restrict valid values for tag, using range specifiers.

! & , – Logical Operators

! – not specifier

& – and specifier

, – or specifier

Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;!{range}][;{option}[;{option}]...]
]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;{range}&{range}][;{option}[;{optio
n}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;{range},{range}][;{option}[;{optio
n}]...]]]]@
```

Description

Range specifiers can be modified by logical operators to control how input is validated. The three available operators (in order of precedence) are: not, and, or.

- The *not* operator is represented with an exclamation point, and is a prefix unary operator. It negates the meaning of the range, meaning that items that satisfy the range return false, and items that fail to satisfy the range return true.
- The *and* operator is represented with an ampersand, and is an infix binary operator. It returns true if and only if both operands return true.

- The *or* operator is represented with a comma, and is an infix binary operator. It returns true if and only if either operand returns true.

Whitespace is not significant when specifying ranges. (Note: The current CML parser requires that whitespace does not appear inside a tag).)

n< n<= <n <=n =n – Comparison Specifiers

n< – greater than specifier

n<= – greater than or equal specifier

<n – less than specifier

<=n – less than or equal specifier

=n – equal specifier

Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;{number}<][;{option}[;{option}]...
]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;{number}<=][;{option}[;{option}]..
.]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;<{number}][;{option}[;{option}]...
]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;<={number}][;{option}[;{option}]..
.]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;={number}][;{option}[;{option}]...
]]]]@
```

Description

The available specifiers for numeric values are: greater than, greater than or equal to, less than, less than or equal to, and equals.

- A greater than specifier (n<) consists of a number, followed by an open angle bracket character. This range is satisfied by numeric values that are greater than the specified number.
- A greater than or equal to specifier (n<=) consists of a number, followed by an open angle bracket character, followed by an equals character. This range is satisfied by numeric values that are greater than or equal to the specified number. (Note that for a number n, n<= is equivalent to !<n, and also equal to n<,=n, and is provided for convenience)
- A less than specifier (<n) consists of an open angle bracket character, followed by a number. This range is satisfied by numeric values that are greater than the specified number.
- A less than or equal to specifier (<=n) consists of an open angle bracket character, followed by an equals character followed by a number. This range is satisfied by numeric values that are greater than or equal to the specified number. (Note that for a number n, <=n is equivalent to !n<, and also equal to <n,=n, and is provided for convenience)
- An equals specifier (=n) consists of an equals character, followed by a number. This range is satisfied by numeric values that are equal to the specified number.

It is suggested that when providing two range specifiers separated by an and operator, the greater than (or equal to) specifier precede the less than (or equal to) specifier, for example, 0<=&<256.

Whitespace is not significant when specifying ranges. (Note: the current CML parser requires that whitespace does not appear inside a tag.)

" – String Literal Specifier

Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;"{string}"][;{option}[;{option}]..
.]]]]@
```

Description

A string literal specifier consists of a double quote character, followed by a string of text, followed by a double quote character. The quoting and escaping rules follow those of the C language; that is, that embedded quotes are escaped with a backslash, a newline is represented by `\n`, a tab character is represented by `\t`, and a literal backslash is represented by `\\`. This range is satisfied by string values that exactly match the text.

Whitespace is not significant when specifying ranges. (Note: the current CML Parser requires that whitespace does not appear inside a tag.)

r" – Regular Expression Specifier

Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;r"{regular
expression}"][;{option}[;{option}]...]]]]@
```

Description

A regular expression specifier consists of the “r” character, a double quote character, followed by a regular expression, followed by a double quote character (“). The quoting and escaping rules follow those of Python regular expressions, with the exception of the quote character, which must be escaped with a backslash character. This range is satisfied by string values that match the regular expression.

Whitespace is not significant when specifying ranges. (Note: The current CML parser requires that whitespace does not appear inside a tag.)

CML Global Option Attributes

CML attributes define and control the semantics of a CML tag. This section defines the possible global attributes you can use in a CML template. Global options can only be used in instruction tags, and cannot be used as option attributes in other tag types.

The @!filename-key Attribute

Syntax

`@!filename-key={key}@`

`{key}` has no default value.

Description

`filename-key` identifies a path to the key in a value set that will contain the file name of the file being generated during a push.

The `filename-key` value is a pathname. It must start with a slash (/).

As of SA 7.0, the filename-key value must not end with a /. This requirement may be relaxed in later versions.

The @!filename-default Attribute

Syntax

`@!filename-default={filename}@`

`{filename}` has no default value.

Description

`filename-default` identifies the default filename that will be returned if there is no filename in the Value Set. For example, the user may enter a filename in the Value-Set Editor, thus overriding the filename-default value.

The @!full-template and @!partial-template Attributes

Syntax

`@!full-template@`

`@!partial-template@`

`full-template` is the default behavior.

Description

`full-template` is the default behavior and indicates that all expected data in the file must be modeled in the template.

`partial-template` indicates that unmatched data in the file should be ignored and passed directly through to the output. This option only works with `preserve-format`.

The @!timeout Attribute

Syntax

`@!timeout={minutes}@`

`{minutes}` default value is 1.

Description

`timeout` represents the number of minutes that should be added onto the Configurations total timeout. A valid timeout is any integer from 0-999 (inclusive). The time-outs of all the templates in a configuration get added together, and that number is added to the default timeout for configurations (10 minutes) to get the final timeout value for the entire configuration.

See also Modifying Push Timeout Values on page 27.

CML Regular Option Attributes

CML attributes define and control the semantics of a CML tag. This section defines the possible option attributes you can use in a CML template. Regular options can be use either as Instruction tags or as Option attributes in other tag types.

The @! unordered-lines and @!ordered-lines Attributes

Instruction Tag Syntax:

```
@!unordered-lines@
@!ordered-lines@
```

`unordered-lines` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;unordered-lines[;{optio
n}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;ordered-lines[;{option}
]...]]]]@
```

Valid for groups.

Description

`unordered-lines` allows child tags of a template to appear in any order; however, position of items within ordered sequence elements is preserved. `unordered-lines` is the default behavior.

`ordered-lines` instructs the parser that child tags of the template object (lines, loops, conditionals, and so on) must appear in the file in the ordered they are specified in the template.

The unordered-elements and ordered-elements Attributes

Instruction Tag Syntax:

```
@!unordered-elements@
@!ordered-elements@
```

`unordered-elements` is the default behavior.

Option Attribute Syntax:

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;unordered-elements[;{op
tion}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;ordered-elements[;{opti
on}]...]]]]@
```

Valid for groups.

Description

`unordered-elements` allows child tags of of the current group to appear in any order; however, position of items within ordered sequence elements is preserved. `unordered-elements` is the default behavior.

`ordered-elements` instructs the parser that child tags of the group object (loops, conditionals, elements, and so on) must appear in the file in the ordered they are specified in the template.

The relaxed-whitespace and strict-whitespace Attributes

Instruction Tag Syntax

```
@!relaxed-whitespace@
@!strict-whitespace@
```

`relaxed-whitespace` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;relaxed-whitespace[;{op
tion}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;strict-whitespace[;{opt
ion}]...]]]]@
```

Valid for groups.

Description

`relaxed-whitespace` allows whitespace in the template to be matched by any combination of tabs and spaces. `relaxed-whitespace` is the default behavior.

`strict-whitespace` requires that white space in the template be matched exactly in the file.

The required-whitespace and optional-whitespace Attributes

Instruction Tag Syntax

```
@!required-whitespace@
@!optional-whitespace@
```

`required-whitespace` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;required-whitespace[;{o
ption}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;optional-whitespace[;{o
ption}]...]]]]@
```

Valid for groups.

Description

`required-whitespace` requires that whitespace in the template be in the file.
`optional-whitespace` makes the presence of non-significant whitespace in the file optional.

The missing-values-are-null and missing-values-are-error Attributes

Instruction Tag Syntax

```
@!missing-values-are-null@
```

```
@!missing-values-are-error@
```

`missing-values-are-null` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;missing-values-are-null
[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;missing-values-are-erro
r[;{option}]...]]]]@
```

Description

`missing-values-are-null` instructs that values that are not found in the file are null, and therefore not provided in the Value Set.

`missing-values-are-error` throws an error if all values specified in a template are not found in a file or Value Set.

The case-insensitive-keywords and case-sensitive-keywords Attributes

Instruction Tag Syntax

```
@!case-insensitive-keywords@
```

```
@!case-sensitive-keywords@
```

`case-insensitive-keywords` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;case-insensitive-keywor
ds[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;case-sensitive-keywords
[;{option}]...]]]]@
```

Description

`case-insensitive-keywords` match literal text in the file ignoring case. case-insensitive-keywords is the default behavior.

`case-sensitive-keywords` instructs that literal text in the template must be matched in a case-sensitive basis in the file.

The required and optional Attributes

Instruction Tag Syntax

```
@!required@
@!optional@
```

`required` is the default behavior.

Using `optional` in an instruction tag may have unintended consequences.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;required[;{option}]...]
]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;optional[;{option}]...]
]]]@
```

Description

`required` elements must be matched (unless nested inside optional groups). `required` is the default behavior.

`optional` elements are optional.

Using `optional` as an option attribute is valid for any tag, except an instruction tag. Using `optional` in an instruction tag may have unintended consequences.

The skip-lines-without-values and show-lines-without-values Attributes

Instruction Tag Syntax

```
@!skip-lines-without-values@
@!show-lines-without-values@
```

`skip-lines-without-values` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;skip-lines-without-valu
es[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;show-lines-without-valu
es[;{option}]...]]]]@
```

Description

`skip-lines-without-values` instructs when a line has replace elements, and all values for those elements are null, that line should be suppressed from the output. `skip-lines-without-values` is the default behavior.

`show-lines-without-values` instructs that all lines should be shown, regardless of the presence or absence of null values.

The skip-groups-without-values and show-groups-without-values Attributes

Instruction Tag Syntax

```
@!skip-groups-without-values@
@!show-groups-without-values@
```

`skip-groups-without-values` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;skip-groups-without-val
ues[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;show-groups-without-val
ues[;{option}]...]]]]@
```

Description

`skip-groups-without-values` instructs when a group has replace elements, and all values for those elements are null, that groups should be suppressed from the output. `skip-groups-without-values` is the default behavior.

`show-groups-without-values` instructs that all groups should be shown, regardless of the presence or absence of null values.

The sequence-append, sequence-replace and sequence-prepend Attributes

Instruction Tag Syntax

```
@!sequence-append@
@!sequence-replace@
@!sequence-prepend@
```

`sequence-append` is the default behavior.

Valid for loops and sequences.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-append[;{optio
n}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-replace[;{opti
on}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-prepend[;{opti
on}]...]]]]@
```

Description

`sequence-append` sequence elements child scopes are appended to sequence elements in parent scopes. `sequence-append` is the default behavior.

`sequence-replace` indicates that sequence elements child scopes replace sequence elements in parent scopes.

`sequence-prepend` sequence elements child scopes are prepended to sequence elements in parent scopes.

The not-primary-field and primary-field Attributes

Instruction Tag Syntax

```
@!not-primary-field@
@!primary-field@
```

`not-primary-field` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;not-primary-field[;{opt
ion}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;primary-field[;{option}
]...]]]]@
```

Description

`not-primary-field` indicates this field should not be used for the purposes of identifying duplicate items when performing list aggregation.

`not-primary-field` is the default behavior.

`primary-field` indicates this field should be used for the purposes of identifying duplicate items when performing list aggregation.

Valid for sequence and replace tags inside a sequence.

The namespace Attribute

Instruction Tag Syntax

```
@!namespace={namespace}@
```

The default value for `{namespace}` is “/” (the root name space).

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;namespace={namespace}[;
{option}]...]]]]@
```

The default value for `{namespace}` is “/” (the root name space).

Description

`namespace` is a string that identifies the name space within which elements with unqualified names (names without a preceding slash or period) will be stored.

The default value for `{namespace}` is the root name space, represented by the string “/” (forward-slash).

The name space value is a path name. It must start with a slash (/).

The boolean-no-format Attribute

Instruction Tag Syntax

```
@!boolean-no-format={string}@
```

The default value for {string} is “no”

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;boolean-no-format={stri
ng}[;{option}]...]]]]@
```

The default value for {string} is “no”

Description

`boolean-no-format` identifies the string that will be used to match false Boolean elements. Valid for Boolean replace tags.

The boolean-yes-format Attribute

Instruction Tag Syntax

```
@!boolean-yes-format={string}@
```

The default value for `{string}` is “yes”

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;boolean-yes-format={str
ing}[;{option}]...]]]]@
```

The default value for `{string}` is “yes”

Description

`boolean-yes-format` and `boolean-no-format` identifies the strings that will be used to match boolean elements. The default value for `{string}` is “yes”

Valid for boolean replace tags.

The line-comment Attributes

```
line-comment-is-comma
line-comment-is-semicolon
line-comment-is-tab
line-comment-is-whitespace
line-comment
```

Instruction Tag Syntax

```
@!line-comment-is-comma@
@!line-comment-is-semicolon@
@!line-comment-is-tab@
@!line-comment-is-whitespace@
@!line-comment={string}@
```

There is no default value for `{string}`.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;line-comment-is-comma[;
{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;line-comment-is-semicol
on[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;line-comment-is-tab[;{o
ption}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;line-comment-is-whitesp
ace[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;line-comment={string}[;
{option}]...]]]]@
```

There is no default value for `{string}`.

Description

`line-comment` sets the character that indicates that the remainder of the line will be parsed as a comment.

The sequence-delimiter Attribute

```
sequence-delimiter-is-comma
sequence-delimiter-issemicolon
sequence-delimiter-is-tab
sequence-delimiter-iswhitespace
sequence-delimiter
```

Instruction Tag Syntax

```
@!sequence-delimiter-is-comma@
@!sequence-delimiter-issemicolon@
@!sequence-delimiter-is-tab@
@!sequence-delimiter-iswhitespace@
@!sequence-delimiter={string}@
```

`sequence-delimiter-iswhitespace` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-delimiter-is-c
omma[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-delimiter-isse
micolon[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-delimiter-is-t
ab[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-delimiter-iswh
itespace[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-delimiter={str
ing}[;{option}]...]]]]@
```

`sequence-delimiter-iswhitespace` is the default behavior.

Description

`sequence-delimiter` sets the character that separates items within a sequence. `sequence-delimiter-iswhitespace` is the default behavior.

Valid for sequences.

The field-delimiter Attribute

```
field-delimiter-is-comma
field-delimiter-is-semicolon
field-delimiter-is-tab
field-delimiter-is-eol
field-delimiter-is-whitespace
field-delimiter
```

Instruction Tag Syntax

```
@!field-delimiter-is-comma@
@!field-delimiter-is-semicolon@
@!field-delimiter-is-tab@
@!field-delimiter-is-whitespace@
@!field-delimiter={string}@
```

`field-delimiter-is-whitespace` is the default behavior.

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;field-delimiter-is-comm
a[;{option}]...]]]]@
```

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;field-delimiter-is-semi
colon[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;field-delimiter-is-tab[
;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;field-delimiter-is-whit
espace[;{option}]...]]]]@
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;field-delimiter={string
}[;{option}]...]]]]@
```

`field-delimiter-is-whitespace` is the default behavior.

Description

`field-delimiter` sets a character that will be used to terminate parsing for a replace element value. `field-delimiter-is-whitespace` is the default behavior.

Valid for replace tags and sequence tags.

The line-continuation Attribute

Instruction Tag Syntax

```
@!line-continuation={string}@
```

Option Attribute Syntax

```
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;line-continuation={stri
ng}[;{option}]...]]]]@
```

Description

`line-continuation` sets a character that will be used to indicate that the current line in a config file should be wrapped to the subsequent line.

Using DTD Tags in CML

CML supports Document Type Definition (DTD) tags that can be used to pre-define attributes for a CML tag. Using a DTD tag in CML allows you to change some aspects of how the template is displayed in the SA Client. The DTD definition generally goes in the beginning of a file and the tag gets shortened to just a name and a tag type.

The main advantage of using DTD tags in CML is the ability to define 'printable' and 'description' values, which are reflected in the SA Client, improving usability. DTD definitions can be used to define any tag that has a name; for example loop tags, loop target tags, replace tags, and so on, but not tags like instruction tags or block tags. DTD tags in CML are also inherently multi-line tags.

DTD Tags Example

Here we will take a tag and create a DTD version of that tag. A DTD tag in CML is not that different than a regular CML tag; it contains all the elements of a tag minus the “tag type”.

For example, in the CML tag below:

`@*deny_header;unordered-string-set;;sequence-delimiter=":";optional@`

this is an instance representing the following format in CML:

`@<tag type><name>;<data type>;;<option1>;<option2>@`

The DTD version of this takes the existing elements and reorders them as follows:

```
<start code block>
@~<name>
type = <data type>
description = <description>
printable = <printable>
<option1>
<option2>
...
@
@<tag type><name>@
<end code block>
```

As you can see, this usage also allows for the addition of two new elements: “description” and “printable”. Defining “printable” will define the main text for this tag in the SA Client. Defining “description” will create a description for this value in the SA Client that is viewable when you move your mouse pointer over the field in the value set editor in the SA Client.

Here is the same tag in full DTD format:

```
<start code block>
@~deny_header
type = unordered-string-set
printable = Headers to Deny
description = This is a list of headers that IIS should deny
sequence-delimiter = ":"
optional
@
@*deny_header@
<end code block>
```

There are a couple things to notice in the example above. In defining a value for “description,” the value can span multiple lines, as long as the lines following the first line have whitespace as the first character.

Options go on a line by themselves, where you have `<option>=<value>` you need to insert spaces before and after the "=" sign.

Now, where ever you use the tag `@*deny_header@`, the parser will use the predefined DTD for all that tags' information.

➤ Redefining a DTD defined tag, `@*deny_header@`, by using a line like `@*deny_header;unordered-string-set@` will cause the CML template to become invalid.

➤ Note also that DTD style CML is not currently required, but is most obvious when viewing the Application Configuration the SA Client. If you don't use DTD tags you will not see the 'printable' and 'description' fields, instead you will only see the underlying variable name.

Sequence Aggregation

Because Application Configuration values can be set across many different levels in the Application Configuration inheritance hierarchy (also referred to as the inheritance scope), it is important that you be able control the way multiple sequence values are merged together when you push an Application Configuration on to a server.

ACM allows you to control the way sequence values are merged across inheritance scopes. This means that you can, for example, add some values to a sequence in the Customer scope, Group scope, and the Server scope, and all the values will be merged together to form the final sequence.

The manner in which sequence values are merged is controlled by special tags in the CML template, using three different sequence merge modes:

- Sequence Replace: Sequence values from more specific scopes completely replace those from less specific scopes. This occurs for both sequences of sets and lists.
- Sequence Append: For lists, values at more general scopes are appended (placed after) to those at more specific scopes. Duplicates, if present, are not removed. For sets, the behavior is the same, except duplicates are merged. For lists, duplicates are identified according to child elements marked with the primary-key tag, and then merged. For scalars, this is done by simply removing duplicate values, leaving only the value from the most specific scope (the last occurrence is the merged sequence). This is the default mode, and will be used if nothing else is specified.
- Sequence Prepend: Works the same as append, but values at more general scopes are preprended (placed before) to those at more specific scopes.

For example, with these two sets:

- “a, b” — At a more specific (inner) level of the inheritance scope, for example, server instance level.
- “c, d” — At a more general (outer) of the inheritance scope, for example, the server group level.

When the application configuration template is pushed onto the server, the merging results would be:

- Sequence replace: “a, b”
- Sequence append: “a, b, c, d”
- Sequence prepend: “c, d, a, b”

Sequence aggregation occurs not only between scopes, but also within a scope itself. This is evident if there are duplicate values within a sequence of namespaces.

Sequence Replace

In the Replace merge mode (CML tag "sequence-replace"), the contents of a sequence defined at a particular scope replace those of less specific scopes, and no merging is performed on the individual elements of the sequence.

For example, if the sequence-replace tag has been set for a list in an configuration template CML source, then values set for that list at the server instance level will override, or replace, those set at the group level and at the Application Configuration default values level.

For example, if a list in an etc/hosts file was defined at the group level (outer) as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine
/system/dns/host/2/ip           10.10.10.10
/system/dns/host/2/hostnames/1  loghost
```

And the same list was defined at the device scope (inner), as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine.mydomain.net
/system/dns/host/2/ip           10.10.10.100
/system/dns/host/2/hostnames/1  mailserver
```

If template had defined the /system/dns/host element with the sequence-replace tag, the final results of the configuration file on the server after the push would be:

```
127.0.0.1 localhost mymachine.mydomain.net
10.10.10.100 mailserver
```

Sequence Append

When the append list merge mode (CML tag "sequence-append") is used for sequences, the values at more general scopes are appended (placed after) those of more specific scopes. Sequence append mode is the default mode for merging list values. If nothing is specified in the CML of the template, the sequence append will be used.

If a list in an etc/hosts file was defined at the group level (outer) as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine
/system/dns/host/2/ip           10.10.10.10
/system/dns/host/2/hostnames/1  loghost
```

And the same list was defined at the device scope (inner), as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine.mydomain.net
/system/dns/host/2/ip           10.10.10.100
/system/dns/host/2/hostnames/1  mailserver
```

Using the value sets from the above example, if the /system/dns/host element was a list with the sequence-append tag set in the configuration template, the final results of the configuration file on the server after the push would be:

```
127.0.0.1 localhost mymachine.mydomain.net
10.10.10.100 mailserver
127.0.0.1 localhost mymachine
```

```
10.10.10.10 loghost
```

But since it is not allowable for a hosts file to contain duplicate entries, the/system/dns/host element will have to be flagged in the configuration template as a set rather than a list, because sets do not allow duplicates. To avoid duplication of the list values in the example, the configuration template author would use the Primary Key option.

Primary Key Option in Sequence Merging

When operating in append mode on sets, new values in more specific scopes are appended to those of less specific ones, and duplicate values are merged with the resulting value placed in the resulting sequence according to its position in the more specific scope.

How this affects merged sequence values depends on what kind of data is contained in the sequence:

- For elements in a sequence which are scalars, the value from the most specific scope is used. In other words, values at the server instance level would replace the values at the group level.
- For elements which are namespace sequences, the value is obtained by applying the merge mode specified for that element (in this example, append) based upon matching up the primary fields.

To avoid the duplication of the /system/dns/host/.ip value, the configuration template author would use the CML primary-key option. With this option set, ACM will treat entries with the same value for /system/dns/host/.ip as the same and merge their contents.

In the example above, the final results of the configuration file on the server after the push would be:

```
127.0.0.1 localhost mymachine.mydomain.net mymachine
10.10.10.100 mailserver
10.10.10.10 loghost
```

Since it is possible to have a set without primary keys, if there are scalars in the sequence, then an aggregation of all scalar values will be used as the primary key. If there are no scalars, then the aggregation of all values in the first sequence will be used as the primary key. Although this is an estimate, in most cases the values will be merged effectively. To ensure that the correct values are used as primary keys, we recommend that you always explicitly set the primary key in a sequence.

Sequence Prepend

When the append list merge mode (CML tag “sequence-prepend”) is used for sequences, the values at more general scopes are prepended (placed before) those those of more specific scopes.

For example, if a sequence in an etc/hosts file was defined at the group level (outer) as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine
```

```
/system/dns/host/2/ip          10.10.10.10
/system/dns/host/2/hostnames/1  loghost
```

And the same sequence was defined at the device scope (inner), as the following:

```
/system/dns/host/1/ip          127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine.mydomain.net
/system/dns/host/2/ip          10.10.10.100
/system/dns/host/2/hostnames/1  mailserver
```

If the /system/dns/host element was a set with the sequence-prepend tag set in the configuration template, the final results of the configuration file on the server after the push would be:

```
10.10.10.10 loghost
127.0.0.1 mymachine localhost mymachine.mydomain.net
10.10.10.100 mailserver
```

CML Grammar

Table 17 describes CML grammar illustrating several types of CML tags.

Table 17 CML Grammar

CML Tag/element	**Description**
replace-tag	"@" source [";" [type] [";" [range] *option]] "@"
data-definition-tag	"@~" source CRLF *def-line "@"
conditional-tag	"@" [group-level] "?" source [";" [type] [";" [range] *option]] "@"
loop-tag	“@" [group-level] "*" source [";" [type] [";" [range] *option]] "@"
loop-target-tag	"@.@"
block-tag	"@" [group-level] "[" *option "@"
block-termination-tag	"@" [group-level] "]@"
line-continuation-tag	"@\"
instruction-tag	"@!" *option "@"
single-line-comment	"@#" [string CRLF]
multi-line-comment	"@##" *[string / CRLF] "#@"
def-line	type-line / range-line / option-line / printable-line / desc-line
type-line	"type" WSP "=" WSP type-elem CRLF
range-line	"range" WSP "=" WSP range CRLF
option-line	option-elem CRLF

Table 17 CML Grammar (cont'd)

CML Tag/element	Description
`printable-line`	`"printable" WSP "=" WSP string CRLF`
`desc-line`	`"description" WSP "=" *[ WSP string CRLF ]`
`group-level`	`int`
`source`	`absolute-path / relative-path / local-path`
`absolute-path`	`"/" path-component* name`
`relative-path`	`[ path-component* ] name`
`path-component`	`( name / sequence-id ) "/"`
`sequence-id`	`int`
`local-path`	`"." name`
`name`	`string`
`type`	`sequence / type-elem`
`sequence`	`[ order "-" ] type-elem "-" sequence-elem`
`sequence-elem`	`"set" / "list"`
`type-elem`	`"int" / "string" / "ip" / "port" / "file" / etc...`
`order`	`ordered" / "unordered"`
`range`	`and-range *[ "," and-range ]`
`and-range`	`range-elem *[ "&" range-elem ]`
`range-elem`	`numeric-range / string range`
`numeric-range`	`gt-range / ge-range / lt-range / le-range / eq-range`
`string range`	`string-literal / regular-exp`
`gr-range`	`int ">"`
`ge-range`	`int ">="`
`lt-range`	`">" int`
`le-range`	`">=" int`
`eq-range`	`"=" int`
`string-literal`	`<"> string <">`
`regular-exp`	`"r" <"> string <">`
`option`	`";" option-elem`
`option-elem`	`option-name / option-nv`

Table 17 CML Grammar (cont'd)

CML Tag/element	Description
`option-nv`	`option-nv`
`option-name`	`string`
`option-value`	`string`