



Server Automation

Software Version: 10.51

Developer Guide

Document Release Date: February, 2017
Software Release Date: November, 2016


Hewlett Packard
Enterprise

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted rights legend

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 2000-2016 Hewlett Packard Enterprise Development LP

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Documentation updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to: <https://softwaresupport.hpe.com/>.

This site requires that you register for an HPE Passport and to sign in. To register for an HPE Passport ID, click **Register** on the HPE Software Support site or click **Create an Account** on the HPE Passport login page.

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

Support

Visit the HPE Software Support site at: <https://softwaresupport.hpe.com>.

This website provides contact information and details about the products, services, and support that HPE Software offers.

HPE 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 website to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HPE 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 HPE Passport user and to sign in. Many also require a support contract. To register for an HPE Passport ID, click **Register** on the HPE Support site or click **Create an Account** on the HPE Passport login page.

To find more information about access levels, go to: <https://softwaresupport.hpe.com/web/softwaresupport/access-levels>.

HPE Software Solutions Now accesses the HPESW Solution and Integration Portal website. This site enables you to explore HPE Product Solutions to meet your business needs, includes a full list of Integrations between HPE Products, as well as a listing of ITIL Processes. The URL for this website is <https://softwaresupport.hpe.com/>.

Contents

Introduction	13
Server Automation Platform	14
Overview of the Server Automation Platform	14
Components	15
Benefits of the SA Platform	21
SA Platform API design	23
Services	23
Objects in the API	24
Exceptions	25
Event Cache	25
Searches	26
Security	26
API Documentation and the Twister	27
Constant field values	27
Supported clients	28
Platform Developer Guide examples	28
SA CLI methods	29
Method invocation	29
Security	30
Mapping between API and SA CLI methods	30
Differences between SA CLI methods and Unix commands	30
SA CLI method tutorial	31
Format specifiers	36
Position of format specifiers	37
Default format specifiers	38
Examples of ID format specifier	38
Structure format specifier syntax	39
Examples of structure format specifier	39
Examples of directory format specifier	42
Value representation	42

SA objects in the OGFS	42
Object attributes	43
Custom attributes	43
Primitive values	44
Arrays	45
SA CLI method parameters and return values	46
Method context and the self parameter	46
Passing arguments on the command-line	47
Specifying the type of a parameter	48
Complex objects and arrays as parameters	48
Overloaded methods	48
Return values	49
Exit status	49
Search filters and SA CLI methods	50
Search syntax	50
Search examples	51
Finding servers	51
Finding other objects	53
Searchable attributes and valid operators	54
Sample scripts	54
create_custom_field.sh	55
create_device_group.sh	56
create_folder.sh	58
remediate_policy.sh	59
remove_custom_field.sh	61
schedule_audit_task.sh	62
Getting usage information on SA CLI methods	63
Listing services	63
Finding a service in the API documentation	64
Listing the methods of a service	64
Listing the parameters of a method	64
Getting information about a value object	65
Determining if an attribute can be modified	65
Determining if an attribute can be used in a filter query	66
Python API access with Pytwist	66

Setup for Pytwist	67
Pytwist examples	67
Virtualization Pytwist examples	72
Pytwist details	80
Automation Platform Extensions (APX)	82
Creating an APX	83
Program APXs	84
Web APXs	85
APX user roles	85
APX permissions	86
Permission escalation	87
APX structure	88
File structure	88
OGFS integration	88
APX interfaces - Defining categories of APX extensions	89
Implementing an interface	90
RightClickToRun interface	91
CoreAffinity interface	92
Using the Interface API	93
apxtool command	93
Syntax of apxtool	93
Using short and long command options	94
Creating a new APX - apxtool new	95
Usage	95
Deleting an APX - apxtool delete	96
Usage	96
Exporting an APX from SA - apxtool export	97
Usage	97
Importing an APX into SA - apxtool import	98
Usage	98
Querying APX information - apxtool query	99
Usage	99
Setting the current version of an APX - apxtool setcurrent	101
Usage	101
Error handling	102

APX files	102
APX configuration file - apx.cfg	103
APX permissions escalation configuration file - apx.perm	104
No escalation	105
All permissions	105
With escalation	105
Showing the progress of an APX	105
apxprogress command	106
Syntax of apxprogress	106
Example shell script that uses apxprogress	106
Viewing APX progress	107
Tutorial: Creating a Web application APX	107
Tutorial prerequisites	108
Setting permissions and creating the tutorial folder	108
Creating a new web application	109
Importing the new web application into SA	111
Running the new web application	111
Modifying the web application	113
Running the modified web application	114
Tutorial: Creating a program APX	114
Tutorial prerequisites	114
Setting permissions and creating the tutorial folder	115
Creating a new program APX	115
Importing the new APX into SA	117
Running the new APX	118
Modifying the APX	119
Running the modified APX	120
Viewing the APX progress in the Twister interface	120
Agent Tools	123
Installation requirements	124
Installation	124
Upgrading Agent Tools	125
Agent Tools scripts	126
Sample Agent Tool scripts	128
Microsoft Windows PowerShell - SA integration	129

Windows PowerShell integration with SA	130
Integrated PowerShell/SA cmdlets	130
Installation requirements	131
Installation	131
Microsoft Windows PowerShell integration with SA features	132
Sample sessions	133
Java RMI clients	146
Setup for Java RMI clients	147
Sample Java RMI	147
Possible issue on Windows	149
Web Services clients	149
Programming language bindings provided in this release	150
URLs for service locations and WSDLs	150
Security for Web Services clients	151
Overloaded operations	151
Java interface support	151
Unsupported data types	151
Invoke setDirtyAttributes when creating or updating VOs	152
Compatibility with SA Web Services API 2.2	153
Perl Web Services clients	153
Required software for Perl clients	153
Running the Perl demo program	154
Sample Perl code	155
Construction of Perl objects for Web Services	158
C# Web Services clients	161
Required software for C# clients	161
Obtaining the C# client stubs	162
Building the C# demo program	162
Running the C# demo program	163
Sample C# code	164
Password security with C#	166
Pluggable checks	167
Setup for pluggable checks	167
Pluggable check tutorial	168
Audit and remediation	175

Creating a pluggable check	177
Creating the audit policy	185
Document Type Definition (DTD) for config.xml file	186
Search filter syntax	193
Filter grammar	193
Rebuilding the Apache HTTP server and PHP	195
Extending the APX HTTP environment	195
Application Configuration	199
Managing XML configuration files	199
Example: Travel manager application and XML configuration file	200
Contents of the Travel Manager mysql.xml file	201
Contents of the Travel Manager mysql.xml DTD-based XML file	201
Non-DTD XML configuration templates	202
Non-DTD XML configuration template for mysql.xml	202
DTD-based XML configuration templates	203
XML-DTD configuration template for mysql.xml	204
Customize XML DTD element display	204
Explicit versus positional display settings	205
Add positional custom display settings	206
Add explicit custom display settings	206
Customize how elements display in the SA Client	207
XML configuration template settings	208
CML primer	210
Terminology	210
CML basic concepts	211
Combining tags on one line	214
Use case 1 - Simple Key=Value configuration file	214
Using the Replace instruction	214
Final CML template	217
Resulting value set	217
Use case 2 - Repeating values in the configuration file	218
Using the Loop instruction tag	218
Final CML	221
Resulting value set	221
Use case 3 - Complex repeating values in the configuration file	221

Final CML	222
Resulting value set	222
Partial templates	223
CML Reference	223
Configuration templates	224
CML overview	225
Structure of CML tags	225
Required CML tags	226
Example CML template for /etc/hosts	228
CML tag types	228
Comment Tag: @# and @##	229
Replace Tag: @	230
Instruction Tags: @!	231
Block (or Group) Tag: @[@...@]@	232
Loop Tag: @*	235
Example 2	237
Loop Target Tag: @.	237
Conditional Tag: @?	238
DTD Tag: @~	239
CML type attributes	241
The ip type	244
Syntax	246
Description	246
Syntax	246
CML range attributes	247
! & , – Logical operators	247
n< n<= <n <=n =n – Comparison specifiers	248
" – String literal specifier	249
r" – Regular expression specifier	249
CML global option attributes	250
The @!filename-key attribute	250
The @!filename-default attribute	250
The @!full-template and @!partial-template attributes	251
The @!timeout attribute	251
The @!unix-newlines and @!windows-newlines attributes	252

CML regular option attributes	252
The @! unordered-lines and @!ordered-lines attributes	252
The unordered-elements and ordered-elements attributes	253
The relaxed-whitespace and strict-whitespace attributes	254
The required-whitespace and optional-whitespace attributes	254
The missing-values-are-null and missing-values-are-error attributes	255
The case-insensitive-keywords and case-sensitive-keywords attributes	255
The reluctant attribute	256
The required and optional attributes	256
The skip-lines-without-values and show-lines-without-values attributes	257
The skip-groups-without-values and show-groups-without-values attributes	258
The sequence-append, sequence-replace and sequence-prepend attributes	258
The not-primary-field and primary-field attributes	259
The namespace attribute	260
The boolean-no-format attribute	260
The boolean-yes-format attribute	261
The delimiter attribute	261
The line-comment attributes	262
The sequence-delimiter attribute	263
The field-delimiter attribute	264
The line-continuation attribute	265
Use DTD tags in CML	266
Example of DTD tags	266
Sequence aggregation	267
Sequence replace	268
Sequence append	269
Sequence prepend	271
XML Tutorial 1 - Creating a non-DTD XML configuration template	273
Sample non-DTD XML mysql.xml file	274
1. Creating an XML configuration template	274
2. Adding XML settings	275

3. Creating an application configuration to contain the template	276
4. Attaching the Application Configuration to a managed server	277
5. Configuring Application Configuration settings for the server	278
6. Editing values and pushing the configuration	279
XML Tutorial 2 - Creating an XML-DTD configuration template	280
Sample Travel Manager DTD-based XML file: mysql.xml	280
Sample Travel Manager XML DTD file: mysql.dtd	281
1. Creating XML-DTD template in a text editor	281
2. Adding custom settings for element descriptions in the Value Set Editor	282
3. Importing the XML-DTD configuration file	284
4. Creating an Application Configuration object	285
5. Attaching the Application Configuration to a managed server	286
6. Importing values from the configuration file	287
7. Editing values and push the configuration	288
CML Tutorial 1 - Creating an Application Configuration for a simple web app server	289
1. Determining the configuration files to be managed	289
2. Creating a template for the configuration file	289
3. Creating an Application Configuration object	291
4. Adding the template file to the Application Configuration object	292
5. Attaching the Application Configuration object to servers	292
7. Comparing the actual configuration files with the configuration template	298
8. Pushing configuration changes to the server	299
CML Tutorial 2 - Creating a template of a web server configuration file	300
1. Analyzing the native configuration file and documentation	301
2. Creating a CML comment block	301
3. Creating CML setup instructions	302
4. Defining the [Options] section — Opening blocks	303
5. Defining the [AllowExtensions] section - Closing a block by opening a new block	307
6. Defining the [DenyExtensions] section	309
7. Defining the [AllowVerbs] and [DenyVerbs] sections	310
8. Defining the [DenyHeaders] section	310

9. Defining the [DenyURLSequences] section	312
10. Defining the [RequestLimits] section	313
11. Placing the template in an Application Configuration	315
Sample UrlScan.ini file	315
Complete url_scan_ini.tpl CML template	322
Send documentation feedback	325

Introduction

This section provides information Server Automation Platform and Application configuration file:

- ["Server Automation Platform" on the next page](#)
- ["Application Configuration " on page 199](#)

Server Automation Platform

The Server Automation Platform is a set of APIs and a runtime environment that facilitate the integration and extension of SA. The Server Automation Platform APIs expose core services such as audit compliance, Windows patch management, and OS provisioning. The runtime environment executes Global Shell scripts that can access the Global File System (OGFS).

This topic provides information about the following:

- ["Overview of the Server Automation Platform" below](#)
- ["SA Platform API design" on page 23](#)
- ["Supported clients" on page 28](#)
- ["SA CLI methods" on page 29](#)
- ["Python API access with Pytwist" on page 66](#)
- ["Automation Platform Extensions \(APX\)" on page 82](#)
- ["Agent Tools" on page 123](#)
- ["Microsoft Windows PowerShell - SA integration" on page 129](#)
- ["Java RMI clients" on page 146](#)
- ["Web Services clients" on page 149](#)
- ["Pluggable checks" on page 167](#)
- ["Search filter syntax" on page 193](#)
- ["Rebuilding the Apache HTTP server and PHP" on page 195](#)

Overview of the Server Automation Platform

Using the Server Automation Platform, you can perform the following tasks:

- Build new automation applications and extend SA to improve IT productivity and comply with your IT policies.

- Exchange information with other IT systems, such as existing monitoring, trouble ticketing, billing, and virtualization technology.
- Use the SA Model Repository to store and organize critical IT information about operations, environment, and assets.
- Automate the management of a wide range of applications and operating systems.
- Incorporate existing Unix and Windows scripts with SA, enabling the scripts to run in a secure, audited environment.

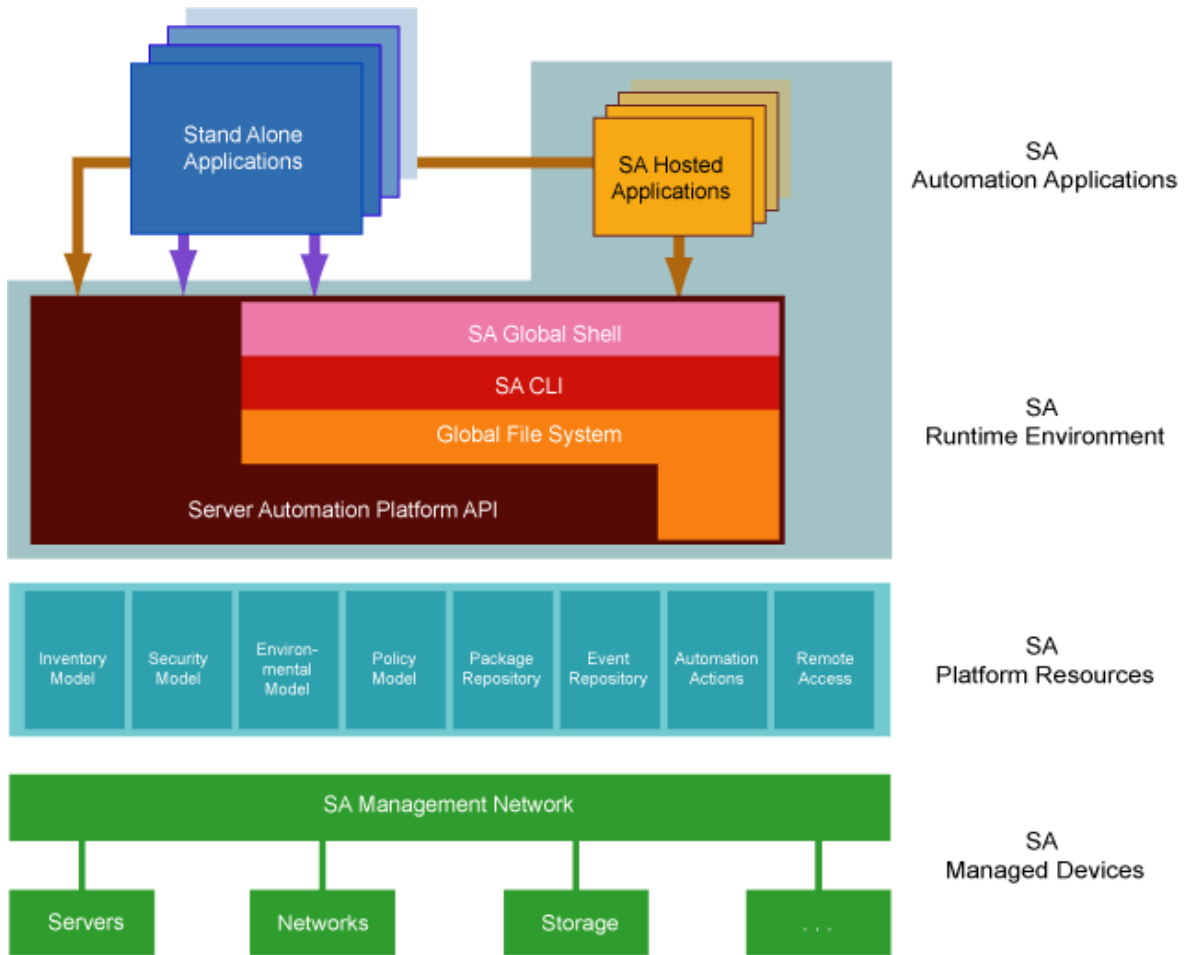
This topic provides an overview on the components and benefits of the Server Automation Platform:

- ["Components" below](#)
- ["Benefits of the SA Platform" on page 21](#)

Components

The following [figure](#) displays the major elements of the Server Automation Platform.

Server Automation Platform components



As the above figure shows, the platform comprises the following five key elements. Each of these elements is discussed in more detail in subsequent sections.

- ["Automation applications" below](#)
- ["SA runtime environment" on the next page](#)
- ["SA Platform resources" on page 18](#)
- ["SA Management Network" on page 20](#)
- ["SA Managed Devices" on page 21](#)

Automation applications

As in the figure above, the Automation Applications are at the top of the stack. These are the applications users write on top of the platform.

Automation applications can either be SA-Hosted Applications, which run in the SA Runtime Environment, or as standalone applications that run in a completely independent context. Standalone applications access the platform remotely through Web Services calls.

Simple applications can be written as simple Unix shell scripts in minutes. More complex applications—such as integration with an existing source control or ticketing system—can take a little longer and might involve Python or Microsoft .NET or Java coding. In either case, the platform is designed as a language-independent system easily adopted by a wide variety of developers.

SA runtime environment

Next down the platform stack is the SA Runtime Environment, which provides a set of powerful, out-of-the box runtime services and a corresponding language-independent programming model. SA-Hosted Applications run in the SA Runtime Environment.

The core of the runtime environment consists of two components that organize and provide access to all managed devices in a familiar Linux/Unix shell file-and-directory paradigm:

- **Global Shell:**

The Global Shell is a command-line interface to the Global File System (OGFS). The command-line interface is exposed through a Linux shell such as `bash` that runs in a terminal window. The OGFS unifies the SA data model and the contents of managed servers—including files—into a single, virtual file system.

- **Global File System:**

The OGFS represents objects in the platform data model (such as facilities, customers, and device groups) and information available on platform managed devices (such as the configuration setting on a managed network device or the file system of a managed server) as a hierarchical structure of file directories and text files. For example, in the OGFS, the `/opsw/Customer` directory contains details about customer objects and the `/opsw/Server` directory has information about managed servers. The `/opsw/Server` directory also contains subdirectories that reflect the contents (such as file systems and registries) of the managed servers.

This file-and-directory paradigm allows administrators familiar with shell scripting to easily write scripts which perform the same task across different servers by iterating through the directories that represent servers. Behind the scenes, the Global File System securely delivers and executes any logic in the script to each managed server.

The contents of devices can be accessed through the Global File System, a virtual file system that represents all devices managed by SA and Network Automation (NA). Given the necessary security authorizations, both end users and automation applications can navigate through the OGFS to the file systems of remote servers. On Windows servers, administrators can also access the registry, II metabase, and COM+ objects.

SA Command Line Interface

The SA Command Line Interface (CLI) provides system administrators and platform automation applications a way to invoke automation tasks such as provisioning software, patching devices, or running audits from the command line. A rich syntax allows users to represent rich object types as input or receive them as output from CLI invocations.

The CLI itself is actually programmatically generated on top of the platform API, discussed in the next section. The advantage of this is that as soon as developers add a new API to the platform API, a corresponding CLI method is automatically available for it. In other words, there is no lag time between the availability of new features in the product and the availability of the corresponding CLI methods in the platform.

SA Platform API

The SA Platform API is the Win32 API of SA: It defines a set of application programming interfaces to get and set values as well as perform actions. The SA user interfaces, including the SA Client and the SA Command Line Interfaces (CLI), are all built on top of the SA Platform API. The API includes libraries for Java RMI clients and WSDLs for SOAP-based Web Services clients. With Web Services support, programmers can create clients in popular languages such as Perl, C#, and Python.

SA Platform resources

SA Platform Resources sit beneath the SA Runtime Environment and give developers access to a rich set of objects and actions which they can re-use and manipulate in their own applications.

- **Inventory Model**

The Inventory Model provides all the information gathered by the SA about each managed devices such as make, manufacturer, CPU, operating system, installed software, and so on. Inventory information is made available through the SA API and also appears as files (in the `attr` subdirectories) in the Global File System. The Inventory Model includes objects such as Servers and Network Devices.

Administrators can extend the data associated with inventory objects. For example, if users want to store a picture of the device or a lease expiration date or the ID of a UPS the device is plugged into, the platform makes it easy to add those attributes to each device record. Users can then add, delete, and work with those attributes just as they would the attributes that come out of the box.

- **Security Model**

The Security Model allows developers to leverage the built-in SA authentication and authorization security systems.

All clients of the platform—management applications, scripts, as well as the end-user interfaces provided by SA are controlled by the same security framework.

The security administrator — not the developer — creates user roles and grants permissions. Developers can re-use all of these user roles and permissions in the context of their own applications. For example, network administrators can write a shell script and share it with other network administrators with the confidence that those network administrators can only run that script on network devices they are authorized to manage and no others.

The authorization mechanism controls access at several levels: the types of tasks users can perform, the servers and network devices accessed by the tasks, and the SA objects (such as software policies).

- **Environment Model**

The Environment Model defines the overall business context in which devices live. In general, devices belong to one or more customers, are located in a particular facility, and belong to one or more groups. The platform makes each of these objects — Customers Facilities, Device Groups, and others — available to application developers.

As with inventory objects, environment objects can easily be extended. This makes it easy, for example, to define attributes such as the SNMP trap receiver used in a particular data center or printers only available in a particular facility, or Apache configurations used by only a particular business unit.

- **Policy Model**

The Policy Model gives developers access to all the best practices defined in SA. Policies describe the desired state on a server or network device. For example, a patch policy describes the patches that should be on a server, a software policy describes what software should be on a server, and so on.

Subject matter experts define these policies which can be used by any authorized system administrator to audit devices to discover whether what's actually on a device differs from what should be on the device. Programmers have access to this complete library of policies to use in their own applications.

Software policies are organized into folders which can define security boundaries. In other words, applications will be able to access only those software policies they are permitted to access based on their user permissions.

- **Package Repository**

The Package Repository gives developers access to all the software and patches stored in SA. These include operating system builds, operating system patches, middleware, agents, and any other pieces of software that users have uploaded into SA.

- **Event Repository**

The Event Repository houses the digitally signed audit trails that the SA generates when actions are performed, either through the user interface or programmatically with the platform. As with other platform objects, these events are available programmatically.

- **Automation Actions**

Automation Actions allow developers to programmatically launch any of the actions that SA can perform on managed devices, ranging from running an audit to provisioning software to applying the latest OS patch.

The platform provides access to the same features available to end-users in the SA Client. These features include tasks such as installing patches, provisioning operating systems, and installing and removing software policies. In fact, the SA Client calls the same APIs that are exposed programmatically through the SA Runtime Environment.

- **Remote Access**

Remote Access gives developers programmatic access to the managed device's file system (in the case of servers) and execution environment (in the case of all devices). Developers can easily write applications which check for the existence of a file or particular software package, run operating system commands to check disk usage, or run system scripts to perform routine maintenance tasks.

SA Management Network

The Management Network is a powerful combination of technologies which enable developers to securely access any device under management. The Management Network delivers several key services:

- **Connectivity:** Allows the platform (and thus automation applications) to reach any managed device.
- **Security:** Includes SSL/TLS-based encryption, authentication, and message integrity.
- **Address space virtualization:** Enables the platform to locate servers across multiple overlapping IP address spaces. Most complex enterprise networks have multiple private IP address spaces.
- **Availability:** Allows system architectures to define redundant paths to any given managed device so that devices can still be reached despite failures in any given network path.
- **Caching:** Enables servers to download software and patches from a nearby server rather than a distant server, saving both time and network connectivity charges.
- **Bandwidth throttling:** Lets system architectures determine how much bandwidth SA and any SA applications can consume as it traverses the network to a particular device.

- **Least cost routing:** Allows system designers to set up rules governing which paths to use to reach a particular device to minimize network connectivity costs.

SA Managed Devices

At the bottom of the platform stack are the actual devices under management. The platform manages over 65 server OS versions and over 35 different network device vendors with thousands of device models/versions supported out of the box.

The list of supported devices is constantly being updated. Platform developers and script writers benefit directly from this device list since their automation applications can consistently reach an ever growing list of managed devices in the same, familiar platform programming environment.

Benefits of the SA Platform

The SA Platform has the following key benefits.

- ["Powerful security" below](#)
- ["Rich services" on the next page](#)
- ["Easily accessible to a broad spectrum of programmers" on the next page](#)

Powerful security

The platform delivers the following comprehensive security mechanisms so developers don't have to worry about providing them in their own applications.

- **Secure communication channels:** End-to-end communication from the automation applications out to the managed devices is encrypted and authenticated.
- **Role-based access control:** The platform respects the role-based access controls built into the SA so developers can easily share their applications with the confidence that they will run just on those devices that an administrator has been granted access to.
- **Digitally signed audit trail:** After an automation application runs, the platform generates a digitally signed audit trail capturing who ran the application, the time of the application execution, and the devices on which the application ran.
- **Comprehensive reach** The platform provides comprehensive reach across all devices so system administrators and developers don't have to worry about how to get to a device:
- **Market-leading platform coverage:** Supported devices include over 65 server OS versions and more than 1,000 network devices.

- **In any physical location:** The devices can be located anywhere in the world whether in a major data center or a retail store or a satellite of.ce.
- **In any IP address space:** The devices can belong to any IP address space, as the platform supports multiple overlapping IP address spaces.
- **In DMZs:** Devices can be located in DMZs or other difficult-to-access network spaces without requiring the developer or system administrator to worry about the details of reaching the device (for example, through a bastion host).

Rich services

The platform exposes practically all the relevant data and actions in the underlying automation system:

- **Rich data out-of-the-box:** Developers have easy access to a rich set of data generated in part by the platform itself (such as device inventory data and facility information) and in part by users interacting with the platform (such as device groups customers, best practices policies, and uploaded software, patches, and scripts). Developers can easily write applications to read and write this data.
- **Extensible data store:** Developers can easily extend the native platform objects to include their own data. Device inventory models can be extended to include attributes the platform does not natively discover. Customer and facility objects can be extended to include attributes that should guide the provisioning or auditing of devices related to that customer.
- **Automation tasks:** The platform exposes nearly all the capabilities of the underlying automation systems to developers: patching, provisioning, auditing, and others. This enables developers writing complex work flows that span multiple systems to simply call these actions from the context of an automation application.

Easily accessible to a broad spectrum of programmers

The platform is explicitly designed to appeal to a broad range of developers ranging from Unix shell and Visual Basic script writers to Perl and Python programmers to enterprise .NET or Java programmers. The platform's Runtime Services layer makes most platform objects available in a file-and-directory paradigm and most platform services available from a command-line interface (the SA CLI). This allows system administrators used to writing shell scripts to instantly use the platform without having to learn a new programming language and tool. They can get started with their favorite text editor, a familiar Unix shell, and then quickly develop scripts.

For more complicated applications and integration with existing systems, system programmers can use whatever programming tools and languages that have Web Services bindings.

SA Platform API design

The Platform API is defined by Java interfaces and organized into Java packages. To support a variety of client languages and remote access protocols, the API follows a function-oriented, call-by-value model.

- ["Services" below](#)
- ["Objects in the API" on the next page](#)
- ["Exceptions" on page 25](#)
- ["Event Cache" on page 25](#)
- ["Searches" on page 26](#)
- ["Security" on page 26](#)
- ["API Documentation and the Twister" on page 27](#)
- ["Constant field values" on page 27](#)

Services

In the Platform API, a service encapsulates a set of related functions. Each service is specified by a Java interface with a name ending in `Service`, such as `ServerService`, `FolderService`, and `JobService`.

Services are the entry points into the API. To access the API, clients invoke the methods defined by the server interface. For example, to retrieve a list of software installed on a managed server, a client invokes the `getInstalledSoftware` method of the `ServerService` interface. Examples of other `ServerService` methods are `checkDuplex`, `setPrimaryInterface`, and `changeCustomer`.

The SA Platform API contains over 70 services – too many to describe here. The following table lists a few of the services that you may want to try out first. For a full list of services, in a browser go to the URL shown in ["API Documentation and the Twister" on page 27](#).

Partial list of services of the SA API

Service name	Some of the operations provided by this service
<code>AuditTaskService</code>	Create, get, and run audit tasks.
<code>ConfigurationService</code>	Create application configurations, get the software policies using an

Partial list of services of the SA API, continued

Service name	Some of the operations provided by this service
	application configuration.
DeviceGroupService	Create device groups, assign devices to groups, get members of groups, set dynamic rules.
EventCacheService	Trigger actions such as updating a client-side cache of value objects. See "Event Cache" on the next page.
FolderService	Create folders, get children of folders, set customers of folders, move folders.
InstallProfileService	Create, get, and update OS installation profiles.
JobService	Get progress and results of jobs, cancel jobs, update job schedules.
NasConnectionService	Get host names of NA servers, run commands on NA servers.
NetworkDeviceService	Get information such as families, names, models, and types, according to specified search filters.
SequenceService	Create, get, and run OS sequences to install operating systems on servers.
ServerService	Get information about servers, reconcile (remediate) policies on servers (install software), get and set custom fields and attributes, execute OS sequences (install OS).
SoftwarePolicyService	Create software policies, assign policies to servers, get contents of policies, remediate (reconcile) policies with servers.
SolPatchService	Install and uninstall Solaris patches, add policy overrides.
VirtualColumnService	Manage custom fields and custom attributes.
WindowsPatchService	Install and uninstall Windows patches, add policy overrides.

Objects in the API

Although the SA Platform API is function-oriented, its design enables clients to create object-oriented libraries. The SA data model includes objects such as servers, folders, and customers. These are persistent objects; that is, they are stored in the Model Repository. In the API, these objects have the following items:

- A service that defines the object's behavior. For example, the methods of the `ServerService` specify the behavior of a managed server object.

- An object (identity) reference that represents an instance of a persistent object. For example, `ServerRef` is a reference that uniquely identifies a managed server. In the `ServerService`, the first parameter of most methods is `ServerRef`, which identifies the managed server operated on by the method. The `Id` attribute of a `ServerRef` is the primary key of the server object stored in the Model Repository.
- One or more value objects (VOs) that represent the data members (attributes, fields) of a persistent object. For example, `ServerVO` contains attributes such as `agentVersion` and `loopbackIP`. The attributes of `ServerHardwareVO` include `manufacturer`, `model`, and `assetTag`. Most attributes cannot be changed by client applications. If an attribute can be changed, then the API documentation for the setter method includes “Field can be set by clients.”

For performance reasons, update operations on persistent objects are coarse-grained. The `update` method of `ServerService`, for example, accepts the entire `ServerVO` as an argument, not individual attributes.

Exceptions

All of the API exceptions that are specific to SA are derived from one of the following exceptions:

`OpwareException` - Thrown when an application-level error occurs, such as when an end-user enters an illegal value that is passed along to a method. Typically, the client application can recover from this type of exception. Examples of exceptions derived from `OpwareException` are `NotFoundException`, `NotInFolderException`, and `JobNotScheduledException`.

`OpwareSystemException` - Thrown when an error occurs within SA. Usually, the SA Administrator must resolve the problem before the client application can run.

The following exceptions are related to security:

`AuthenticationException` - Thrown when an invalid SA user name or password is specified.

`AuthorizationException` - Thrown when the user does not have permission to perform an operation or access an object. For more information on permissions, see the SA 10.51 Administration Guide.

Event Cache

Some client applications need to keep local copies of SA objects. Accessed by clients through the `EventCacheService`, the cache contains events that describe the most recent change made to SA objects. Clients can periodically poll the cache to check whether objects have been created, updated,

or deleted. The cache maintains events over a configured sliding window of time. By default, events for the most recent two hours are maintained. To change the sliding window size, edit the Web Services Data Access Engine configuration file, as described in the Server Automation Administration Guide on the HPE SSO portal.

Searches

The search mechanism of the SA Platform API retrieves object references according to the attributes (fields) of value objects. For example, the `getServerRefs` method searches by attributes of the `ServerVO` value object. The `getServerRefs` method has the following signature:

```
public ServerRef[] getServerRefs(Filter filter)...
```

Each `get*Refs` method accepts the `filter` parameter, an object that specifies the search criteria. A `filter` parameter with a simple expression has the following syntax:

value-object.attribute operator value

(This syntax is simplified. For the full definition, see ["Filter grammar" on page 193.](#))

The following examples are `filter` parameters for the `getServerRefs` method:

```
ServerVO.hostName = "d04.example.com"  
ServerVO.model BEGINS_WITH "POWER"  
ServerVO.use IN "UNKNOWN" "PRODUCTION"
```

Complex expressions are allowed, for example:

```
(ServerVO.model BEGINS_WITH "POWER") AND (ServerVO.use = "UNKNOWN")
```

Not every attribute of a value object can be specified in a `filter` parameter. For example, `ServerVO.state` is allowed in a `filter` parameter, but `ServerVO.OsFlavor` is not. To find out which attributes are allowed, locate the value object in the API documentation and look for the comment, "Field can be used in a filter query."

Security

Users of the SA Platform must be authenticated and authorized to invoke methods on the SA Automation Platform API. To connect to SA, a client supplies an SA user name and password (authentication). To invoke methods, the SA user must belong to a user group with the necessary permissions (authorization). These permissions restrict not only the types of operations that users can perform, but also limit access to the servers and network devices used in the operations.

Before application clients can run on the platform, the SA Administrator must specify the required users and permissions with the Command Center. For instructions, see the "User Group and Setup" section in the SA 10.51 Administration Guide. For information about security-related exceptions, see ["Exceptions" on page 25](#).

Communication between clients and SA is encrypted. For Web Services clients, the request and response SOAP messages (which implement the operation calls) are encrypted using SSL over HTTP (HTTPS).

API Documentation and the Twister

SA includes API documentation (Javadocs) that describe the SA Platform API. To access the API documentation, specify the following URL in a browser:

```
https://<SA_core_host>/twister
```

The `<SA_core_host>` is the IP address or host name of the SA core server running the Command Center component.

The *Twister* is a program that lets you invoke API methods, one at a time, from within a browser. For example, to invoke the `ServerService.getServerVO` method, perform the following steps:

1. Open the API documentation in a browser.
2. In the All Classes pane, select `com.opsware.server`.
3. In the `com.opsware.server` pane, select `ServerService`.
4. In the main pane, scroll down to the `getServerVO` method.
5. Click **Try It** for the `getServerVO` method.
6. Enter your SA user name and password.
7. In the Twister pane for `ServerService.getServerVO`, enter the ID of a managed server in the `oid` field.
8. Click **Go**. The Twister pane displays the attributes of the `ServerVO` object returned.

Constant field values

Some of the API's value objects (VOs) have fields with values defined as constants. For example, `JobInfoVO` has a `status` field that can have a value defined by constants such as `STATUS_ACTIVE`, `STATUS_PENDING`, and so forth. The API specifies constants as Java `static final` fields, but the

WSDLs generated from the API do not define the constants. To view the definitions for constants, in the API documentation, go to the Constant Field Values page:

https://<SA_core_host>/twister/docs/constant-values.html

For example, the Constant Field Values page defines `STATUS_ACTIVE` as the integer 1.

Supported clients

The SA platform supports programmers with different skills, from system administrators who write shell scripts to .NET and Java programmers familiar with the latest tools and technologies. All supported clients call the same set of methods, which are organized into the services of the SA Platform. A developer can create the following types of clients that call methods in the SA Platform API:

- **SA Command-Line Interface (CLI):** Launched from Global Shell sessions, shell scripts can access the SA Platform API by invoking the CLI methods, which are executable programs in the OGFS. Each CLI method corresponds to a method in the API.
- **Web Services:** Using SOAP over HTTPS, these clients send requests to SA and get responses back. The Web Services operations (defined in WSDLs) correspond to the methods in the API. Developers can write Web Services clients in popular languages such as Perl and C#.
- **Java RMI:** These clients invoke remote Java objects from other Java virtual machines.
- **Pytwist:** These Python programs can run on an SA Core or managed servers.

The Web Services and Java RMI clients can run on servers different than the SA Core or managed servers. The CLI methods execute in a Global Shell session on the core server where the OGFS is installed.

Platform Developer Guide examples

The **Platform Developer Guide examples** file is a ZIP archive containing sample codes for illustrating different techniques for developing software that makes use of the API provided by Server Automation. The ZIP file contains the sample codes provided in this document along with other referred samples.

You can download the **Platform Developer Guide examples** ZIP archive from <https://softwaresupport.hpe.com/km/KM00417670>. The zip file is also bundled with the **All Manuals Download SA 10.5** archive available on the support site.

SA CLI methods

End-users access SA through the SA Client. At times, advanced users need to access SA in a command-line environment to perform bulk operations or repetitive tasks on multiple servers. In SA, the command-line environment consists of the Global Shell (OGSH), Global File System (OGFS), and SA Command-Line Interface (CLI) methods.

To perform SA operations from the command line, you invoke the SA CLI methods from within an OGSH session. An SA CLI method is an executable in the OGFS that corresponds to a method in the SA API. When you run an SA CLI method, the underlying API method is invoked.

To understand this section, you should be familiar with the OGSH and the OGFS. For more information, see the OGSH in the Server Automation Using Guide on the HPE SSO portal.

For information on the `upload` and `download` commands, see the OCLI 1.0 in the Server Automation Using Guide on the HPE SSO portal.

Method invocation

As shown in the following [figure](#), when you invoke an SA CLI method in an OGSH session, the following operations occur:

1. The OGSH parses the command and parameters you entered to determine the API method.
2. The OGSH invokes the underlying API method.
3. An authorization check verifies that the user has permission to perform this operation. SA then performs the operation.
4. The API method passes the results back to the SA CLI method.
5. The SA CLI method writes the return value to the `stdout` of the OGSH session. If an exception was thrown, the SA CLI method returns a non-zero status.

Overview of an SA CLI method invocation



Security

SA CLI methods use the same authentication and authorization mechanisms as the SA Client. When you start an OGS session, SA authenticates your SA user. When you run an SA CLI method, authorization is performed. To run an SA CLI method successfully, your SA user must belong to a group that has the required permissions. For more information on security, see the Server Automation Administration Guide on the HPE SSO portal.

Mapping between API and SA CLI methods

The OGFS represents SA objects as directory structures, object attributes as text files, and API methods as executables. These executables are the SA CLI methods. Every SA CLI method matches an underlying API method. The method name, parameters, and return value are the same for both types of methods.

For example, the `setCustomer` API method has the following Java signature:

```
public void setCustomer(ServerRef self,  
                        CustomerRef customer)...
```

In the OGFS, the corresponding SA CLI method has the following syntax:

```
setCustomer self:i=server-id customer:i=customer-id
```

Note that the parameter names, `self` and `customer`, are the same in both languages. (The `:i` notations are called format specifiers, which are discussed later in this section.) In this example, the return type is `void`, so the SA CLI method does not write the result to the `stdout`. For information on how SA CLI methods return strings that represent objects, see ["Return values " on page 49](#).

Differences between SA CLI methods and Unix commands

Although you can run both Unix commands and SA CLI methods in the OGS, SA CLI methods differ in several ways:

- Unlike many Unix commands, SA CLI methods do not read data from `stdin`. Therefore, you cannot insert an SA CLI method within a group of commands connected by pipes (`|`). (However, SA CLI methods do write to `stdout`.)
- Most Unix commands accept parameters as flags and values (for example, `ls -l /usr`). With SA CLI methods, command-line parameters are name-value pairs, joined by equal signs.
- Unix commands are text based: They accept and return data as strings. In contrast, SA CLI methods can accept and return complex objects.
- With SA CLI methods, you can specify the format of the parameter and return values. Unix commands do not have an equivalent feature.

SA CLI method tutorial

This topic introduces you to the SA CLI methods with examples you can try in your own environment. After completing this tutorial, you should be able to run SA CLI methods, examine the `self` file of an SA object, and create a script that invokes SA CLI methods on multiple servers.

Before starting the tutorial, you need the following capabilities:

- You can log on to the SA Client.
- Your SA user has Read & Write permissions on at least one managed server. Typically assigned by a security administrator, permissions are discussed in the Server Automation Administration Guide on the HPE SSO portal.
- Your SA user has all OGSF permissions on the same managed server. For information on these permissions, see the “aaa Utility” section in the Server Automation Using Guide on the HPE SSO portal.
- You are familiar with the OGSF and the OGFS. If these features are new to you, before proceeding with this tutorial, see the Global Shell section in the Server Automation Using Guide on the HPE SSO portal.

The example commands in this tutorial operate on a Windows server named `abc.example.com`. This server belongs to a server group named All Windows Servers. When trying out these commands, substitute `abc.example.com` with the host name of the managed server you have permission to access.

1. Open an OGSF session.

You can open a Global Shell session from within the SA Client. From the **Actions** menu, select **Global Shell**. You can also open an OGSH session from a terminal client running on your desktop. For instructions, see [Opening a Global Shell Session](#) section in the Server Automation Using Guide on the HPE SSO portal.

2. List the SA CLI methods for a server.

The method subdirectory of a specific server contains executable files—the methods you can run for that server. The following example lists the SA CLI methods for the `abc.example.com` server:

```
$ cd /opsw/Server/@/abc.example.com/method
$ ls -l
addDeviceGroups
attachPolicies
attachVirtualColumn
checkDuplex
clearCustAttrs
...
```

These methods have instance context – they act on a specific server instance (in this case, `abc.example.com`). The server instance can be inferred from the path of the method. Methods with static context are discussed in [step 5](#).

3. Run an SA CLI method without parameters.

To display the public server groups that `abc.example.com` belongs to, invoke the `getDeviceGroups` method:

```
$ cd /opsw/Server/@/abc.example.com/method
$ ./getDeviceGroups
Accounting App
All Windows Servers
Visalia Vendors
```

4. Run a method with a parameter.

Command-line parameters for methods are indicated by name-value pairs, separated by white space characters. In the following invocation of `setCustomer`, the parameter name is `customer` and the value is `20039`. The `:i` at the end of the parameter name is an ID format specifier, which is discussed in a later step.

The following method invocation changes the customer of the `abc.example.com` server from Opsware to C39. The ID of customer C39 is 20039.

```
$ cd /opsw/Server/@/abc.example.com
$ cat attr/customer ; echo
Opsware
$ method/setCustomer customer:i=20039
$ cat attr/customer ; echo
C39
```

5. List the static context methods for managed servers.

Static context methods reside under the `/opsw/api` directory. These methods are not limited to a specific instance of an object.

To list the static methods for servers, enter the following commands:

```
$ cd /opsw/api/com/opsware/server/ServerService/method
$ ls
```

The methods listed are the same as those displayed in [step 2](#).

6. Run a method with the `self` parameter.

This step invokes `getDeviceGroups` as a static context method. Unlike the instance context method shown in [step 3](#), the static context method requires the `self` parameter to identify the server instance.

For example, suppose that the `abc.example.com` server has an ID of 530039. To list the groups of this server, enter the following commands:

```
$ cd /opsw/api/com/opsware/server/ServerService/method
$ ./getDeviceGroups self:i=530039
Accounting App
All Windows Servers
Visalia Vendors
```

Compare this invocation of `getDeviceGroups` with the invocation in [step 3](#) that demonstrates instance context. Both invocations run the same underlying method in the API and return the same results.

7. Examine the `self` file of a server.

Within SA, each managed server is an object. However, OGFS is a file system, not an object model. The `self` file provides access to various representations of an SA object. These representations are the ID, name, and structure.

The default representation for a server is its name. For example, to display the name of a server, enter the following commands:

```
$ cd /opsw/Server/@/abc.example.com
$ cat self ; echo
abc.example.com
```

If you know the ID of a server, you can get the name from the `self` file, as in the following example:

```
$ cat /opsw/.Server.ID/530039/self ; echo
abc.example.com
```

8. Indicate an ID format specifier on a `self` file.

To select a particular representation of the `self` file, enter a period, then the file name, followed by the format specifier. For example, the following `cat` command includes the format specifier (`:i`) to display the server ID:

```
$ cd /opsw/Server/@/abc.example.com
$ cat .self:i ; echo
com.opsware.server.ServerRef:530039
```

This output shows that the ID of `abc.example.com` is 530039. The `com.opsware.server.ServerRef` is the class name of a server reference, the corresponding object in the SA API.

Note: The leading period is required with format specifiers on files and method return values, but is not indicated with method parameters.

9. Indicate the structure format specifier.

The structure format specifier (`:s`) indicates the attributes of a complex object. The attributes are displayed as name-value pairs, all enclosed in curly braces. Structure formats are used to specify

method parameters on the command-line that are complex objects. For an example method call, see ["Complex objects and arrays as parameters" on page 48](#).

The following example displays `abc.example.com` with the structure format:

```
$ cd /opsw/Server/@/abc.example.com
$ cat .self:s ; echo
{
managementIP="192.168.8.217"
modifiedBy="spujare"
manufacturer="DELL COMPUTER CORPORATION"
use="UNKNOWN"
discoveredDate=1149012848000
origin="ASSIMILATED"
osSPVersion="SP4"
locale="English_United States.1252"
reporting=false
netBIOSName=
previousSWReg=1150673874000
osFlavor="Windows 2000 Advanced Server"
. . .
```

The attributes of a server are also represented by the files in the `attr` directory, for example:

```
$ pwd
/opsw/Server/@/abc.example.com
$ cat attr/osFlavor ; echo
Windows 2000 Advanced Server
```

10. Create a script that invokes an SA CLI method.

The example script shown in this step iterates through the servers of the public server group named All Windows Servers. On each server, the script runs the `getCommCheckTime` SA CLI method.

First, return to your home directory in the OGFS:

```
$ cd
$ cd public/bin
```

Next, run the `vi` editor:

```
$ vi
```

In `vi`, insert the following lines to create a bash script:

```
#!/bin/bash
# iterate_time.sh

METHOD_DIR="/opsw/api/com/opsware/server/ServerService/method"
GROUP_NAME="All Windows Servers"
cd "/opsw/Group/Public/$GROUP_NAME/@/Server"

for SERVER_NAME in *
do
  SERVER_ID=`cat $SERVER_NAME/.self:i`
  echo $SERVER_NAME
  $METHOD_DIR/getCommCheckTime self:i=$SERVER_ID
  echo
  echo
done
```

Save the file in `vi`, naming it `iterate_time.sh`. Quit `vi`.

Change the permissions of `iterate_time.sh` with `chmod`, and then run it:

```
$ chmod 755 iterate_time.sh
$ ./iterate_time.sh
abc.example.com
2006/06/20 16:46:56.000
. . .
```

Format specifiers

Format specifiers indicate how values are displayed or interpreted in the SA CLI environment. You can apply a format specifier to a method parameter, a method return type, the `self` file, and an object attribute. To indicate a format specifier, append a colon followed by one of the letters shown in the following table.

If a format specifier is indicated for a file or a method return value, a period must precede the file or method name. For method return values that have format specifiers, the leading period is not included.

Summary of format specifiers

Format specifier	Description	Valid object types	Allowed as method parameter?
:n	Name: A string identifying the object. Unique names are preferred, but not required. For objects that do not have a name, this representation is the same as the ID representation.	SA objects	Yes. If the name is ambiguous, an error occurs.
:i	ID: A format that uniquely identifies the object type and its SA ID. Also known as an object reference.	SA objects; Dates (<code>java.util.Calendar</code>) objects	Yes. If the type is clear from the context, the type may be omitted.
:s	Structure: A compact representation intended for specifying complex values on the command-line. Attributes are enclosed in curly braces.	Any complex object	Yes
:d	Directory: Represents an attribute as a directory in the OGFS.	Any complex object that is an attribute. This representation cannot be used for method parameters or return values.	No

Position of format specifiers

A format specifier immediately follows the item it affects. For files, a format specifier follows the file name. In the following example, note the leading period:

```
cat .self:s
```

When applied to a method return type, a format specifier follows the method name. The following invocation displays the IDs of the groups returned:

```
./getDeviceGroups:i
```

With method parameters, a format specifier follows the parameter name and precedes the equal sign, as in the following example:

```
./setCustomer self:i=9977 customer:i=239
```

A method parameter with a format specifier does not have a leading period.

Default format specifiers

Every value or object has a default format specifier. For example, the name format specifier is the default for the `osVersion` attribute. The following two `cat` commands generate the same output:

```
cd /opsw/Server/@/d04.example.com/attr
cat osVersion
cat .osVersion:n
```

The name format specifier is the default for SA objects stored in the Model Repository, such as servers and customers. The structure format specifier is the default for other complex objects.

Examples of ID format specifier

The next example displays the ID of the facility that the `d04.example.com` server belongs to:

```
cd /opsw/Server/@/d04.example.com/attr
cat .facility:i ; echo
```

(The preceding `echo` command is optional. It generates a new-line character, which makes the output easier to read. The semicolon separates `bash` statements entered on the same line.)

The output of a value with the ID format specifier is prefixed by the Java class name. For example, if the facility value has an ID of 39, then the previous `cat` command displays the following output:

```
com.opsware.locality.FacilityRef:39
```

The following invocation of the `getDeviceGroups` method lists the IDs of the public server groups that `d04.example.com` belongs to:

```
cd /opsw/Server/@/d04.example.com/method
./getDeviceGroups:i
```

For more ID format examples, see ["The self file" on page 44](#).

Structure format specifier syntax

The structure format represents complex objects, which can contain various attributes. You might use this format to specify a method parameter that is a complex object. For examples, see "[Complex objects and arrays as parameters](#)" on page 48.

The structure format is a series of name-value pairs, separated by white space characters, enclosed in curly braces. Each name-value pair represents an attribute. The structure format has the following syntax:

```
{ name-1=value-1 name-2=value-2 . . . }
```

Here's a simple example:

```
{ version=10.1.3 isCurrent=true }
```

Any white space character can be used as a delimiter:

```
{  
    version=10.1.3  
    isCurrent=true  
}
```

Attributes can be specified as structures, enabling the representation of nested objects. In the following example, the `versionDesc` attribute is represented as a structure:

```
{  
program=agent  
versionDesc={  
    version=10.1.3  
    isCurrent=true  
    comment="Latest version"  
}  
}
```

To specify an array within a structure, repeat the attribute name. The following structure contains an array named `steps` that has three elements with the values 33, 14, and 28.

```
{ moduleName="Some Initiator" steps=33 steps=14 steps=28 }
```

Examples of structure format specifier

The following example specifies the structure format for the `facility` attribute:

```
cd /opsw/Server/@/d04.example.com/attr
cat .facility:s
```

This `cat` command generates the following output. Note that `customers` is an array, which contains an element for every customer associated with this facility.

```
{
  modifiedBy="192.168.9.246"
  customers="Customer Independent"
  customers="Not Assigned"
  customers="Opsware Inc."
  customers="Acme Inc."
  . . .
  ontogeny="PROD"
  createdBy=
  status="ACTIVE"
  createdDt=-1
  realms="Transitional"
  realms="C39"
  realms="C39-agents"
  modifiedDt=1146528752000
  name="C39"
  displayName="C39"
}
```

The following invocation of `getDeviceGroups` indicates the structure format specifier for the return value:

```
cd /opsw/Server/@/d04.example.com/method
./getDeviceGroups:s
```

This call to `getDeviceGroups` displays the following output. Because `d04.example.com` belongs to two server groups, the output includes two structures. In each structure, the `devices` array has elements for the servers belonging to that group.

```
{
  dynamic=true
  devices="m302-w2k-vm1.dev.example.com"
  devices="d04.example.com"
  . . .
  status="ACTIVE"
  34 Chapter 2
  public=true
  fullName="Device Groups Public All Windows Servers"
  description="test"
  createdDt=-1
  modifiedDt=1142019861000
  parent="Public"
}
```



```
{
    dynamic=true
    devices="opsware-nibwp.build.example.com"
    devices="glengarriff.snv1.dev.example.com"
    devices="millstreet"
    . . .
    fullName="Device Groups Public z_testsrvgroup"
    . . .
}
```

The structure format specifier is the default for methods that retrieve value objects (VOs). For example, the following two calls to `getServerVO` are equivalent:

```
cd /opsw/Server/@/d04.example.com/method
./getServerVO:s./getServerVO
```

In this example, `getServerVO` displays the following output:

```
{
    managementIP="192.168.198.93"
    modifiedBy=
    manufacturer="DELL COMPUTER CORPORATION"
    use="UNKNOWN"
    discoveredDate=1145308867000
    origin="ASSIMILATED"
    osSPVersion="RTM"
    locale="English_United States.1252"
    reporting=false
    netBIOSName=
    previousSWReg=1147678609000
    osFlavor="Windows Server 2003, Standard Edition"
    peerIP="192.168.198.93"
    modifiedDt=1145308868000
    . . .
    serialNumber="HVKZS51"
}
```

This structure represents the `ServerVO` class of the SA API. Every attribute in this structure corresponds to a file in the `attr` directory. In the next example, the `getServerVO` and `cat` commands both display the value of the `serialNumber` attribute of a server:

```
cd /opsw/Server/@/d04.example.com
./method/getServerVO | grep serialNumber
cat attr/serialNumber ; echo
```

Examples of directory format specifier

The following command changes the current working directory to the customer associated with the server `d04.example.com`:

```
cd /opsw/Server/@/d04.example.com/attr/.customer:d
```

The next command lists the name of this customer:

```
cat /opsw/Server/@/d04.example.com/attr/  
.customer:d/attr/name
```

The directory specifier can be used only in command arguments that require directory names. The following `cat` command fails because it attempts to display a directory:

```
cat /opsw/Server/@/d04.example.com/attr/.customer:d # WRONG!
```

However, the next command is legal:

```
ls /opsw/Server/@/d04.example.com/attr/.customer:d
```

Value representation

Because they run in a shell environment (the OGSHELL), SA CLI methods accept and return data as strings. However, the underlying API methods can accept and return other data types, such as numbers, Booleans, and objects. The sections that follow describe how the OGFS and SA CLI methods represent non-string data types.

SA objects in the OGFS

The SA data model includes objects such as servers, server groups, customers, and facilities. In the OGFS, these objects are represented as directory structures:

```
/opsw/Customer  
/opsw/Facility  
/opsw/Group  
/opsw/Library  
/opsw/Realm  
/opsw/Server  
. . .
```

The preceding list is not complete. To see the full list, enter `ls /opsw`.

Object attributes

The attributes of an SA object are represented by text files in the `attr` subdirectory. The name of each file matches the name of the attribute. The contents of a file reveals the value of the attribute.

For example, the `/opsw/Server/@/buzz.example.com/attr` directory contains the following files:

```
agentVersion
codeset
createdBy
createdDt
customer
defaultGw
36 Chapter 2
description
discoveredDate
facility
hostName
locale
lockInfo
loopbackIP
managementIP
manufacturer
. . .
```

To display the management IP address of the `buzz.example.com` server, enter the following commands:

```
cd /opsw/Server/@/buzz.example.com/attr
cat managementIP ; echo
```

Custom attributes

Custom attributes are name-value pairs that you can assign to SA objects such as servers. In the OGFS, custom attributes are represented as text files in the `CustAttr` subdirectory. You can create custom attributes in an OGSH session by creating new text files under `CustAttr`. The following example creates a custom attribute named `MyGreeting`, with a value of `hello there`, on the `buzz.example.com` server:

```
cd /opsw/Server/@/buzz.example.com/CustAttr
echo -n "hello there" > MyGreeting
```

For more examples, see the “Managing Custom Attributes” section in the SA 10.51 User Guide.

The self file

The `self` file resides in the directory of an SA object such as a server or customer. This file provides access to various representations of the current object, depending on the format specifier. For details, see ["Format specifiers" on page 36](#).

To list the ID of the `buzz.example.com` server, enter the following commands:

```
cd /opsw/Server/@/buzz.example.com
cat .self:i ; echo
```

For a server, the default format specifier is the name. The following commands display the same output:

```
cat self ; echo
cat .self:n ; echo
```

The next command lists the attributes of a server in the structure format:

```
cat .self:s
```

Primitive values

The following table indicates how primitive values are converted between the API and their string representations in SA CLI methods. Except for Dates, primitive values do not support format specifiers. Dates support ID format specifiers.

Conversion between primitive types and SA CLI Methods

Primitive type	Java equivalent	Output from SA CLI method	Input to SA CLI methods
String	<code>java.lang.String</code>	Character string, presented in the encoding of the current session.	Character string, converted to Unicode from the current session encoding.
Number	<code>byte</code> , <code>short</code> , <code>int</code> , <code>long</code> , <code>float</code> , <code>double</code> ; and their object equivalents	Decimal format, not localized. Scientific notation for very large or small values.	Examples - Decimal: 101, 512.34, -104 Hex: 0x1F32, 0x2e40 Octal: 0543

Conversion between primitive types and SA CLI Methods, continued

Primitive type	Java equivalent	Output from SA CLI method	Input to SA CLI methods
			Scientific: 4.3E4, 6.532e-9, 1.945e+02
Boolean	boolean, Boolean	true or false	The string "true" and all mixed-case variants evaluate to true. All other values evaluate to false.
Binary data	byte[], Byte[]	Binary string. No conversion from session encoding.	Binary string. No conversion to session encoding.
Date	java.util.Calendar	Date value. By default, presented in this format: YYYY/MM/DD HH:MM:SS.mmm The time is presented in UTC. If an ID format specifier is indicated, the value is presented as the number of milliseconds since the epoch, in UTC.	Same as output.

Arrays

The representation of array objects depends on whether they are standalone (an array attribute file or a method return value) or contained in the structure of a complex object.

First, standalone array objects are presented according the underlying type, separated by new-line characters. Within an array element, a new-line character is escaped by `\n` and a back slash by `\\`.

Array values can be output or input using any representation supported by the underlying type. For example, by default, the `getDeviceGroups` method lists the groups as names:

```
All Windows Servers
Servers in Austin
Testing Pool
```

If you indicate the ID format specifier, (`.getDeviceGroups:i`) the method displays the IDs of the groups:

```
com.opsware.device.DeviceGroupRef:15960039
com.opsware.device.DeviceGroupRef:10390039
com.opsware.device.DeviceGroupRef:17380039
```

Second, an array contained in the structure of a complex object is represented as a set of name-value pairs, using the attribute as the name. The attribute appears multiple times, once for each element in the array. The order in which the attributes appear determine the order of the elements in the array. The following example shows a structure that contains two attributes, a string called `subject` and a three-element array of numbers called `ranks`:

```
{ subject="my favorites" ranks=17 ranks=44 ranks=24 }
```

Arrays can also be represented by directories. Within an array directory, each array element has a corresponding file (for primitive types) or subdirectory (for complex types). The name of each entry is the index number of the array element, starting with zero.

For an array that is the attribute of a complex object, you should modify the array by editing its attribute file. This action completely replaces the array with the contents of the edited file.

For an array containing elements that are complex objects, you should modify the array by changing its directory representation. To change an element value, edit the element file. For example, suppose you have an array with five string elements. The `ls` command lists the elements as follows:

```
0 1 2 3 4
```

The following command changes the value of the third element:

```
echo -n "My new value" > 2
```

SA CLI method parameters and return values

This section discusses the details of method context (instance or static), parameter usage, return values, and exit status.

Method context and the self parameter

In the OGFS, a method resides in multiple locations. The location of a method is related to its context, which is either instance or static.

The method with instance context resides in `method` directory of a specific SA object. The method invocation does not require the `self` parameter. The instance of the object affected by the method is implied by the method location. The following example changes the customer of the `d04.example.com` server:

```
cd /opsw/Server/@/d04.example.com/method  
./setCustomer customer:i=9
```

A method with static context resides in a single location under `/opsw/api`. The method invocation requires the `self` parameter to identify the instance affected by the method. In the following static context example, `self:i` specifies the ID of the managed server:

```
cd /opsw/api/com/opsware/server/ServerService/method
./setCustomer self:i=230054 customer:i=9
```

Passing arguments on the command-line

The command-line arguments are specified as name-value pairs, joined by the equal sign (=). The name-value pairs are separated by one or more white space characters, typically spaces. The names on the command-line match the parameter names of the corresponding Java method in the SA API.

For example, in the SA API, the `setCustomField` method has the following definition:

```
public void setCustomField(CustomFieldReference self,
    java.lang.String fieldName, java.lang.String strValue)...
```

The following SA CLI method example assigns a value to a custom field of the server with ID 3670039:

```
cd /opsw/api/com/opsware/server/ServerService/method
./setCustomField self:i=3670039 \
fieldName="Service Agreement" strValue="Gold"
```

Writer's Note: check the above command by running it (TBD)

As described in the previous section, a method with an instance context does not require the `self` parameter. The following `setCustomField` example is equivalent to the preceding example:

```
cd /opsw/.Server.ID/3670039
./setCustomField \
fieldName="Service Agreement" strValue="Gold"
```

You can specify the command-line arguments in any order. The following two SA CLI method invocations are equivalent:

```
./setCustomField fieldName="My Stuff" strValue="abc"
./setCustomField strValue="abc" fieldName="My Stuff"
```

To specify a null value for a parameter, either omit the parameter or insert a white space after the equal sign. In the following examples, the value of `myParam` is null:

```
./someMethod myField="more info" myParam= anotherParam=9834
./someMethod myField="more info"          anotherParam=9834
```

Specifying the type of a parameter

If a method has an abstract type for a parameter, you must specify the concrete type as well as the value. In the following example, the `com.opsware.folder.FolderRef` type is required:

```
cd /opsw/api/com/opsware/folder/FolderService/method
./remove self:i="com.opsware.folder.FolderRef:730555"
```

If you do not specify the concrete type, the following error message is displayed:

Object type *type-name* is abstract. Specify a concrete sub-type.

Complex objects and arrays as parameters

To pass an argument that is a complex object, enclose the object's attributes in curly braces, as shown in ["Structure format specifier syntax " on page 39](#).

The following example creates a public server group named AllMine. The `create` method has a single parameter, `pattern`, which encloses the `parent` and `shortName` attributes in curly braces. In this example, `getPublicRoot` returns 2340555, the ID of the top public group.

```
cd /opsw/api/com/opsware/device/DeviceGroupService/method
./getPublicRoot:i ; echo
./create "pattern={ parent:i=2340555 shortName='AllMine' }"
```

Specify array parameters by repeating the parameter name, once for each array element. For example, the following invocation of the `assign` method specifies the first two elements in the array parameter named `policies`:

```
cd /opsw/api/com/opsware/swmgmt
cd SoftwarePolicyService/method
./attachPolicies self:i=4220039 \
policies:i=4400335 policies:i=4400942
```

Overloaded methods

A Java method name is overloaded if multiple methods in the same class have the same name but different parameter lists. With overloaded SA CLI methods, the argument names on the command-line indicate which method to invoke. The `setCustomField` method, for example, is overloaded to support

the setting of different data types. The following two commands invoke different versions of the method:

```
./setCustomField \  
fieldName="Service Agreement" strValue="Gold"  
./setCustomField \  
fieldName=hmp longValue=2245
```

Return values

If the API method underlying an SA CLI method returns a value, then the SA CLI method outputs the value to `stdout`. As with Unix commands, you can redirect a method's `stdout` to a file or assign it to an environment variable.

To change the representation of the return value, insert a leading period and append a format specifier to the method name. The following example returns server references as IDs, instead of the default names:

```
cd /opsw/api/com/opsware/server/ServerService/method  
./findServerRefs:i
```

If you indicate a format specifier that is incompatible with the method's return type, the file system responds with an error.

Exit status

Like Unix shell commands, SA CLI methods use the exit status ($\$?$) to indicate the result of the call. An exit status of zero indicates success; a non-zero indicates an error. SA CLI methods output error messages to `stderr`.

Exit status codes for SA CLI methods

Exit status	Category	Description
0	Success	The method completed successfully.
1	Command-Line Parse Error	The command-line for the method call is malformed and could not be parsed into a set of options (<code>--option[=value]</code>) and parameter values (<code>param=value</code>).
2	Parameter Parse Error	The parameter values could not be parsed into the object types required by the API.

Exit status codes for SA CLI methods, continued

Exit status	Category	Description
3	API Usage Error	The call failed because of a usage error, such as an invalid parameter value.
4	Access Error	The user does not have permission to perform the operation.
5	Other Error	An error occurred other than those indicated by exit statuses 1- 4.

For example, the following bash script checks the exit status of the `getDeviceGroups` method:

```
#!/bin/bash

cd /opsw/Server/@/toro.snv1.corp.example.com/method
./getDeviceGroups
cmd_exit_status=$?
if [ $cmd_exit_status -eq 0 ]
then
    echo "The command was successful."
else
    echo "The command failed."
    echo "Exit status = " $cmd_exit_status
fi
```

An SA CLI method invokes an underlying API method. If the API method throws an exception, the SA CLI method returns a non-zero exit status. When debugging a method call, you might find it helpful to view information about a thrown exception. The

`/sys/last-exception` file in the OGFS contains the stack trace of an exception thrown by the most recent API call. After this file has been read, the system discards the file contents.

Search filters and SA CLI methods

Many methods in the SA API accept object references as parameters. To retrieve object references based on search criteria, you invoke methods such as `findServerRefs` and `findJobRefs`. For example, you can invoke `findServerRefs` to search for all servers that have `example.com` in the `hostname` attribute.

Search syntax

Methods such as `findServerRefs` have the following syntax:

```
findobjectRefs filter='[object-type:]expression'
```

The `filter` parameter includes an expression, which specifies the search criteria. You enclose an expression in either parentheses or curly brackets. A simple expression has the following syntax:

value-object.attribute operator value

This syntax is simplified. For the full definition, see ["Filter grammar" on page 193](#).

Search examples

Most of the SA object types have associated finder methods. This section shows how to use just a few of them. To see how searches are used with other SA CLI methods, see ["Sample scripts" on page 54](#).

Finding servers

Find servers with host names containing `example.com`:

```
cd /opsw/api/com/opsware/server/ServerService/method
./findServerRefs:i \
filter='device:{ ServerVO.hostname CONTAINS example.com }'
```

Find servers with a use attribute value of either UNKNOWN or PRODUCTION:

```
cd /opsw/api/com/opsware/server/ServerService/method
./findServerRefs:i \
filter='{ ServerVO.use IN "UNKNOWN" "PRODUCTION" }'
```

The following bash script shows how to search for servers, save their IDs in a temporary file, and then specify each ID as the parameter of another method invocation. This script displays the public groups that each Linux server belongs to.

```
#!/bin/bash
```

```
TMPFILE=/tmp/server-list.txt
```

```
rm -f $TMPFILE
```

```
cd /opsw/api/com/opsware/server/ServerService/method
```

```
./findServerRefs:i \
```

```
filter='{ ServerVO.osVersion CONTAINS Linux }' > $TMPFILE
```

```
for ID in `cat "$TMPFILE"`  
do  
    echo Server ID: $ID  
    ./getDeviceGroups self:i=$ID  
    echo  
done
```

Finding jobs

The examples in this section return the IDs of jobs such as server audits or policy remediations.

Find the jobs that have completed successfully:

```
cd /opsw/api/com/opsware/job/JobService/method  
./findJobRefs:i filter='job:{ job_status = "SUCCESS" }'
```

(For a list of allowed values of `job_status`, see “Job Approval Integration” in the SA 10.51 Integration Guide.)

Find the jobs that have completed successfully or with warning:

```
cd /opsw/api/com/opsware/job/JobService/method  
./findJobRefs:i \  
filter='job:{ job_status IN "SUCCESS" "WARNING" }'
```

Find the jobs that have been started today:

```
cd /opsw/api/com/opsware/job/JobService/method  
./findJobRefs:i \  
filter='job:{ JobInfoVO.startDate IS_TODAY "" }'
```

Find all server audit jobs:

```
cd /opsw/api/com/opsware/job/JobService/method  
./findJobRefs \  
filter='job:{ JobInfoVO.description = "Server Audit" }'
```

Find the jobs that have run on the server with the ID 280039:

```
cd /opsw/api/com/opsware/job/JobService/method
```

```
./findJobRefs:i filter='job:{ job_device_id = "280039" }'
```

Find today's jobs that have failed:

```
cd /opsw/api/com/opsware/job/JobService/method
./findJobRefs:i \
filter='job:{ (( JobInfoVO.startDate IS_TODAY "" ) \
& ( job_status = "FAILURE" )) }'
```

Finding other objects

This section has examples that search for software policies and packages.

Find the software policies created by the SA user jdoe:

```
cd /opsw/api/com/opsware/swmgmt/SoftwarePolicyService/method
./findSoftwarePolicyRefs:i \
filter='{ SoftwarePolicyVO.createdBy CONTAINS jdoe }'
```

Find the MSIs with ismtool for the Windows 2003 platforms:

```
cd /opsw/api/com/opsware/pkg/UnitService/method
./findUnitRefs:i \
filter='software_unit:{ ((UnitVO.unitType = "MSI") \
& ( UnitVO.name contains "ismtool" ) \
& ( software_platform_name = "Windows 2003" )) }'
```

Find the Solaris patches named 117170-01:

```
cd /opsw/api/com/opsware/pkg/solaris/SolPatchService/method
./findSolPatchRefs:i filter='{name = 117170-01}'
```

Find the folder with the name that includes the string Test and with a parent folder named My Stuff.

```
cd /opsw/api/com/opsware/folder/FolderService/method
./findFolders:s \
filter='{ ( FolderVO.name CONTAINS "Test" ) \
& ( folder_parent_name = "My Stuff" ) }'
```

Searchable attributes and valid operators

Not every attribute of a value object can be specified in a search filter. For example, you can search on `ServerVO.use` but not on `ServerVO.OsFlavor`.

To find out which attributes are searchable for a given object type, invoke the `getSearchableAttributes` method. The following example lists the attributes of `ServerVO` that can be specified in a search expression:

```
cd /opsw/api/com/opsware/search/SearchService/method
./getSearchableAttributes searchableType=device
```

The `searchableType` parameter indicates the object type. To determine the allowed values for `searchableType`, enter the following commands:

```
cd /opsw/api/com/opsware/search/SearchService/method
./getSearchableTypes
```

To find out which operators are valid for an attribute, invoke the `getSearchableAttributeOperators` method. The following example lists valid operators (such as `CONTAINS` and `IN`) for the attribute `ServerVO.hostname`:

```
cd /opsw/api/com/opsware/search/SearchService/method
./getSearchableAttributeOperators searchableType=device \
searchableAttribute=ServerVO.hostname
```

Sample scripts

This section has code listings for simple bash scripts that invoke a variety of SA CLI methods. These scripts demonstrate how to pass method parameters on the command-line, including complex objects and the `self` parameter. If you decide to copy and paste these example scripts, you will need to change some of the hard-coded object names, such as the `d04.example.com` server. For tutorial instructions on creating and running scripts within the OGFS, see ["SA CLI method tutorial" on page 31](#).

The script ["remediate_policy.sh" on page 59](#) creates a software policy, adds a package to the policy, and in the last line, installs the package on a managed server by invoking the `startFullRemediateNow` method.

create_custom_field.sh

This script creates a custom field (virtual column), named `TestFieldA` attaches the field to all servers, and then sets the value of the field on a single server. Until it is attached, the custom field does not appear in the SA Client. You can create custom fields for servers, device groups, or software policies. To create a custom field, your SA user must belong to a user group with the Manage Virtual Columns permission.

Unlike a custom attribute, a custom field applies to all instances of a type. For an example that creates a custom attribute in the OGFS, see [Managing custom attributes in the Server Automation Using Guide](#) on the HPE SSO portal.

The `create_custom_field.sh` script has the following code:

```
#!/bin/bash
# create_custom_field.sh

cd /opsw/api/com/opsware/custattr/VirtualColumnService/method

# Create a virtual column.
# Remember the name because you cannot search for the
# displayName.
./create vo='{ name=TestFieldA type=SHORT_STRING \
displayName="Test Field A" }'

column_id='./findVirtualColumn:i name=TestFieldA'

echo --- column_id = $column_id

cd /opsw/api/com/opsware/server/ServerService/method

# Attach the column to all servers.
# All servers will have this custom field.
./attachVirtualColumn virtualColumn:i=$column_id
```

```
# Get the ID of the server named d04.example.com
devices_id='./findServerRefs:i \
filter=\
'device:{ ServerVO.hostname CONTAINS "d04.example.com" }'

echo --- devices_id = $devices_id

# Set the value of the custom field (virtual column) for
# a specific server.
./setCustomField self:i=$devices_id fieldName=TestFieldA \
strValue="This is something."
```

create_device_group.sh

This script creates a static device group and adds a server to the group. Next, the script creates a dynamic group, sets a rule on the group, and refreshes the membership of the group. The last statement of the script lists the devices that belong to the dynamic group.

Here is the script's code:

```
#!/bin/bash
# create_device_group.sh

cd /opsw/api/com/opsware/device/DeviceGroupService/method

# Get the ID of the public root group (top of hierarchy).
public_root='./getPublicRoot:i'

# Create a public static group.
./create "vo={ parent:i=$public_root shortName='Test Group A' }"

# Get the ID of the group just created.
group_id='./findDeviceGroupRefs:i \
```



```
filter='{ DeviceGroupVO.shortName = "Test Group A" }' '

echo --- group_id = $group_id

cd /opsw/api/com/opsware/server/ServerService/method

# Get the ID of the server named d04.example.com
devices_id='./findServerRefs:i \
filter=\
'device:{ ServerVO.hostname CONTAINS "d04.example.com" }' '

echo --- devices_id = $devices_id

cd /opsw/api/com/opsware/device/DeviceGroupService/method

# Add a server to the device group.
./addDevices \
self:i=$group_id devices:i=$devices_id

# Create a dynamic device group.
./create \
"vo={ parent:i=$public_root \
shortName='Test Dyn B' dynamic=true }"

# Get the ID of the device group.
dynamic_group_id='./findDeviceGroupRefs:i \
filter='{ DeviceGroupVO.shortName = "Test Dyn B" }' '

echo --- dynamic_group_id = $dynamic_group_id

# Set the rule so that this group contains servers with
```

```
# hostnames containing the string example.com.
# The rule parameter has the same syntax as the filter
# parameter of the find methods.
./setDynamicRule self:i=$dynamic_group_id \
rule='device:{ ServerVO.hostname CONTAINS example.com }'

# By default, membership in dynamic device groups is refreshed
# once
# an hour, so force the refresh now.
./refreshMembership selves:i=$dynamic_group_id now=true

# Display the names of the devices that belong to the group.
echo --- Devices in group:
./getDevices selves:i=$dynamic_group_id
```

create_folder.sh

This script creates a folder named /Test 1, lists the folders under the root (/) folder, and then creates the subfolder /Test 1/Test 2. After creating these folders, you can view them under the Library in the navigation pane of the SA Client.

Here is the code for this script:

```
#!/bin/bash
# create_folder.sh

cd /opsw/api/com/opsware/folder/FolderService/method

# Get the ID of the root (top) folder.
root_id=`./getRoot:i`

# Create a new folder under the root folder.
./create vo="{ name='Test 1' folder:i=$root_id }"
```

```
# Display the names of the folders under the root folder.
./getChildren self:i=$root_id

# Get the ID of the folder "/Test 1"
folder_id=`./getFolderRef:i path="Test 1"`

# Create a subfolder.
./create vo="{ name='Test 2' folder:i=$folder_id }"

# Get the ID of the folder "/Test 1/Test 2"
folder_id=`./getFolderRef:i path="Test 1" path="Test 2"`
echo folder_id = $folder_id
```

remediate_policy.sh

This script creates a software policy named `TestPolicyA` in an existing folder named `Test 2`, adds a package containing `ismtool` to the policy, attaches the policy to a single server (not a group), and then remediates the server. The remediation action launches a job that installs the package onto the server. You can check the progress and results of the job in the SA Client. For examples that search for jobs with SA CLI methods, see ["Finding jobs" on page 52](#).

In this script, in the `create` method of the `SoftwarePolicyService`, the value of the `platforms` parameter is hard-coded. In most of these example scripts, hard-coding is avoided by searching for an object by name. In the case of `platforms`, searching by the `name` attribute is difficult because it differs from the `displayName` attribute, which is exposed in the SA Client but is not searchable. The easiest way to find a platform ID is by going to the twister and running the `PlatformService.findPlatformRefs` method with no parameters.

The `update` method in this script hard-codes the ID of `softwarePolicyItems`, an object that can be difficult to search for by name if the Software Repository contains many packages with similar names. One way to get the ID is to run the SA Client, search for Software by fields such as File Name and Operating System, open the package located by the search, and note the SA ID in the properties view of the package.

In the following listing, the `update` method has a bad line break. If you copy this code, edit the script so that the `vo` parameter is on a single line.

Here is the source code for the `remediate_policy.sh` script:

```
#!/bin/bash
# remediate_policy.sh

# Get the ID of the folder where the policy will reside.
cd /opsw/api/com/opsware/folder/FolderService/method
folder_id=`./findFolders:i filter='{ FolderVO.name = "Test 2" }'`

cd /opsw/api/com/opsware/swmgmt/SoftwarePolicyService/method

# Create a software policy named TestPolicyA.
# This policy resides in the folder located in the preceding findFolders
# call.
# The platform for this policy is Windows 2008 (ID 160076)
./create vo="{ platforms:i=160076 name="TestPolicyA" \
folder:i=$folder_id lifecycle=AVAILABLE }"

policy_id=`./findSoftwarePolicyRefs:i \
filter='{ SoftwarePolicyVO.name = "TestPolicyA" }'`

echo --- policy_id = $policy_id

# Call the update method to add a package to the software policy.
# The package ID for the "ismtool" msi installer is 4010001.

# Note that "force = true" is required.

./update self:i=$policy_id force=true \
vo='{ softwarePolicyItems:i=com.opsware.pkg.windows.MSIRef:4010001 }'

cd /opsw/api/com/opsware/server/ServerService/method
```

```
# Get the ID of the server named d04.opsware.com
devices_id=`./findServerRefs:i \
filter='device:{ ServerV0.hostname CONTAINS "d04.opsware.com" }'`

echo --- devices_id = $devices_id

# Attach the policy to a single server (not a group).
./attachPolicies self:i=$devices_id \
policies:i=$policy_id

# Remediate the server to install the package in the policy.
job_id=`./startFullRemediateNow:i self:i=$devices_id`

echo --- job_id = $job_id
```

remove_custom_field.sh

Although not common in an operational environment, removing custom fields is sometimes necessary in a testing environment. Note that a custom field must be unattached before it can be removed.

Here is the code for `remove_custom_field.sh`:

```
#!/bin/bash
# remove_custom_field.sh

if [ ! -n "$1" ]
then
    echo "Usage: 'basename $0' <name>"
    echo "Example: 'basename $0' hmp"
    exit
fi

cd /opsw/api/com/opsware/custattr/VirtualColumnService/method
```

```
column_id='./findVirtualColumn:i name=$1'  
  
echo --- column_id = $column_id  
  
cd /opsw/api/com/opsware/server/ServerService/method  
  
# Column must be detached before it can be removed.  
./detachVirtualColumn virtualColumn:i=$column_id  
  
cd /opsw/api/com/opsware/custattr/VirtualColumnService/method  
  
# Remove the virtual column.  
./remove self:i=$column_id
```

schedule_audit_task.sh

This script starts an audit task, scheduling it for a future date. With SA CLI methods, date parameters are specified with the following syntax:

```
YYYY/MM/DD HH:MM:SS.sss
```

The method that launches the task, `startAudit`, returns the ID of the job that performs the audit. For examples that search for jobs with SA CLI methods, see ["Finding jobs" on page 52](#).

Here is the code for `schedule_audit_task.sh`:

```
#!/bin/bash  
# schedule_audit_task.sh  
  
cd /opsw/api/com/opsware/compliance/sco/AuditTaskService/method  
  
# Get the ID of the audit task to schedule.  
  
audit_task_id=`./findAuditTask:i \  
`
```

```
filter='audit_task:{ (( AuditTaskVO.name BEGINS_WITH "HW check" ) \  
& ( AuditTaskVO.createdBy = "gsmith" )) }`'  
  
echo --- audit_task_id = $audit_task_id  
  
# Schedule the audit task for Oct. 16, 2013.  
# In the startDate parameter, note that the last delimiter for the time  
# is a period, not a colon.  
  
job_id=`./startAudit:i self:i=$audit_task_id  
schedule:s='{ startDate="2013/10/16 00:00:00.000" }' \  
notification:s='{ onFailureOwner="sjones@opsware.com" \  
onFailureRecipients="jdoe@opsware.com" \  
onSuccessOwner="sjones@opsware.com" \  
onSuccessRecipients="jdoe@opsware.com" }`'  
  
echo --- job_id = $job_id
```

Getting usage information on SA CLI methods

In a future release, the SA CLI methods will display usage information. Until then, you can get the necessary information from the API documentation or the OGFS with the techniques described in the following sections.

Listing services

The SA API methods are organized into services. To find out what services are available for SA CLI methods, enter the following commands in an OGS session:

```
cd /opsw/api/com/opsware  
find . -name "*Service"
```

To list the services in the API documentation, specify the following URL in your browser:

```
https://occ_host:1032
```

The `occ_host` is the IP address or host name of the core server running the Command Center component.

Finding a service in the API documentation

The path of the service in the OGFS maps to the Java package name in the API documentation. For example, in the OGFS, the `ServerService` methods appear in the following directory:

```
/opsw/api/com/opsware/server
```

In the API documentation, the following interface defines these methods:

```
com.opsware.server.ServerService
```

Listing the methods of a service

In the OGFS, you can list the contents of the method directory of a service. For example, to display the method names of the `ServerService`, enter the following command:

```
ls /opsw/api/com/opsware/server/ServerService/method
```

In the API documentation, perform the following steps to view the methods of `ServerService`:

In the upper left pane, select `com.opsware.server`.

In the lower left pane, select `ServerService`.

In the main pane, scroll down to view the methods.

Listing the parameters of a method

In the API documentation, perform the steps described in the preceding section. In the Method Detail section of the service interface page, view the parameters and return types. For more information about method parameters, see ["Passing arguments on the command-line" on page 47](#).

Getting information about a value object

The API documentation shows that some service methods pass or return value objects (VOs), which contain data members (attributes). For example, the `ServerService.getServerVO` method returns a `ServerVO` object. To find out what attributes `ServerVO` contains, perform the following steps:

In the API documentation, select the `ServerVO` link. You can find this link in several places:

- The method signature for `getServerVO`
- The list of classes (lower left pane) for `com.opsware.server`
- On the Index page. A link to the Index page is at the top of the main pane of the API documentation.
- On the `ServerVO` page, note the getter and setter methods. Each getter-setter pair corresponds to an attribute contained in the value object. For example, `getCustomer` and `setCustomer` indicate that `ServerVO` contains an attribute named `customer`.

Determining if an attribute can be modified

Only a few object attributes can be modified by client applications. To find out if an attribute can be modified, perform the following steps:

1. In the API documentation, go to the value object page, as described in the preceding section.
2. In the Method Detail section of the setter method, look for “Field can be set by clients.”

For SA objects represented in the OGFS, such as servers and customers, you can determine which attributes are modifiable by checking the access types of the files in the `attr` directory. The files that have read-write (`rw`) access types correspond to modifiable attributes. For example, to list the modifiable attributes of a server, enter the following commands:

```
cd /opsw/Server/@/server-name/attr
```

```
ls -l | grep rw
```

Determining if an attribute can be used in a filter query

To find out if an attribute of a value object can be used in a filter query (a search), perform the following steps:

1. In the API documentation, go to the value object page.
2. In the Method Detail section of the getter method that corresponds to the attribute, look for the string, "Field can be used in a filter query."

From within an OGSH session, to find out if an attribute can be searched on, follow the techniques described in "[Searchable attributes and valid operators](#)" on page 54.

Python API access with Pytwist

Pytwist is a set of Python libraries that provide access to the SA API from managed servers and custom extensions. (The twist is the internal name for the Web Services Data Access Engine.) For managed servers, you can set up Python scripts that call SA APIs through Pytwist so that end users can invoke the scripts as DSEs or ISM controls. Created by HPE SA Professional Services, custom extensions are Python scripts that run in the Command Engine (way). Pytwist enables custom extensions to access recent additions to the SA data model, such as folders and software policies, which are not accessible from Command Engine scripts.

This topic is intended for developers and consultants who are already familiar with the SA data model, custom extensions, Agents, and the Python programming language.

Following topics are discussed in this section:

- "[Setup for Pytwist](#)" on the next page
- "[Pytwist examples](#)" on the next page
- "[Virtualization Pytwist examples](#)" on page 72
- "[Pytwist details](#)" on page 80

Setup for Pytwist

Before trying out the examples in this section, make sure that your environment meets the following setup requirements, as detailed in the following sections.

- ["Supported platforms for Pytwist" below](#)
- ["Access requirements for Pytwist" below](#)
- ["Installing Pytwist libraries" below](#)

Supported platforms for Pytwist

Pytwist is supported on managed servers and core servers. For a list of operating systems supported for these servers.

Pytwist relies on Python version 2.7.10, the version used by SA Agents and custom extensions.

Unlike Web Services and Java RMI clients, a Pytwist client relies on internal SA libraries. If your client program needs to access the SA API from a server that is not a managed or core server, then use a Web Services or Java RMI client, not Pytwist.

Access requirements for Pytwist

Pytwist needs to access port 1032 of the core server running the Web Services Data Access Engine. By default, the engine listens on port 1032.

Installing Pytwist libraries

The pytwist libraries need not be installed as they are part of the agent libraries.

Pytwist examples

The Python code examples in this section show how to get information from managed servers, create folders, and remediate software policies. Each Pytwist example performs the following operations:

1. Import the packages.

When importing objects of the SA API name space, such as `Filter`, the path includes the Java package name, preceded by `pytwist`. Here are the `import` statements for the `get_server_info.py` example:

```
import sys
from pytwist import *
from pytwist.com.opsware.search import Filter
```

2. Create the TwistServer object:

```
ts = twistserver.TwistServer()
```

See ["TwistServer method syntax" on page 80](#) for information about the method's arguments.

3. Get a reference to the service.

The Python package name of the service is the same as the Java package name, but without the leading `opsware.com`. For example, the Java `com.opsware.server.ServerService` package maps to the Pytwist `server.ServerService`:

```
serverservice = ts.server.ServerService
```

4. Invoke the SA API methods of the service:

```
filter = Filter()
. . .
servers = serverservice.findServerRefs(filter)
. . .
for server in servers: vo = serverservice.getServerVO(server)
. . .
```

get_server_info.py

This script searches for all managed servers with host names containing the command-line argument. The search method, `findServerRefs`, returns an array of references to server persistent objects. For each reference, the `getServerVO` method returns the value object (VO), which is the data representation that holds the server's attributes. Here is the code for the `get_server_info.py` script:

```
#!/opt/opsware/agent/bin/python
# get_server_info.py

# Search for servers by partial hostname.

import sys
from pytwist import *
from pytwist.com.opsware.search import Filter

# Check for the command-line argument.
if len(sys.argv) < 2:
    print "You must specify part of the hostname as the search target."
```

```
        print "Example: " + sys.argv[0] + " " + "opsware.com"
        sys.exit(2)

# Construct a search filter.
filter = Filter()
filter.expression = 'device_hostname *=* "%s" ' % (sys.argv[1])

# Create a TwistServer object.
ts = twistserver.TwistServer()

# Get a reference to ServerService.
serverservice = ts.server.ServerService

# Perform the search, returning a tuple of references.
servers = serverservice.findServerRefs(filter)

if len(servers) < 1:
    print "No matching servers found"
    sys.exit(3)

# For each server found, get the server's value object (VO)
# and print some of the VO's attributes.
for server in servers:
    vo = serverservice.getServerVO(server)
    print "Name: " + vo.name
    print "Management IP: " + vo.managementIP
    print "OS Version: " + vo.osVersion
```

create_folder.py

This script creates a folder named /TestA/TestB by invoking the createPath method. Note that the path parameter of createPath does not contain slashes. Each string element in path indicates a level in the folder. Next, the script retrieves and prints the names of all folders directly below the root folder. The listing for the create_folder.py script follows:

```
#!/opt/opsware/agent/bin/python
# create_folder.py

# Create a folder in SA.

import sys
from pytwist import *

# Create a TwistServer object.
ts = twistserver.TwistServer()

# Get a reference to FolderService.
folderservice = ts.folder.FolderService
```

```
# Get a reference to the root folder.
rootfolder = folderservice.getRoot()
# Construct the path of the new folder.
path = 'TestA', 'TestB'

# Create the folder /TestA/TestB relative to the root.
folderservice.createPath(rootfolder, path)

# Get the child folders of the root folder.
rootchildren = folderservice.getChildren(rootfolder,
'com.opsware.folder.FolderRef')

# Print the names of the child folders.
for child in rootchildren:
    vo = folderservice.getFolderVO(child)
    print vo.name
```

remediate_policy.py

This script creates a software policy, attaches it to a server, and then remediates the policy. Several names are hard-coded in the script: the platform, server, and parent folder. Optionally, you can specify the policy name on the command-line, which is convenient if you run the script multiple times. The platform of the software policy must match the OS of the packages contained in the policy. Therefore, if you change the hard-coded platform name, then you also change the name in `unitfilter.expression`.

The following listing has several bad line breaks. If you copy this code, be sure to fix the bad line breaks before running it. The comment lines beginning with "NOTE" point out the bad line breaks.

```
#!/opt/opsware/agent/bin/python
# remediate_policy.py

# Create, attach, and remediate a software policy.

import sys
from pytwist import *
from pytwist.com.opsware.search import Filter
from pytwist.com.opsware.swmgmt import SoftwarePolicyVO

# Initialize the names used by this script.
foldername = 'TestB'
platformname = 'Windows 2003'
servername = 'd04.example.com'
# If a command-line argument is specified,
# use it as the policy name
if len(sys.argv) == 2:
    policyname = sys.argv[1]
else:
```

```
    policyname = 'TestPolicyA'

# Create a TwistServer object.
ts = twistserver.TwistServer()
ts.authenticate("SAUser", "SAPassword")

# Get the references to the services used by this script.
folderservice = ts.folder.FolderService
swpolicyservice = ts.swmgmt.SoftwarePolicyService
serverservice = ts.server.ServerService
unitservice = ts.pkg.UnitService
platformservice = ts.device.PlatformService

# Search for the folder that will contain the policy.
folderfilter = Filter()
folderfilter.expression = 'FolderVO.name = %s' % foldername
folderrefs = folderservice.findFolderRefs(folderfilter)
if len(folderrefs) == 1:
    parent = folderrefs[0]
elif len(folderrefs) < 1:
    print "No matching folders found."
    sys.exit(2)
else:
    print "Non-unique folder name: " + foldername
    sys.exit(3)

# Search for the reference to the platform "Windows Server 2003."
platformfilter = Filter()
platformfilter.objectType = 'platform'
# Because the platform name contains spaces,
# it's enclosed in double quotes
# NOTE: The following code line has a bad line break.
# The assignment statement should be on a single line.
platformfilter.expression = 'platform_name = "%s"' % platformname
platformrefs = platformservice.findPlatformRefs(platformfilter)

if len(platformrefs) == 0:
    print "No matching platforms found."
    sys.exit(4)

# Search for the references to some software packages.
unitfilter = Filter()
unitfilter.objectType = 'software_unit'
# NOTE: The following code line has a bad line break.
# The assignment statement should be on a single line.
unitfilter.expression = '((UnitVO.unitType = "MSI") & ( UnitVO.name contains
"ismtool" ) & ( software_platform_name = "Windows 2003" ))'
unitrefs = unitservice.findUnitRefs(unitfilter)

# Create a value object for the new software policy.
```

```
vo = SoftwarePolicyVO()
vo.name = policyname
vo.folder = parent
vo.platforms = platformrefs
vo.softwarePolicyItems = unitrefs

# Create the software policy.
swpolicyvo = swpolicyservice.create(vo)

# Search by hostname for the reference to a managed server.
serverfilter = Filter()
serverfilter.objectType = 'server'
# NOTE: The following code line has a bad line break.
# The assignment statement should be on a single line.
serverfilter.expression = 'ServerVO.hostname = %s' % servername
serverrefs = serverservice.findServerRefs(serverfilter)

if len(serverrefs) == 0:
    print "No matching servers found."
    sys.exit(5)

# Create an array that has a reference to the
# newly created policy.
swpolicyrefs = [1]
swpolicyrefs[0] = swpolicyvo.ref

# Attach the software policy to the server.
swpolicyservice.attachToPolicies(swpolicyrefs, serverrefs)

# Remediate the policy and the server.
# NOTE: The following code line has a bad line break.
# The assignment statement should be on a single line.
jobref = swpolicyservice.startRemediateNow(swpolicyrefs, serverrefs)
print "The remediation job ID is %d" % jobref.id
```

Virtualization Pytwist examples

This topic provides examples (["createVM_WithOSBP.py" below](#) and ["deployVM.py " on page 77](#)) of creating and deploying virtual machines (VMs) using SA API. For more examples about Virtualization, see the Server Automation Using Guide on the HPE SSO portal.

createVM_WithOSBP.py

This basic example creates a VM on a VMware vCenter using CD boot with static IP configuration.

All properties have not been set in these examples. Please refer to API documentation (javadocs) to understand and set the properties for your use case.

```
#!/opt/opsware/agent/bin/python
from pytwist import twistserver
from pytwist.com.opsware.locality import CustomerRef, RealmRef
from pytwist.com.opsware.osprov import OSBuildPlanRef
from pytwist.com.opsware.pkg import UnknownPkgRef
from pytwist.com.opsware.v12n import AdapterIPSettings, V12nHypervisorRef, \
    V12nHypervisorService, V12nInventoryFolderRef, V12nResourcePoolRef, \
    V12nResourcePoolRef, V12nVIManagerService, VirtualCpuConfig, \
VirtualDevice, \
    VirtualDeviceChangeConfig, VirtualDeviceTypeConstant, \
VirtualHardwareConfigSpec, \
    VirtualMemoryConfig, VirtualServerCDProvisioningSpec, \
VirtualServerComputeSpec, \
    VirtualServerConfigSpec, VirtualServerCreateSpec, \
VirtualStorageDeviceConstant, \
    VirtualStorageDeviceHWConfig
from pytwist.com.opsware.v12n.vmware import V12nDatastoreRef, \
    VmwareVirtualInterfaceBacking, VmwareVirtualNicHWConfig, \
    VmwareVirtualServerDetails, VmwareVirtualServerStorageSpec, \
    VmwareVirtualStorageFileBacking
import time
```

```
# This is a bare bones example of creating a Virtual Machine on a VMware
# vCenter while booting from CD with Static IP configuration. It also
# provisions the Virtual Machine with the give OS Build Plan. For more
# detailed information please refer to the java doc. All the properties have
# not been set in the example below, please review the java doc to understand
# and set the properties for your use case.
```

```
# This method constructs the create specification to create the Virtual
# Machine and provision it.
```

```
def constructCreateSpec():

    # Construct VmwareVirtualServerDetails
    detail = VmwareVirtualServerDetails()
    # Virtual Machine Name
    detail.name = "Test VM"
    # Description for the Virtual Machine
    detail.description = "Sample test create VM"
    # This is the key for the guest operating system that will installed on
    # the Virtual Machine.
    # V12nVIManagerService.getGuestOSList() provides the supported list for
    # the given V12n Manager and hypervisor.
    detail.guestId = "rhel6Guest"
```

```
# This is folder where the VM will reside in you can see the list of
# folders at V12nInventoryFolderService.findV12nInventoryFolderRefs() it
# is the inventory location of the Virtual Machine
folder = V12nInventoryFolderRef(2020001)
detail.inventoryFolderRef = folder

# Configure the number of Virtual processors on the Virtual Machine
cpuConfig = VirtualCpuConfig()
cpuConfig.virtualCpuCount = 1

# Configure the Memory for the Virtual Machine
memoryConfig = VirtualMemoryConfig()
memoryConfig.size = 1024*1024*1024

# Configure NICs
# Construct the virtual device of type network i.e a NIC
virtualNetworkDevice = VirtualDevice()
virtualNetworkDevice.type = VirtualDeviceTypeConstant.NETWORK
# A unique identifier for the virtual device
virtualNetworkDevice.key = "4001"
backingNetwork = VmwareVirtualInterfaceBacking()
# This is the port group that the nic will be assigned to
backingNetwork.portGroup = "VLAN 625"

hwConfigNetwork = VmwareVirtualNicHWConfig()
# The kind of network adapter to use, other options are listed in
# VmwareVirtualNicHWConfig
hwConfigNetwork.adapterType = VmwareVirtualNicHWConfig.E1000
hwConfigNetwork.macAddressIsDynamic = True

virtualNetworkDevice.hwConfig = hwConfigNetwork
virtualNetworkDevice.backingInfo = backingNetwork
virtualNetworkDevice.connected = True
virtualNetworkDevice.startConnected = True

# Configure Hard Disk
virtualDiskDevice = VirtualDevice()
virtualDiskDevice.type = VirtualDeviceTypeConstant.STORAGE

backingStorage = VmwareVirtualStorageFileBacking()

# This is Ref for the data store on the hypervisor where the VM will be
# hosted. The list of datastores associated with the Hypervisors are
# listed at V12nHypervisorService.getV12nHypervisorVO() under storage
# config
dataStoreRef = V12nDatastoreRef(90001)
backingStorage.datastore = dataStoreRef
backingStorage.lazyAllocation = True
```

```
hwConfigStorage = VirtualStorageDeviceHWConfig()
hwConfigStorage.capacity = 10*1024*1024*1024
hwConfigStorage.usageType =
VirtualStorageDeviceConstant.USAGE_TYPE_DISK_DRIVE

virtualDiskDevice.hwConfig = hwConfigStorage
virtualDiskDevice.backingInfo = backingStorage

# Add both the virtual devices to be created, i.e. the hard disk and the
# nic
virtualDvcs_toAdd = []
virtualDvcs_toAdd.append(virtualNetworkDevice)
virtualDvcs_toAdd.append(virtualDiskDevice)
deviceChange = VirtualDeviceChangeConfig()
deviceChange.addList = virtualDvcs_toAdd

# Finalize the Config Spec
configSpec = VirtualServerConfigSpec()
configSpec.detail = detail
configSpec.virtualHardware = VirtualHardwareConfigSpec()
configSpec.virtualHardware.cpuConfig = cpuConfig
configSpec.virtualHardware.memoryConfig = memoryConfig
configSpec.virtualHardware.deviceChange = deviceChange

# Constructing the Compute Spec
computeSpec = VirtualServerComputeSpec()
# This is the hypervisor hosting the VM
hypervisorRef = V12nHypervisorRef(2030001)
computeSpec.computeProviderRef = hypervisorRef
# This is resource pool on the hypervisor/cluster that the VM belongs to
# It can be retrieved by using hypervisorVO.children or the Cluster
# children
resourcePool = V12nResourcePoolRef(2040001)
computeSpec.resourcePoolRef = resourcePool

storageSpec = VmwareVirtualServerStorageSpec()
storageSpec.datastore = datastoreRef

# This example deals with provisioning a VM through CD boot and with
# static IP configuration. The example deals setting the boot ISO and
# network information to be used.
# All the information for this is contained in the
# VirtualServerCDProvisioningSpec

# Set all the network information
gateways = []
gw = "192.168.135.33"
gateways.append(gw)
```

```
dnsServers = []
dnsServer = "192.168.2.13"
dnsServers.append(dnsServer)

interfaces = []
interface = AdapterIPSettings()

# Construct the network interface
interface.useDHCP=False
# Note this is the virtual device we have created above, we use the same
# device key to indicate to provisioning which virtual device is to be
# used for provisioning
interface.virtualDeviceKey="4001"
interface.gateways=gateways
interface.ipAddress="192.168.135.45"
interface.netmask="255.255.255.224"
interface.dnsServerList=dnsServers

interfaces.append(interface)

# This is the boot ISO Ref that will be used to get the server into
# maintenance mode
# The name and the id need to match the packages on the core.
# Use the UnitService.findUnitRefs() to find the boot ISO's
bootISORef = UnknownPkgRef(5340001)
bootISORef.name="HPSA_linux_boot_cd.iso"
# The realm assigned to the Virtual Machine will be the realm of the
# Virtualization Service
realmRef = RealmRef(30001)
# The OS Build Plan that needs be run on the Virtual Machine after the VM
# has been created.
osbpRef = OSBuildPlanRef(580001)

provisioningSpec = VirtualServerCDProvisioningSpec()

provisioningSpec.bootISORef = bootISORef
provisioningSpec.interfaces = interfaces
provisioningSpec.realmRef = realmRef
provisioningSpec.osBuildPlanRef = osbpRef

# Finally put together all the information to be set on the Create
# Specification
createSpec = VirtualServerCreateSpec()
createSpec.configSpec = configSpec
createSpec.computeSpec = computeSpec
createSpec.storageSpec = storageSpec

createSpec.provisioningSpec = provisioningSpec
```

```
#Set the customer to be associated with the Virtual Machine
customer = CustomerRef(9)
createSpec.setCustomerRef(customer)
return createSpec

def createVirtualMachine():
    twist = twistserver.TwistServer()
    twist.authenticate("hp", "opsware")
    vmService = twist.v12n.V12nVirtualServerService
    createSpec = constructCreateSpec()
    jobRef = vmService.startCreate(createSpec,4*60*60,"Sample create
VM",None, None)

createVirtualMachine()
```

deployVM.py

This basic example shows how to deploy a VM from a VM template on VMware vCenter and customize the guest OS of the deployed VM.

All properties have not been set in these examples. Please refer to API documentation (javadocs) to understand and set the properties for your use case.

```
#!/opt/opsware/agent/bin/python
from pytwist import twistserver
from pytwist.com.opsware.locality import CustomerRef, RealmRef
from pytwist.com.opsware.osprov import OSBuildPlanRef
from pytwist.com.opsware.pkg import UnknownPkgRef
from pytwist.com.opsware.v12n import AdapterIPSettings, V12nHypervisorRef, \
    V12nHypervisorService, V12nInventoryFolderRef, V12nResourcePoolRef, \
    V12nResourcePoolRef, V12nVIManagerService, VirtualCpuConfig,
VirtualDevice, \
    VirtualDeviceChangeConfig, VirtualDeviceTypeConstant,
VirtualHardwareConfigSpec, \
    VirtualMemoryConfig, VirtualServerCDProvisioningSpec,
VirtualServerComputeSpec, \
    VirtualServerConfigSpec, VirtualServerCreateSpec,
VirtualStorageDeviceConstant, \
    VirtualStorageDeviceHWConfig, V12nVirtualServerTemplateRef, \
    VirtualServerCloneSpec, VirtualServerGuestCustomizationSpec
from pytwist.com.opsware.v12n.vmware import V12nDatastoreRef, \
    VmwareVirtualInterfaceBacking, VmwareVirtualNicHWConfig, \
    VmwareVirtualServerDetails, VmwareVirtualServerStorageSpec, \
    VmwareVirtualStorageFileBacking
import time

# This is a bare bones example of deploying a Template VMware vCenter. It
# deploys the template and then guest customizes the deployed Virtual Machine.
```

```
# For more detailed information please refer to the java doc. All the
# properties have not been set in the example below, please review the java
# doc to understand and set the properties for your use case.

# This method constructs the deploy specification to deploy the Template and
# customizes it.
def constructDeploySpec(sourceTemplateVO):

    # Construct the Deploy Spec
    clonespec = VirtualServerCloneSpec()

    clonespec.computeSpec = VirtualServerComputeSpec()
    # This is the hypervisor hosting the VM
    targetHypervisorRef = V12nHypervisorRef(2030001)
    clonespec.computeSpec.computeProviderRef = targetHypervisorRef

    computeSpec = VirtualServerComputeSpec()
    # This is the resource pool on the hypervisor/cluster that the VM belongs to
    # It can be retrieved by using hypervisorVO.children or the Cluster
    # children
    targetResourcePoolRef = V12nResourcePoolRef(2040001)
    computeSpec.resourcePoolRef = targetResourcePoolRef
    clonespec.computeSpec.resourcePoolRef = targetResourcePoolRef

    storageSpec = VmwareVirtualServerStorageSpec()
    dataStoreRef = V12nDatastoreRef(90001)
    storageSpec.datastore = dataStoreRef
    clonespec.storageSpec = storageSpec
    # Construct VmwareVirtualServerDetails
    detail = VmwareVirtualServerDetails()
    # Virtual Machine Name
    detail.name = "Test Deploy VM"
    # Description for the Virtual Machine
    detail.description = "Sample Deploy create VM"

    # This is the folder where the VM will reside in. You can see the list of
    # folders at V12nInventoryFolderService.findV12nInventoryFolderRefs(). It
    # is the inventory location of the Virtual Machine
    targetFolderRef = V12nInventoryFolderRef(2020001)
    detail.inventoryFolderRef = targetFolderRef
    configSpec = VirtualServerConfigSpec()
    configSpec.detail = detail
    clonespec.configSpec=configSpec

    # Create the Guest Customization Spec, this is needed to customized the
    # deployed VM so that it does not use the network settings and host name
    # of the source template
    # In this example all the interfaces are set to DHCP but you can
    # customize each of the interfaces by either providing static or DHCP
```

```
# configuration details
interfaces = createInterfaces(sourceTemplateVO)
# The realm assigned to the Virtual Machine will be the realm of the
# Virtualization Service
realmRef = RealmRef(30001)
clonespec.guestCustomizationSpec =
createGuestCustomizationSpec("testDeployVM",realmRef,interfaces)
clonespec.setPowerOn(True)
# Set the customer to be associated with the Virtual Machine
customerRef = CustomerRef(9)
clonespec.customerRef = customerRef
return clonespec

def createGuestCustomizationSpec(newVmNameVal,realmRef,interfaces):
    gcSpec = VirtualServerGuestCustomizationSpec()
    gcSpec.computerName = newVmNameVal
    gcSpec.interfaces = interfaces
    gcSpec.realmRef = realmRef
    return gcSpec

def createInterfaces(virtualServerVO):
    interfaces = []
    virtualDevices = virtualServerVO.virtualHardware.deviceList
    vNICs = [vd for vd in virtualDevices if vd.type ==
VirtualDeviceTypeConstant.NETWORK]
    for vNIC in vNICs:
        intf = AdapterIPSettings()
        intf.useDHCP = True
        intf.hardwareAddress = vNIC.hwConfig.macAddress
        intf.virtualDeviceKey = vNIC.key
        interfaces.append(intf)
    return interfaces

def deployVirtualMachine():
    twist = twistserver.TwistServer()
    twist.authenticate("hp", "opsware")
    vmTemplateService = twist.v12n.V12nVirtualServerTemplateService
    vmService = twist.v12n.V12nVirtualServerBaseService
    sourceTemplateRef = V12nVirtualServerTemplateRef(1520001)
    sourceTemplateVO =
vmService.getV12nVirtualServerBaseVO(sourceTemplateRef)
    deploySpec = constructDeploySpec(sourceTemplateVO)
    jobRef =
vmTemplateService.startDeploy(sourceTemplateRef,deploySpec,30*60,"Sample
Deploy VM",None, None);

deployVirtualMachine()
```

Pytwist details

This topic describes the behavior and syntax that is specific to Pytwist:

- ["Authentication modes" below](#)
- ["TwistServer method syntax" below](#)
- ["Error handling" on the next page](#)
- ["Mapping Java package names and data types to Pytwist" on the next page](#)

Authentication modes

The authentication mode of a Pytwist client is important because it affects the SA features and the resources that the client can access. A Pytwist client can run in one of the following modes:

- **Authenticated:** The client has called the `authenticate(username, password)` method on a `TwistServer` object. After calling the `authenticate` method, the client is authorized as the SA user specified by the `username` parameter, much like an end user who logs onto the SA Client.
- **Not Authenticated:** The client has not called the `TwistServer.authenticate` method. On a managed server, the client is authenticated as if it is the device that controls the Agent certificate. When used within a custom extension, a non-authenticated Pytwist client needs access to the Command Engine certificate. For more information on custom extensions and certificates, contact your technical support representative.

TwistServer method syntax

The `TwistServer` method configures the connection from the client to the Web Services Data Access Engine. (For sample invocations, see ["Pytwist examples" on page 67](#).) All of the arguments of `TwistServer` are optional. The following table lists the default values for the arguments.

Arguments of the TwistServer method

Argument	Description	Default
host	The hostname to connect to.	twist
port	The port number to connect to.	1032
secure	Whether to use https for the connection. Allowed values: 1 (true) or 0 (false).	1
ctx	The SSL context for the connection.	None. (See also "Authentication modes" above .)

When the `TwistServer` object is created, the client does not establish a connection with the server. Therefore, if a connectivity problem occurs, it is not encountered until the client calls `authenticate` or an SA API method.

Error handling

If the `TwistServer.authenticate` method or an SA API method encounters a problem, a Python exception is raised. You can catch these exceptions in an `except` clause, as in the following example:

```
# Create the TwistServer object.
ts = twistserver.TwistServer('localhost')
# Authenticate by passing an SA user name and password.
try:
    ts.authenticate('jdoe', 'secretpass')
except:
    print "Authentication failed."
    sys.exit(2)
```

Mapping Java package names and data types to Pytwist

The Pytwist interface is for Python, but the SA API is written in Java. Because of the differences between two programming languages a Pytwist client must follow the mapping rules described in this section.

In the SA API documentation, Java package names begin with `com.opsware`. When specifying the package name in Pytwist, insert `pytwist` at the beginning, for example:

```
from pytwist.com.opsware.compliance.sco import *
```

The SA API documentation specifies method parameters and return values as Java data types. The following table shows how to map the Java data types to Python for the API method invocations in Pytwist.

Mapping data types from Java to Python

Java data type in SA API	Python data type in Pytwist
Boolean	An integer 1 for true or the integer 0 for false.
Object[] (object array)	As input parameters to API method calls, object arrays can be either Python tuples or lists. As output from API method calls, object arrays are returned as Python tuples.
Map	Dictionary
Date	A long data type representing the number of milliseconds since epoch (midnight on January 1, 1970).

Automation Platform Extensions (APX)

This topic describes how to create and manage Automation Platform Extensions (APX), commonly just called *extensions*. APXs provide a framework that allows anyone familiar with script-based programming tools such as shell scripts, Python, Perl, and PHP, to extend the functionality of SA and create applications that are tightly integrated into SA. SA provides two types of APXs:

- **Program APXs** (also called **Script APXs**) run in the Global File System (OGFS) and can use all of the OGFS functionality. You can use typical programming practices to leverage the SA API and access a core's Managed Servers to implement new custom functionality. For example, you could write an APX that gathers BIOS information from managed servers and populates custom fields using shell commands. See "[Program APXs](#)" on page 84.
- **Web APXs** allow you to create a web-based application, where either an Apache 2.x process or a CGI/PHP script is called using GET or POST URL. Web APXs can contain static web resources such as images, and can employ CGI or PHP for dynamic content generation. See "[Web APXs](#)" on page 85.

APXs allow you to access data about your managed environment and share and process that data with web applications, scripts, programs and other applications. Below are some of the benefits of APXs:

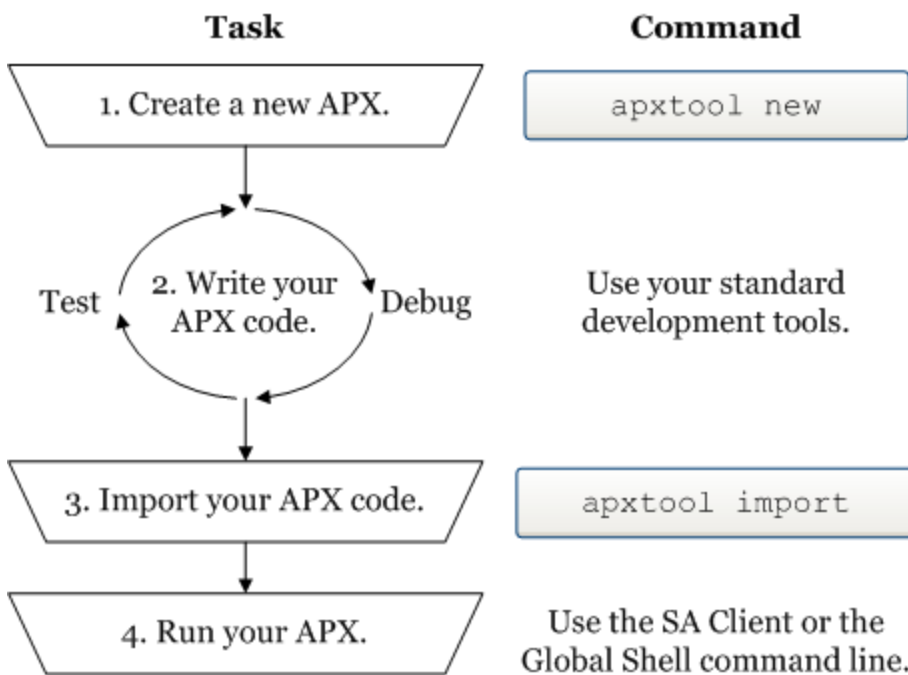
- Listed in the SA Library and can be used from the SA Client.
- Uniquely identified and managed through versioning.
- Secure because they take full advantage of SA's security model. When needed, APXs can securely and temporarily escalate a user's permissions beyond the normal defaults during the APX session.
- Scalable within and across SA cores.
- You can schedule them to be pushed automatically to servers.
- Auditable.
- Able to persist through an upgrade of the SA platform. APXs do not have to be rewritten after an upgrade.

For information on using APX extensions, see the Running Extensions to SA section in the SA 10.51 User Guide. See also the SA Global Shell section in the SA 10.51 User Guide because you can also run APX extensions from the SA Global Shell.

Creating an APX

The following diagram shows the basic steps to creating an APX and the corresponding commands to use. For a tutorial on how to create a web APX, see ["Tutorial: Creating a Web application APX" on page 107](#). For a tutorial on how to create a program APX, see ["Tutorial: Creating a program APX" on page 114](#).

Creating an APX



1. To create a new APX, use the `apxtool new` command. This command creates a set of template files you can edit to create your own APX.

You can optionally register your new APX with the `apxtool new` command. Registering your APX reserves the name of your APX in SA. If you do not register your APX at this step, you can register it with the `apxtool import` command in step 3 below.

See ["apxtool command" on page 93](#).

2. After creating APX template files, develop your APX code by modifying the template files created by the `apxtool new` command and possibly adding your own files. You can test your APX code to make sure it is running correctly.

3. When your APX code is tested, you must import it into SA with the `apxtool import` command.
4. Run your APX either from the SA Client or from the Global Shell command line.
 - From the SA Client: Select **Library > By Type tab > Extensions > Program**. Select an APX. Select the **Actions > Run menu**.
 - From the Global Shell command line: Open the Global Shell from the SA Client by selecting the **Tools > Global Shell menu**. Run your APX by entering the command:

```
/opsw/apx/bin/<APX name>
```

For more information, see *Running Extensions to SA* and the *SA Global Shell* sections in the *Server Automation Using Guide* on the HPE SSO portal.

To create an APX extension that is intended to run on VMware ESXi servers, the APX extension must communicate with the ESXi server remotely using its web services interface. For more information on VMware ESXi servers, see the *Virtual Server Management* section in the *Server Automation Using Guide* on the HPE SSO portal.

Program APXs

Program APXs, also called Script APXs, are similar to shell commands and are implemented as OGFS server scripts. You can invoke them from the OGFS command line and pass input arguments to them using STDIN or command-line arguments. Their output goes to STDOUT and STDERR.

Program APXs are executed inside a Global Shell (OGSH) session and have access to all OGSH features permissible to the user who invokes the APX. This includes rosh, CLI, OGFS, and more. You can write Program APXs using any script-based tool, such as shell script, Python, Perl, and so on.

You can invoke Program APXs from the OGSH command prompt. Typically, Program APXs are executed synchronously, meaning the shell prompt does not return until the Program APX returns. APXs cannot be scheduled as recurring jobs in either the twister or in OGFS.

Program APXs are located in the OGFS directory `/opsw/apx/bin`.

During an interactive OGSH session, a user only sees those Program APXs in `/opsw/apx/bin` that they have permission to execute. Attempting to invoke a Program APX for which a user has no execution permission results in a `File Not Found` error from the shell.

A Program APX can also be invoked by other Web APXs or Program APXs. For example, a CGI program or PHP script from a Web APX can invoke a Program APX.

Web APXs

Web APXs are implemented using CGI programs or PHP scripts. These CGI programs and PHP scripts are executed inside a user-specific OGS session. They may access SA facilities such as rosh, the SA API, CLI, or any commands allowable from within an OGS session. Web APXs are served by a built-in Apache web server with a PHP module enabled.

You can access Web APXs in two ways: using a stand-alone web browser such as Internet Explorer or Firefox, or from the SA Client. Microsoft ActiveX is not supported.

Invoking a Web APX from a stand-alone Web browser the first time will trigger a login dialog that requires verification of the SA user credentials. Invoking a Web APX from the SA Client does not require additional login. Web APXs can be used to build user Interfaces for custom customer applications.

To launch APXs using Microsoft Internet Explorer versions 6 and 7 on Windows Server 2003, 2008 and 2012 with Enhanced Security Configuration enabled, the SA Client URL must first be added to Internet Explorer's trusted site list.

APX user roles

There are three general roles of APX users as shown in the following table:

APX user roles

User role	Description
End User	Runs APXs. This user typically does not have permission to modify an APX or see its content.
APX Developer	Creates and publishes APXs. This class of users can import and export APXs, and can modify APX content.
APX Administrator	Determines APXs users are permitted to run. These users assign executable permission to run an APX by managing folder permissions. APX Administrators may not have permission to modify the APX itself, but can have the permission to view APX content in order to determine which APXs to make executable.

APX permissions

APXs requires that you have the SA Client Feature permission **Manage Extensions**. A user group can be given one of the permissions:

- Manage Extensions: Read
- Manage Extensions: Read & Write
- Manage Extensions: None

APX feature permissions

Automation Platform Extension	
Name	Permission
Manage Extensions	<input type="radio"/> Read <input checked="" type="radio"/> Read & Write <input type="radio"/> None

These feature permissions apply only to APX developers and administrators, they do not apply to those users who only need to run APXs.

- **Read** permission grants the ability to display the APX source contents or to export (download) the APX source archives.
- **Read & Write** permission grants the ability to modify the contents of an APX in addition to read access.
- **None** permission denies all access to the APX source.

In addition to the SA Client Feature **Manage Extensions** permission, folder permissions (list, read, write, execute) must be used to determine which APXs a user has access to.

APX permissions

Permission	Description
List	Permission to list the system's APXs.
Read	Permission to view APX contents.
Write	Permission to modify APX content and to import and export APXs.
Execute	Permission to run APXs and view APX properties.

The following table shows a matrix of how permissions are determined based on the combination of the Manage Extensions feature permissions and folder permissions.

APX permission matrix

Folder Permission:	Manage Extensions Permission:		
	Read	Read & Write	None
List	List APXs	List APXs	List APXs
Read	Export APXs	Export APXs	List APXs
Write	Export APXs	Import, export APXs	List APXs
Execute	Run APXs	Run APXs	Run APXs

Like other SA features, you can grant a user access to an APX and specify to which managed servers and/or policies the user can apply the APX.

If a user attempts to access a Web APX for which he does not have execution permission, the Web browser will receive an HTTP 403 Forbidden return code.

For more information on SA permissions, see the Server Automation Administration Guide on the HPE SSO portal.

Permission escalation

When executing an APX, the user has only the privileges to access resources and operations granted in SA. However, in some cases, it will be necessary to temporarily grant the user *escalated permissions*, privileges beyond the SA privileges, while executing an APX. You can explicitly grant certain privileges to users, over-and-above their default SA privileges, on a temporary basis while running an APX. Permission escalation is transparent to the user running the APX.

For example, you may want a user to be able to run a BIOS information gathering application on a managed server, but the user does not have the permissions granted to do so. You can write an APX for a user without the privileges required to run the BIOS gathering application that temporarily grants that user the required privileges. The user's privileges return to the default after the APX ends its run.

Privilege escalation is specified in the file `apx.perm` file. For more information, see ["APX permissions escalation configuration file - apx.perm" on page 104](#).

APX structure

An APX has the following attributes:

- `APX type`: Either Program APX (also called Script APX) or Web APX.
- `APX unique name`: This is the full name of the APX that must be unique. For example, `com.hpe.sa.RestartMyApp`.
- `APX display name`: This is usually a shorter name than the APX unique name. For example, `RestartMyApp`.
- `APX version`: You can maintain multiple versions of your APX by setting a version string or you can let SA manage versions for you automatically. The APX version can be a simple number such as version 1, 2, 3, and so on, or it can be any alphanumeric string.

See ["Importing an APX into SA - apxtool import" on page 98](#) and ["Setting the current version of an APX - apxtool setcurrent" on page 101](#) for more information.

File structure

To SA, an APX is just a set of files and directories that conform to the contract of the APX type (Program APX or Web APX) such that the APX runtime can properly execute it. For example, a Web APX may need an `index.html` file or an `index.php` file. A Program APX may require a shell command with the same name as the APX.

For more information on the files in an APX, see ["APX files" on page 102](#).

OGFS integration

The APX infrastructure depends on the OGFS to manage user sessions and to expose various parts of the APX in the SA file system. The following sections describe how APX is integrated into the OGFS and its various applications.

APX Executable Directory

Program APXs are treated as executable programs in the Global Shell, OGS. These APXs are exposed as an executable command in the OGS. This allows a shell user to invoke the APX as if running a shell command.

The APX executable directory has the following format:

```
/opsw/apx/bin/{apx_name}
```

where `apx_name` is the name of the APX. Running `apx_name` in `/opsw/apx/bin/{apx_name}` invokes the current version of `apx_name`.

APX Runtime Directory

The APX Runtime directory is used by the APX runtime to support execution of an APX. The APX Runtime directory must have access to the APX source. In addition, users who have developer privileges and have read permission to an APX can also access the APX. The APX Runtime directory is not available for non-APX developers in the Global Shell.

The APX Runtime directory references the source of the current version of an APX. It has the format:

```
/opsw/apx/runtime/{apx_type}/{apx_name}
```

where `apx_type` can be `script` or `web`.

APX interfaces - Defining categories of APX extensions

APX interfaces enable you to create named categories of APXs and to find all the APXs of a given category. An interface is the name of the category. For example, you could create a category of APXs that all take a certain set of input parameters and produces a certain type of output data. Or you could create a category of APXs that all perform a specific set of operations.

You can also create an APX or an external application that gets the names of all APXs of the desired category and executes them. Or the APX or application could just present the list of APXs of the desired category and let the user select one to execute.

An APX interface is a name that defines an informal contract between the caller of an APX and the APX.

- An APX that **defines an interface name** creates a category of APX with that name.
- An APX that **implements an interface** declares itself to be an APX of that category.

A sample interface

SA provides an interface named `RightClickToRun`. This interface defines a category of APX that takes one or more devices as input parameters and runs against those devices. In addition, the SA Client displays all APXs that implement this interface in the **Actions > Run Extension** menu, which allows

users to select one or more devices and run these APXs against the selected devices. For more information on this interface, see ["RightClickToRun interface" on the next page](#).

Defining an interface

An APX interface defines the name of a category of APXs. All APXs that implement the interface belong to the category and must adhere to the conventions of the interface. To create a new category, you make your APX “define” the interface.

To make your APX define an interface, perform the following steps:

1. Create the APX with the `apxtool new` command. For details on this command, see ["Creating a new APX - apxtool new" on page 95](#).
2. Locate the files of your new APX and open the file named `interfaces` in a text editor. The `interfaces` file is located in the APX-INF directory of your APX directory.
3. At the end of the `interfaces` file, add three lines for:
 - The name of the interface section in the file. This is the unique name of the interface.
 - The display name of the interface.
 - A description of the interface.

For example, the following shows the interface section name, the display name and the description of the interface named “com.hpe.sa.MyNewInterface”:

```
[com.hpe.sa.MyNewInterface]
name=MyNewInterface
description="This is a simple interface for testing purposes."
```

4. Save your changes and close the file.
5. Import your modified APX into SA with the `apxtool import` command. For details on this command, see ["Importing an APX into SA - apxtool import" on page 98](#).

To upgrade an existing APX to define an interface you must create the `interfaces` file and add your interfaces as described above.

Implementing an interface

An APX interface specifies a category of APX that adheres to the conventions of the interface. To specify that your APX belongs to a category, you make your APX “implement” the interface. To make your APX implement an interface, perform the following steps.

1. Create the APX with the `apxtool new` command. For details on this command, see ["Creating a new APX - apxtool new" on page 95](#).
2. Locate the files of your new APX and open the file named `apx.cfg` in a text editor.
3. Locate the section in your `apx.cfg` file that discusses the "Implementing" section. This section briefly describes how to specify the interfaces that your APX implements.
4. Locate the following lines in the file `apx.cfg`:

```
[Implementing]
interfaces=
```

5. Modify the `interfaces=` line and add the name of your interface at the end of the line. For example, if your APX implements the interface named "com.hpe.sa.MyNewInterface", the `apx.cfg` file would contain the following lines:

```
[Implementing]
interfaces=com.hpe.sa.MyNewInterface
```

To implement more than one interface, add them to the `interfaces` line separated by colon, as follows:

```
[Implementing]
interfaces=com.hpe.sa.MyNewInterface:com.hpe.sa.AnotherInterface
```

6. Save your changes and close the file `apx.cfg`.
7. Import your modified APX into SA with the `apxtool import` command. For details on this command, see ["Importing an APX into SA - apxtool import" on page 98](#).

You must set the current version of the APX to see the implemented interfaces when viewing the APX in the SA Client or with the `apxtool query` command. For more information, see ["Setting the current version of an APX - apxtool setcurrent" on page 101](#).

To upgrade an existing APX to use an interface you must add your interfaces to your existing `apx.cfg` file as described above.

RightClickToRun interface

SA provides an interface you can use with your APXs named `com.hpe.client.server.RightClickToRun`. This interface works only with program APXs, not with web APXs. Use this interface when you want your APX to do all of the following:

- Take one or more devices as input parameters to the APX. APXs that implement this interface must take “-d <device id>” as an input argument.
- Appear in the **Actions > Run Extension >Select Extension...** window.
- Appear in the **Actions > Run Extension** menu of the SA Client. APXs appear in this menu after they have been run once using the **Actions > Run Extension >Select Extension...** menu.

To execute an APX from the **Actions > Run Extension** menu, the user must have execute permission on the APX. Any APX the user does not have permission to execute will not appear under this menu item. For information on permissions, see the Server Automation Administration Guide on the HPE SSO portal.

The RightClickToRun interface lets users select one or more devices in the SA Client and run your APX against those devices.

When you select the **Actions > Run Extension** menu item, the SA Client displays all of the program APXs that implement the interface `com.hpe.client.server.RightClickToRun`. When you select an APX, it is run against all the selected servers. The APX will be invoked once for each selected server.

For instructions on making your APX implement this interface, see ["Implementing an interface" on page 90](#). For details on using an APX that implements this interface, see the Running SA Extensions section in the Server Automation Using Guide on the HPE SSO portal.

CoreAffinity interface

SA provides an interface that you can use with your APXs named ‘`com.hpe.client.server.CoreAffinity`’. You can use this interface when you want to run your APX in CoreAffinity mode.

CoreAffinity mode only applies when you have a mesh with at least two SA cores. When this mode is enabled for each target server, the APX is executed on the SA core to which this target server is registered, regardless of where the actual job was started.

For example:

- You have a mesh with two cores, core A and core B
- You start an APX job from core A on two target servers MA (registered to core A) and MB (registered to core B)

In core affinity mode this job runs the APX for MA on core A and MB on core B. If CoreAffinity is disabled then both executions will be done on core A (because that is where the job started).

For instructions on making your APX implement CoreAffinity interface, see ["Implementing an interface" on page 90](#).

Using the Interface API

You can use the SA API to integrate your own applications with SA and APXs. Your application can determine all the APXs that implement a particular interface by using the interface named APXInterfaceService in the package named com.opsware.apx in the SA API. ["API Documentation and the Twister" on page 27](#) on using the SA API.

apxtool command

Use the apxtool command in an OGFS session to create and manage APXs. The apxtool command is available in the Global Shell in the directory `/opsw/bin/apxtool`.

For a tutorial on how to use the apxtool to create a web APX, see ["Tutorial: Creating a Web application APX" on page 107](#).

Syntax of apxtool

Invoke the APX tool from the OGFS command line as follows:

```
apxtool [-h | --help] {function} arguments
```

To obtain a complete list of commands and arguments supported by the APX tool, run apxtool from an OGS command line with no arguments.

The APX Tool supports the following major functions:

APX tool functions

Function	Usage
new	Creates a new APX source directory and a new set of template files in the OGFS. Optionally registers the APX into SA. Registering assigns an APX ID and makes the name of your APX available to others (with appropriate permissions) using SA. See "Creating a new APX - apxtool new" on page 95 for more information.
import	Imports your APX files into the SA Library and creates a new version of your APX. Optionally registers the APX into SA. Registering assigns an APX ID and makes the name of your APX available to others (with appropriate permissions) using SA. See

APX tool functions, continued

Function	Usage
	"Importing an APX into SA - apxtool import" on page 98 for more information.
setcurrent	Sets the current version of an APX in the SA Library. You can have multiple versions of an APX in SA, but only the current version can be executed. See "Setting the current version of an APX - apxtool setcurrent" on page 101 for more information.
query	Displays information about an APX. See "Querying APX information - apxtool query" on page 99 for more information.
export	Copies all of an APXs files from the SA Library to a separate set of files.
delete	Deletes an APX from the SA Library.

Using short and long command options

Most of the options to the apxtool command accept a short form or a long form.

- The short form is a single hyphen and a character, for example, “-t” and “-v”.
- The long format is two hyphens followed by a word, for example, “--type” and “--view”.

Some options require an argument following the option. For example, “-t webapp” and “-t details”. Arguments can be specified in one of four formats, which are all equivalent. To illustrate, the following commands are equivalent and produce the same results:

```
apxtool query -t webapp
apxtool query -twebapp
apxtool query -tw
apxtool query --type webapp
apxtool query --type=webapp
```

Some options only require typing a minimum number of characters, enough to identify the option argument. For example, in the query function, the --view option requires argument “list”, “details”, “versions”. The following commands produce the same result:

```
apxtool query --view=details
apxtool query --view=d
apxtool query -vdetails
apxtool query -vd
```

Creating a new APX - apxtool new

You can use the APX tool to create a new APX and optionally register the name of the APX into SA. This command creates a set of template files for an APX that you can modify. For information on the files that make up an APX, see ["APX files" on page 102](#).

Usage

```
apxtool new [options] {src_dir}
```

where the `src_dir` argument specifies the directory where the template files of the new APX are to be created. If this argument is omitted, the template files are placed into the current directory.

The following [table](#) lists the options for creating a new APX:

Options for apxtool new

Option	Usage
-h, --help	Show this help message and exit.
-t <type> --type=<type>	(Required) The APX type. Valid values are: <code>script</code> or <code>webapp</code> . For example, <code>-ts</code> for script APX, <code>-tw</code> for web APX. (A script APX is also known as a program APX.)
-u <unique name> --uniquename=<unique name>	(Required) The unique name of the APX. A unique name is a dot separated name that conforms to file system format. It must have at least one dot. Valid characters are: <code>[a-zA-Z0-9_.</code>]. Example: <code>com.hpe.sa.security.scan_ports</code>
-n <name> --name=<name>	(Optional) The display name of the APX in a folder. If a name is not specified, but a unique name is specified, the last part of the APX unique name is used as the display name. Note that this name must be unique within the specified folder. For example, if the unique name were <code>com.hpe.sa.MyWebExt</code> , the default display name would be <code>MyWebExt</code> .
-d <description> --description=<description>	(Required) A brief description of an APX. If the description is a filename with the extension <code>.txt</code> , the file is assumed to be a text file and its content is used as the APX description.
-r	(Optional) Registers the name of the APX into the system. If you

Options for apxtool new, continued

Option	Usage
--register	specify this option, you must also specify <code>-f</code> or <code>--folder</code> . If you do not specify <code>-r</code> and <code>-f</code> with <code>apxtool new</code> , you must use <code>-f</code> with <code>apxtool import</code> .
<code>-f <path></code> <code>--folder=<path></code>	(Optional) The SA folder path where the APX will be registered. This can be a full path, partial path, absolute path, or relative path, as long as it can uniquely identify a specific folder. This option is only needed if <code>-r</code> or <code>--register</code> is used. If you do not specify <code>-r</code> and <code>-f</code> with <code>apxtool new</code> , you must use <code>-f</code> with <code>apxtool import</code> .
<code>-Q, --quiet</code>	(Optional) Suppresses output messages.
<code>-F, --force</code>	(Optional) Suppresses confirmation prompts.

Deleting an APX - apxtool delete

You can use the APX tool to delete an existing APX from the SA library.

Usage

```
apxtool delete [options]
```

The following [table](#) lists the options for deleting an APX:

Options for apxtool delete

Option	Usage
<code>-h</code> <code>--help</code>	Show this help message and exit.
<code>-t <type></code> <code>--type=<type></code>	(Required) APX type. Valid values are: <code>script</code> or <code>webapp</code> . For example <code>-ts</code> for <code>script</code> .
<code>--id=<APX id></code>	(Optional) The object identifier of the desired APX.
<code>-u <unique_name></code> <code>--</code> <code>unique_name=<unique_</code>	(Optional) The unique name of the APX. A unique name is a dot separated name that conforms to file system format. It must have at least one dot. Valid characters are: <code>[a-zA-Z0-9_.</code>].

Options for apxtool delete, continued

Option	Usage
name>	Example: com.hpe.sa.security.scan_ports
-n <name>, --name=<name>	(Optional) APX display name in a folder.
-f <path>, --folder=<path>	(Optional) SA folder path. Path can be a full path, partial path, absolute, or relative, as long as it can uniquely identify a specific folder.
-Q, --quiet	(Optional) Suppresses output messages.
-F, --force	(Optional) Suppresses confirmation prompts.

Exporting an APX from SA - apxtool export

You can use the APX tool to export an APX. Export downloads a specific version of an APX source archive file and places the files into a directory or into a .zip archive file.

Usage

```
apxtool export [options] {target_dir}
```

where the argument `target_dir` is the directory into which the APX source archive file is copied or into which the APX source archive content is expanded, depending on whether or not the `--archive` option is specified. If omitted, the current directory is used.

The following [table](#) lists the options for exporting an APX.

Options for apxtool export

Option	Usage
-h, --help	Show this help message and exit.
-t <type>, --type=<type>	(Required) APX type. Valid values are: <code>script</code> or <code>webapp</code> . For example, <code>-ts</code> for <code>script</code> .
--id=<APX id>	(Optional) The object identifier of the desired APX.
-u <unique_name>, --unique_name=<unique_name>	(Optional) The unique name of the APX. A unique name is a dot separated name that conforms to file system format. It must have at least one dot. Valid characters are: <code>[a-zA-Z0-9_.</code>].

Options for apxtool export, continued

Option	Usage
	Example: <code>com.hp.e.sa.security.scan_ports</code>
<code>-n <name>, --name=<name></code>	(Optional) APX display name in a folder.
<code>-f <path>, --folder=<path></code>	(Optional) SA folder path. Path can be a full path, partial path, absolute, or relative, as long as it can uniquely identify a specific folder.
<code>-v v<ersion_string>, --version=<version_string></code>	(Optional) This option specifies which APX version to download. If omitted, the current version is downloaded.
<code>-a, --archive</code>	If specified, export the APX source in its original source archive as a ZIP or JAR file.
<code>-Q, --quiet</code>	(Optional) Suppresses output messages.
<code>-F, --force</code>	(Optional) Suppresses confirmation prompts.

Importing an APX into SA - apxtool import

You can use the APX Tool to import APXs. Import publishes a new version of an APX and optionally sets this version as the current version. If the APX has not been registered yet, this command also registers the APX.

Only the current version of an APX can be run. If you do not set the current version, the APX will not be runnable. You can set the current version with either `apxtool import` or with `apxtool setcurrent`. See "[Setting the current version of an APX - apxtool setcurrent](#)" on page 101 for more information.

Usage

```
apxtool import [options] {apx_src}
```

where `apx_src` can be an archived APX source file with extension `.zip` or `.jar` or it can be the name of a directory containing the APX files to be published. `apx_src` may be a relative or absolute path. If omitted, the current directory is used. The specified directory or archive file must contain the directory `APX-INF`.

The following [table](#) lists the options that are available when importing an APX:

Options for apxtool import

Option	Usage
-h, --help	Show this help message and exit.
-c, --setcurrent	If specified, set the newly published version as the current version of an APX.
-- version=<version_ string>	The new version of this APX. This option must not be used if version_string is already specified in apx.cfg. If no version is specified, one will be assigned automatically.
-f <path>, -- folder=<path>	(Optional) SA folder path. Path can be a full path, partial path, absolute, or relative, as long as it can uniquely identify a specific folder. If you did not specify -r and -f with apxtool new, you must use -r with apxtool import.
-Q, --quiet	(Optional) Suppresses output messages.
-F, --force	(Optional) Suppresses confirmation prompts.

Querying APX information - apxtool query

You can use the APX Tool to get and view APX information. You can specify additional options to limit resulting APXs. Multiple occurrences of the same option form a logical OR expression. If no matching result is found, this command returns exit code 100.

Usage

```
apxtool query [options]
```

The following [table](#) lists the options that are available when querying APX information:

Options for apxtool query

Option	Usage
-h, --help	Show this help message and exit.
-v <view>, -- view=<view>	(Optional) Select one of the predefined views of the query results. Choices are <code>list</code> (default), <code>details</code> , and <code>versions</code> . -v <code>list</code> is a single line representation of APX basic information presented in tabular format.

Options for apxtool query, continued

Option	Usage
	<p><code>-v details</code> is a multiple line representation of APX information.</p> <p><code>-v versions</code> lists all APX versions. You would only need to specify enough characters for the view type; for example, <code>-vd</code>, is the same as <code>-v details</code>. If the versions layout is selected, the query must result in a single APX object.</p>
<code>-t <type>, --type=<type></code>	<p>(Optional) Specifies the type of APX to display. Valid values are: <code>script</code> or <code>webapp</code> or <code>interface</code>. The default is to display all types.</p> <p><code>-t script</code> displays all script APXs.</p> <p><code>-t webapp</code> displays all web APXs.</p> <p><code>-t interface</code> displays all APXs that define one or more interfaces.</p> <p>For example, <code>apxtool query -ts</code> displays all the script APXs.</p>
<code>--id=<APX id></code>	<p>(Optional) The object identifier of the desired APX.</p>
<code>-u <unique_name> -- uniquename=<unique_name></code>	<p>(Optional) The unique name of the APX. A unique name is a dot separated name that conforms to file system format. It must have at least one dot. Valid characters are: <code>[a-zA-Z0-9_]</code>.</p> <p>Example: <code>com.hp.e.sa.security.scan_ports</code></p>
<code>-n <name>, --name=<name></code>	<p>(Optional) APX display name in a folder.</p>
<code>-f <path>, --folder=<path></code>	<p>(Optional) SA folder path. Path can be a full path, partial path, absolute, or relative, as long as it can uniquely identify a specific folder.</p>
<code>--current</code>	<p>(Optional) if specified, only query APX objects that have a current version set.</p>
<code>--format=<format_string></code>	<p>(Optional) This advanced option allows you to specify custom display formatting for an APX listing.</p> <p><code>format_string</code> is a string containing embedded tag names that are substituted with values at display time. Tag names must have a format of <code>%(tag_name)</code>.</p> <p>Use the format string <code>"__show_tags__"</code> to display a list of all the supported tag names.</p>
<code>--csv</code>	<p>(Optional) Displays the output in comma-separated values format. Ignored if the <code>--format</code> option is specified.</p>
<code>-Q, --quiet</code>	<p>(Optional) Suppresses extraneous output messages.</p>

Setting the current version of an APX - apxtool setcurrent

You can use the APX tool to set an APX version as the current version.

Only the current version of an APX can be run. If you do not set the current version, the APX will not be runnable. You can set the current version with either `apxtool import` or with `apxtool setcurrent`. See ["Importing an APX into SA - apxtool import" on page 98](#) for more information.

Usage

```
apxtool setcurrent [options] {version_str}
```

where the argument `version_str` is required to uniquely identify an existing version of an APX.

The following [table](#) lists the options that are available when setting an APX version:

Options for apxtool setcurrent

Option	Usage
-h, --help	Show this help message and exit.
-t <type>, --type=<type>	(Required) APX type. Valid values are: <code>script</code> , <code>webapp</code> . For example, <code>-ts</code> for <code>script</code> .
--id=<APX id>	(Optional) The object identifier of the desired APX.
-u <unique_name>, --uniquename=<unique_name>	(Optional) APX unique name. A unique name is a dot separated name that conforms to file system format. It must have at least one dot. Valid characters are <code>[a-zA-Z0-9_.</code>]. Example: <code>com.hpe.sa.security.scan_ports</code>
-n <name>, --name=<name>	(Optional) APX display name in a folder.
-f <path>, --folder=<path>	(Optional) SA folder path. Path can be a full path, partial path, absolute, or relative, as long as it can uniquely identify a specific folder.
-Q, --quiet	(Optional) Suppresses output messages.
-F, --force	(Optional) Suppresses confirmation prompts.

Error handling

The APX tool command conforms to the standard POSIX convention and returns 0 on success and a non-zero value for other errors. The APX tool sends normal output to STDOUT and errors and warnings to STDERR. When an error occurs, the APX tool typically returns a descriptive message to STDERR.

Error conditions are typically categorized as shown in the following [table](#):

APX Tool Error Conditions

Return Code	Description
0	Success
1	Syntax or usage error
2	Permission related error
3	User canceled operation
4	Runtime error

There may be other undocumented exit codes. The only guarantee is that if the exit code is 0, the command completed its operation successfully.

APX files

This section describes the template files created when you run the `apxtool new` command. The following table summarizes these files. The sections below describe some of the files in more detail.

APX files

File name	Description
<code>apx.cfg</code>	APX configuration file, contains metadata that fully describes the APX. See "APX configuration file - apx.cfg" on the next page for more information.
<code>apx.perm</code>	APX permissions file, specifies permission escalation rules. See "APX permissions escalation configuration file - apx.perm" on page 104 for more information.
<code>description.txt</code>	Text description of the APX. Specified with the <code>apxtool new -d</code> option. See "Creating a new APX - apxtool new" on page 95 for more information.
<code>interfaces</code>	APX interface definition file. Specifies the interfaces the APX defines or implements. See "APX interfaces - Defining categories of APX extensions" on

APX files, continued

File name	Description
	page 89 for more information.
usage.txt	Text description of how to use the APX.
run.sh	For program APXs only, this file contains the executable code of the APX. This file contains the functionality of the program APX. For an example, see "Tutorial: Creating a program APX" on page 114 for more information.
index.php	For web APXs only, this file contains the PHP source code for the web APX. This file contains the functionality of the web APX. For an example, see "Tutorial: Creating a Web application APX" on page 107 for more information.

APX configuration file - apx.cfg

All APXs regardless of type must have a configuration file named `apx.cfg`. The `apxtool new` command creates a template of this file for you to modify. This file contains metadata that fully describes the APX. The `apx.cfg` uses a "key=value" format to define the properties of the APX. Multiple lines are joined together with a line continuation character, "\".

The ["APX configuration file attributes" below](#) table describes common attributes for all APXs. APX type specific attributes are described in the corresponding APX type functional specifications. Note that some of the attributes may be extracted from the `apx.cfg` configuration file and managed in SA. For modifiable attributes such as the description, subsequent updates of the `apx.cfg` file will update the SA managed data accordingly.

To see an example `apx.cfg` file, run the `apxtool new` command and open the files it creates.

APX configuration file attributes

Attribute	Modifiable?	Description
type	No	The type of the APX, which must be either <code>webapp</code> or <code>script</code> . (Script APXs are also known as Program APXs.) Once created, you cannot change the APX type.
name	Yes	This is the APX display name and may contain multi-byte characters. This name can be changed at any time. This name will be listed in the SA Client APX folders.
unique_name	No	The unique name of the APX. This name will be used as the file name for the APX as it appears in the OGFS. This name together with the type forms a key that uniquely identifies an APX. Once created, the name cannot be changed. Since this name is used in the file system, it must conform to the file system naming specification. Generally,

APX configuration file attributes, continued

Attribute	Modifiable?	Description
		this name should be in ASCII.
version	Yes	The version string representing the current version of the APX. If the value begins with the string "auto:", then SA will automatically manage the versions using an integer incremented for each new version.
description	Yes	A text description of what the APX does. You can alternatively use the file <code>description.txt</code> instead of this attribute.
usage	Yes	A text description describing how to use the APX. You can alternatively use the file <code>usage.txt</code> instead of this attribute.
interfaces	Yes	One or more interfaces the APX implements. Separate multiple interfaces with a colon (:) character.
command	Yes	The executable file the APX is to run when it is invoked.

APX permissions escalation configuration file - apx.perm

Use the file `apx.perm` to specify permission escalation rules. If this file does not exist, or if it contains no escalation permissions, the APX will run with the user's default permissions.

When a new APX is created using the APX Tool's **New** command, it generates certain default files, including a default `apx.perm` file, which by default has no escalation permissions defined. The default file does contain some commented out examples which an APX developer can use as templates.

There are three ways to specify escalations, described below.

- ["No escalation" on the next page](#)
- ["All permissions" on the next page](#)
- ["With escalation" on the next page](#)

No escalation

The `escalations` attribute is not specified. The APX runtime uses the current user privilege to execute an APX. If an APX invokes privileged operation which a user does not have, APX execution will terminate with an error.

All permissions

This is a special privilege that temporarily grants all operation permissions to a user. It is intended for development or demo use only. This is a useful tool for speedy proof of concept, or demo, without worrying fine grain permission tuning. It is a poor choice for a production environment due to its lack of security.

To grant all permissions, edit file `apx.perm` with a macro that matches all features with wildcard characters. For example:

```
use_feature(name="*")
```

With escalation

Specify a list of predefined common operations in the `apx.perm` file. When executing the APX, the APX runtime temporarily grants these permissions to the APX. SA has a comprehensive list of feature and resource permissions. To simplify the task of escalating related feature, one can use wildcard characters to match groups of related features. For example:

```
@use_feature(name="Application.*")
```

Showing the progress of an APX

You can use the `apxprogress` command in your program APX to provide information about the progress of your APX. This is useful for program APXs that run for a long period of time when you want to give the user status on the progress of your APX.

You can use a web APX as a front-end to the program APX and display the progress in the web APX.

apxprogress command

Use the `apxprogress` command to define the number of steps in the execution of a program APX and to record when each step has completed. This lets users of the APX know how far the APX has progressed and how much is remaining.

Syntax of apxprogress

```
apxprogress {option}...
```

Options to the apxprogress command

Option	Description
<code>-i <total number of steps></code>	Specifies the total number of steps the APX takes to run. Use this option once at the beginning of the APX to specify the total number of steps the APX will take. You can use this option multiple times in an APX to increase the number steps. Each use increments the total number of steps by the specified value.
<code>-c <current step></code>	Specifies the current step number. Call <code>apxprogress</code> with this option after each step in the APX code has completed.
<code>-m <message></code>	Specifies a text message describing the status of the APX.
<code>-a <data></code>	Specifies additional information the APX can make available about itself.
<code>-d</code>	Indicates debug mode. Displays the output of the command to stdout for debugging purposes.
<code>-h</code>	Displays help information about the <code>apxprogress</code> command.

Example shell script that uses apxprogress

The following shell script is part of a program APX that uses the `apxprogress` command. The APX defines a total of 100 steps and announces its current progress 100 times. Each time it also provides a message that includes the step number.

```
#!/bin/sh
#####
# A simple shell script for a program APX that displays progress
# about itself.
# Author: <name>
```

```
#####  
echo "This is a simple APX that uses apxprogress."  
totalsteps=100  
apxprogress -i $totalsteps -c 1  
for i in `seq $totalsteps`; do  
apxprogress -c $i -m "APX is running, working on step $i" -d  
sleep 10  
done
```

Viewing APX progress

You can use the SA API method `JobService.getProgress()` to access the progress information about a running APX that calls the `apxprogress` command. For an example showing this method, see ["Viewing the APX progress in the Twister interface" on page 120](#), which is part of the ["Tutorial: Creating a program APX" on page 114](#).

Tutorial: Creating a Web application APX

This tutorial demonstrates how to create, publish, and run a simple web application APX named `mywebapp`.

Running the default version of the APX created during this tutorial displays the output of the PHP command, `phpinfo`. Later the tutorial shows you how to modify the PHP code so that it displays a list of managed servers. Because the tutorial provides the source code, prior knowledge of PHP is not required.

Complete the following tasks in order.

1. ["Setting permissions and creating the tutorial folder" on the next page](#)
2. ["Creating a new web application" on page 109](#)
3. ["Importing the new web application into SA" on page 111](#)
4. ["Running the new web application" on page 111](#)
5. ["Modifying the web application" on page 113](#)
6. ["Running the modified web application" on page 114](#)

Tutorial prerequisites

To complete this tutorial, you must have the following capabilities and environment:

- The ability to log on to SA as `admin` or as another member of the **Super Administrators** group. Logging on as `admin` enables you to set permissions.
- The ability to log on to SA as a user who belongs to the **Advanced Users** group.
- Advanced users have permission to create and run the web application. In the example commands shown in this tutorial, the name of this user is `jdoh`.
- An understanding of how to set client feature permissions in the SA Client.
- For more information about permissions, see the "User and Group Setup" section in the the SA 10.51 Administration Guide.
- An understanding of how to create folders in the SA Client
- For details on folders, see the SA 10.51 User Guide.
- An understanding of how to open a Global Shell session.
- An understanding of basic Unix commands such as `ls` and `cd`.
- Experience developing web applications that run on HTTP servers.

Setting permissions and creating the tutorial folder

1. Log on to the SA Client as a member of the **Advanced Users** group and create the following folder in the SA Library:

```
/Dev/MyApp
```

Later in the tutorial, you will upload a web application into the MyApp folder. In the non-tutorial environment, the name of this folder is arbitrary. You can create or choose any other folder to contain your web applications.

2. Exit the SA Client.
3. Log on to the SA Client as `admin` and open the **Folder Properties** of the MyApp folder.
4. On the **Permissions** tab of **Folder Properties**, make sure that the **Advanced Users** group has the following permissions:

- List Contents of Folder
 - Read Objects Within Folder
 - Write Objects Within Folder
 - Execute Objects Within Folder
5. Exit the SA Client.

Creating a new web application

1. Open a Global Shell session as an SA user who belongs to the **Advanced Users** group.
2. In your core's OGFS home directory, create a directory named `mywebapp` and then change to that directory:

```
$ mkdir mywebapp  
$ cd mywebapp
```

The web application files will be stored in the `mywebapp` directory.

3. Using the `apxtool new` command, create the directory structure and default files for the web application as shown below.

```
$ pwd  
/home/jdoe/mywebapp  
$ ls  
$  
$ apxtool new -tw -d "This is my first app." \  
-u com.hpe.sa.jdoe.mywebapp  
Create source directory /home/jdoe/mywebapp/com.hpe.sa.jdoe.mywebapp? Y/N y  
Info: Successfully created APX 'mywebapp' source directory:  
/home/jdoe/mywebapp.
```

The `-tw` option indicates that the APX type is a web application, `-d` specifies a description, and `-u` specifies a unique name for the application.

For more information about the `apxtool new` command options, see the online help:

```
$ apxtool new -h
```

4. Change directories into the new directory created by the `apxtool new` command and list the files there.

```
$ pwd
/home/jdoe/mywebapp
$ cd com.hpe.sa.jdoe.mywebapp
$ ls
APX-INF cgi-bin css images index.php
$ ls -R
.:
APX-INF cgi-bin css images index.php
./APX-INF:
apx.cfg apx.perm description.txt interfaces usage.txt
./cgi-bin:
./css:
hp_sa.css
./images:
```

5. Display the contents of the default `index.php` file:

```
$ cat index.php
<?php
// Show information about PHP
phpinfo();
?>
```

As with other web applications, you can replace the `index.php` file with an `index.html` file. However, this tutorial uses the `index.php` file, which you will modify in a later section.

6. Examine some of the files in the `APX-INF` directory. For more information, see ["APX files" on page 102](#).

The `APX-INF` directory contains information that is specific to APX web applications. As shown by the following `cat` command, the `description.txt` file holds the text you specified with the `-d` option of `apxtool new`.

```
$ ls APX-INF/
description.txt apx.cfg apx.perm usage.txt
$ cat APX-INF/description.txt
This is my first app $
```

The following `grep` command shows some of the properties in `apx.cfg`, the APX configuration file. The values for `type` and `uniquename` result from the `-t` and `-u` options of the `apxtool new` command. For details on the APX configuration file, see ["APX configuration file - apx.cfg" on page 103](#).

```
$ grep "=" APX-INF/apx.cfg
type=webapp
name=mywebapp
unique_name=com.hpe.sa.jdoe.mywebapp
```

Importing the new web application into SA

Importing the web application performs the following actions:

- Installs the web application on an HTTP server within SA.
- Copies the web application to a folder that appears in the SA Library and in the Global Shell.
- Assigns a version number to the web application.

Enter the `apxtool import` command and respond to the prompts with `y`, as shown below. The `-f` option specifies the folder in the SA Library where the web application will be stored. The `-c` option sets the current version of the web application.

```
$ pwd
/home/jdoe/mywebapp/com.hpe.sa.jdoe.mywebapp
$
$ apxtool import -f "/Dev/MyApp" -c
APX source is not specified.
Do you want to publish current directory: /home/jdoe/mywebapp/
com.hpe.sa.jdoe.mywebapp? Y/N y
APX with unique name 'com.hpe.sa.jdoe.mywebapp' does not exist.
Register it into the system? Y/N y
Info: Successfully registered APX 'mywebapp' (310001) in folder '/Dev/
MyApp'.
Info: Successfully published a new version '1' for APX 'mywebapp'.
Info: Successfully set APX 'mywebapp'(310001) current version as '1'.
```

Running the new web application

Now that you have published the web application, you are ready to run it from the SA Client, just as an end-user would.

1. Log on to the SA Client as a user who belongs to the **Advanced Users** group.
2. Select the Library tab and the By Type tab.
3. Navigate to the **Extensions > Web** node where you should see the mywebapp extension.

If you do not see mywebapp, make sure that you have the necessary permissions as described in ["Setting permissions and creating the tutorial folder" on page 108](#).

4. To run the web application, select mywebapp. and select the **Actions > Run** menu.

The following figure appears. The web application displays the information generated by the phpinfo statement of the index.php file.

Web Application Version 1



Modifying the web application

Running the default `index.php` file is a good way to check your development environment, but it does not take advantage of SA functionality. In this section, you modify the `index.php` file so that it lists the names of servers managed by SA.

1. In the Global Shell session, locate the `index.php` file of the web application.

```
$ cd /home/jdoe/mywebapp/com.hpe.sa.jdoe.mywebapp
$ ls
APX-INF cgi-bin css images index.php
```

2. Open the `index.php` file in a text editor such as `vi`.
3. Replace the contents of `index.php` with the following lines:

```
<html>
<head>
<title>Servers</title>
</head>
<body>
<p>List of servers:</p>
<?php
passthru("ls /opsw/Server/@");
?>
</body>
</html>
```

The `passthru` statement above runs the `ls` command and passes `stdout` (without reinflates) back to the web page. The `ls` command lists the names of your managed servers as they appear in the OGFS.

4. Save the `index.php` file and exit the text editor.
5. Publish the modified web application.

The following `apxtool import` command sets the current version to 2. The `-F` option suppresses the confirmation prompts.

```
$ apxtool import -f "/home/jdoe/mywebapp/com.hpe.sa.jdoe.mywebapp" \
-c --version=2 -F
```

```
Info: Successfully published a new version '2' for APX 'mywebapp'  
Info: Successfully set APX 'mywebapp'(310001) current version as '2'.
```

Running the modified web application

1. In the SA Client, use the **View> Refresh** menu to refresh the view of your web extensions, which should now contain version 2 of mywebapp.
2. Select mywebapp and select the **Actions > Run** menu. The output should be similar to the Web Application Version 1 except it displays the output of the PHP passthru statement and the OGSH ls statement, which lists all your managed servers. Note that the passthru statement removes the line feeds that separate the server names returned by the ls command.

Tutorial: Creating a program APX

This tutorial demonstrates how to create, publish, and run a simple program APX named myshellapp that runs a simple shell script. Later the tutorial shows you how to modify the shell script to call the apxprogress command and provide progress information. Because the tutorial provides the source code, prior knowledge of shell programming is not required.

Complete the following tasks in order.

- ["Setting permissions and creating the tutorial folder" on the next page](#)
- ["Creating a new program APX" on the next page](#)
- ["Importing the new APX into SA" on page 117](#)
- ["Running the new APX" on page 118](#)
- ["Modifying the APX" on page 119](#)
- ["Running the modified APX" on page 120](#)
- ["Viewing the APX progress in the Twister interface" on page 120](#)

Tutorial prerequisites

To complete this tutorial, you must have the following capabilities and environment:

- The ability to log on to SA as `admin` or as another member of the **Super Administrators** group. Logging on as `admin` enables you to set permissions.
- The ability to log on to SA as a user who belongs to the **Advanced Users** group.
- Advanced users have permission to create and run the web application. In the example commands shown in this tutorial, the name of this user is `jdoue`.
- An understanding of how to set client feature permissions in the SA Client.
- For more information about permissions, see the "User and Group Setup section" in the SA 10.51 Administration Guide.
- An understanding of how to create folders in the SA Client
- For details on folders, see the SA 10.51 User Guide.
- An understanding of how to open a Global Shell (OGSH) session and use the Global Shell.
- An understanding of basic Unix commands such as `ls` and `cd`.

Setting permissions and creating the tutorial folder

1. Log on to the SA Client as `admin` and open the **Folder Properties** of the `MyApp` folder.
2. On the **Permissions** tab of **Folder Properties**, make sure that the **Advanced Users** group has the following permissions:
 - List Contents of Folder
 - Read Objects Within Folder
 - Write Objects Within Folder
 - Execute Objects Within Folder
3. Exit the SA Client.

Creating a new program APX

1. Open a Global Shell session as an SA user who belongs to the **Advanced Users** group.
2. In your core's OGFS home directory, create a directory named `myshellapp` and then change to that directory:

```
$ mkdir myshellapp
```

```
$ cd myshellapp
```

The program APX files will be stored in the `myshellapp` directory.

3. Using the `apxtool new` command, create the directory structure and default files for the program APX as shown below.

```
$ pwd
/home/jdoe/myshellapp
$ ls
$
$ apxtool new -ts -d "This is my first program APX." \
-u com.hpe.sa.jdoe.myshellapp
```

Create source directory under

```
'/home/jdoe/myshellapp/com.hpe.sa.jdoe.myshellapp' for APX 'myshellapp'? Y/N y
Info: Successfully created source directory
'/home/jdoe/myshellapp/com.hpe.sa.jdoe.myshellapp for APX 'myshellapp'.
```

The `-ts` option indicates that the APX type is a program APX (also called a script APX), `-d` specifies a description, and `-u` specifies a unique name for the application.

For more information about the `apxtool new` command options, see the online help:

```
$ apxtool new -h
```

4. List the files created by the `apxtool new` command:

```
$ pwd
/home/jdoe/mywebapp
$ ls
com.hpe.sa.jdoe.myshellapp
$ cd com.hpe.sa.jdoe.myshellapp
$ pwd
/home/jdoe/myshellapp/com.hpe.sa.jdoe.myshellapp
$ ls -R
.:
APX-INF run.sh
./APX-INF:
apx.cfg apx.perm description.txt interfaces usage.txt
```

5. Display the contents of the default `run.sh` file:

```
$ cat run.sh
#!/bin/sh

#####
# APX myshellapp
#
# Created by: jdoe
#
#####
echo "This is APX myshellapp"
```

6. Examine some of the files in the `APX-INF` directory. For more information on these files see ["APX files" on page 102](#).

The `APX-INF` directory contains information that is specific to APXs. As shown by the following `cat` command, the `description.txt` file holds the text you specified with the `-d` option of `apxtool new`.

```
$ ls APX-INF/
apx.cfg apx.perm description.txt interfaces usage.txt
$ cat APX-INF/description.txt
This is my first program APX.$
```

The following `grep` command shows some of the properties in `apx.cfg`, the APX configuration file. The values for `type` and `uniquename` result from the `-t` and `-u` options of the `apxtool new` command. For details on the APX configuration file, see ["APX configuration file - apx.cfg" on page 103](#).

```
$ grep "=" APX-INF/apx.cfg
type=script
name=myshellapp
unique_name=com.hpe.sa.jdoe.myshellapp
command=run.sh
```

Importing the new APX into SA

Importing the APX performs the following actions:

- Copies the APX to a folder that appears in the SA Library.
- Assigns a version number to the APX.

Enter the `apxtool import` command and respond to the prompts with `y`, as shown below. The `-f` option specifies the folder in the SA Library where the web application will be stored. The `-c` option sets the current version of the web application.

```
$ pwd
/home/jdoe/myshellapp/com.hpe.sa.jdoe.myshellapp
$
$ apxtool import -f "/Dev/MyApp" -c
APX source is not specified.
Do you want to publish current directory: /home/jdoe/myshellapp/
com.hpe.sa.jdoe.myshellapp? Y/N y
APX with unique name 'com.hpe.sa.jdoe.myshellapp' does not exist.
Register it into the system? Y/N y
Info: Successfully registered APX 'myshellapp' (20001).
Info: Successfully published a new version '1' for APX 'myshellapp'
Info: Successfully set APX 'myshellapp'(20001) current version as '1'.
```

Now that you have published the APX, you are ready to run it from the SA Client, just as another SA user would.

Running the new APX

Now that you have published the APX, you are ready to run it from the SA Client.

1. Log on to the SA Client as a user who belongs to the **Advanced Users** group.
2. In the navigation pane, select the Library tab, then the By Type tab.
3. Open the Extensions node and select the Program node. This displays all the program APXs in the SA Library. You should see your APX there. If you do not see `myshellapp`, make sure that you have the necessary permissions as described in ["Setting permissions and creating the tutorial folder" on page 115](#).
4. Select your APX.
5. Select the **Actions > Run** menu item. This displays the Run Program Extension wizard.
6. Select the Next button.
7. Select the Start Job button.
8. When your APX finishes, select the status indicator to display details.
9. Select the Close button.

Modifying the APX

In this section, you modify the `run.sh` file and add calls to the `apxprogress` command to provide progress information.

1. In the Global Shell session, locate the `run.sh` file of the APX.

```
$ cd /home/jdoe/myshellapp/com.hpe.sa.jdoe.myshellapp
$ ls
APX-INF run.sh
```

2. Open the `run.sh` file in a text editor such as `vi`.
3. Replace the contents of `run.sh` with the following lines:

```
echo "This is a simple APX that uses apxprogress."

totalsteps=100
apxprogress -i $totalsteps -c 1
for i in `seq $totalsteps`; do
    apxprogress -c $i -m "myshellapx is running, working on step $i" #-d
    sleep 10
done
```

These `apxprogress` commands specify that the APX has 100 steps and it calls `apxprogress` 100 times, once for each step, waiting ten seconds between calls. For more information, see ["Showing the progress of an APX" on page 105](#).

For debugging, you can change `"#-d"` to `"-d"` and run the shell script manually to display the output from the `apxprogress` commands on `stdout`.

4. Save the `run.sh` file and exit the text editor.
5. Publish the modified APX.

The following `apxtool import` command loads the new version of the APX and sets the current version to 2. The `-F` option suppresses the confirmation prompts.

```
$ apxtool import -f "/home/jdoe/myshellapp" \
-c --version=2 -F
```

```
Info: Successfully published a new version '2' for APX 'myshellapp'
```

```
Info: Successfully set APX 'myshellapp'(20001) current version as '2'.
```

Running the modified APX

Now that you have modified and republished the APX, run it from the SA Client as before.

1. In the SA Client, use the **View >Refresh** menu to refresh the view of the program extensions, which should now show version 2 of *myshellapp*.
2. Select your APX.
3. Select the **Actions > Run** menu item. This displays the Run Program Extension wizard.
4. Select **Next**.
5. Select **Start Job**.

Viewing the APX progress in the Twister interface

The `apxprogress` commands report the progress of the running APX. You can obtain this progress information by calling the API method `JobService.getProgress()`. This section shows you how to run this method from the Twister interface. For more information on the Twister interface to the SA API, see ["API Documentation and the Twister" on page 27](#).

1. In the SA Client, select the Jobs and Sessions tab.
2. Locate your APX in the list of jobs.
3. Note the Job ID number of your APX job. You will use this in a later step.
4. Run the SA Twist interface by entering the following URL into a web browser:

```
https://<core_host>:1032
```

where `<core_host>` is the IP address or host name of your SA core server. This displays the Twist interface to the SA API in the web browser.

5. Select the “Twister” link. This displays the Twister interface to the SA API where you can get complete information about API interfaces, packages and methods and where you can run methods.
6. Locate and select the `JobService` interface, which is in the `com.opsware.job` package.
7. Scroll down and locate the `getProgress()` method.
8. Select the Try It button just above the `getProgress()` method.
9. Enter your SA credentials.
10. Select **Login**.
11. In the “id” field, enter the job number of your running APX, from step 3 above.
12. Select **Go**. This calls the `getProgress()` method and displays the current progress information about your APX from the `apxprogress` command, as shown below. Notice that the total number of steps is 100 and the number of completed steps is 94 in this snapshot. For more information on the output from the `getProgress()` method, see the Javadocs documentation by selecting the

getProgress() method in the navigation pane of the Twister web browser.

JobService.getProgress()

(self) JobRef.	name	<input type="text"/>	(type: java.lang.String)
	id	2780001	(type: long)

Return type: com.opsware.job.JobProgress

Invocation took: 0.08 secs

errorCount: 0
totalCount: 1
doneCount: 0
elemProgressInfo:
[ObjectArray][size=1]
1. **message:**
 key: myshellapx is running, working on step 94
 values: null
 defaultMsg: myshellapx is running, working on step 94
 class: class com.opsware.job.JobMessageInfo
status: 0
error: null
element: [Server](#) : 0 <null>
stage:
 key: RUN
 values: null
 defaultMsg: RUN
 class: class com.opsware.job.JobMessageInfo
doneSteps: 94
totalSteps: 100
applicationData:
 class: class com.opsware.script.ScriptJobTargetProgress
active: true

Agent Tools

Agent Tools is a suite of shell scripts, batch files, and Python scripts specifically designed to retrieve and modify information about Managed Servers. The information is retrieved from and modified in the SA database.

Using the scripts, you can retrieve and modify such data as custom fields, customer assignments, custom attributes, and more. Given this ability, you can automate many procedures that in the past had to be accomplished on a server-by-server basis.

In addition, you can incorporate the information the scripts retrieve into customized scripts of your own design. Since information such as customer assignment and custom attributes varies from managed server to managed server, the ability to retrieve and use this information *on-the-fly* in customized scripts can be very useful.

For example:

- You may have a script that handles post-installation configuration for a certain application that must be able to discover the Facility name in which the server is registered. Agent Tools provides a script to get the Facility name and insert it into your post-installation script without manual intervention.
- When installing a monitoring agent, a post-installation script must modify a configuration file to include the IP address of the monitoring server in that particular facility. Agent Tools provides a script to discover the monitoring server's IP address by reading a custom attribute on the Core so that it can be inserted into the configuration file.
- A DSE can be written to retrieve the EEPROM version from many servers and store that information as a custom attribute or custom field.

Some other uses of Agent Tools scripts include:

- Gathering information from an SA Core during software installation for use in configuration.
- Storing metadata from managed servers in the SA database while executing a DSE, Global Shell script, or software installation.
- Retrieving custom attribute information for Managed Servers.

Following topics are discussed in this section:

- ["Installation requirements" on the next page](#)
- ["Installation" on the next page](#)

- ["Upgrading Agent Tools" on the next page](#)
- ["Agent Tools scripts" on page 126](#)
- ["Sample Agent Tool scripts" on page 128](#)

Installation requirements

The Agent Tools suite has the following requirements:

Operating System support

Agent Tools supports the operating systems supported by the SA Managed Servers. For a list of supported operating systems, See the Server Automation Install Guide.

Security, access control, and authentication

Agent Tools must be run as the *root user* on UNIX/Linux systems or as the *Administrator* on Windows systems. Agent Tools use the Server Agent's certificate to connect to the Web Services Data Access Engine (twist) which is pyTwist's default behavior, and is granted the privileges that the Web Services Data Access Engine gives to the Agent. This typically applies to read/write privileges on the server from which Agent Tools is run, therefore, no user authentication is required.

An exception is the `set_customer` script. You must have read access to a customer to be able to associate a server with that customer. Agent certificates do not have read access to other customers, therefore the user must authenticate when running this script.

Running Agent Tools scripts on Windows is not supported when UAC (User Access Control) is enabled.

Other requirements

- Access privileges to pyTwist
- Access privileges to the SA API
- Installed Python 2.4 (shipped with the Server Agent)

Installation

Agent Tools is installed in the Core during the normal HPE SA Installer Core installation process. However, you must also install Agent Tools on your Managed Servers to make it available on those

servers. This section describes that process.

Agent Tools is installed on Managed Servers as a set of executable scripts. Depending on your operating system, these will be shell or batch scripts and Python scripts which are called by the shell and batch scripts. You can run these scripts from a managed server to retrieve and modify information in the SA Core. These scripts can be run manually or called from package installation scripts, DSEs, Global Shell scripts, and so on.

Agent Tools is included as part of the Python SA API Access (pyTwist) software policy. This policy is located in the directory:

```
/Opware/Tools/Python Opware API Access
```

Manually installing Agent Tools

To install Agent Tools on a Managed Server:

1. Launch the SA Client.
2. Go to the **Managed Servers** list and select the Managed Server(s) on which you want to install Agent Tools.
3. Right click and select **Install Software**.
4. Select the **Python Opware API Access** software Policy.
5. The Software Policy installation wizard will guide you through the rest of the process.

Installing Agent Tools when installing an Agent

Alternatively, you can specify the Python SA API Access software Policy ID and specify that it be remediated during Agent installation. For information about Agent installation, see [Administer](#).

Upgrading Agent Tools

Since Agent Tools is provided as a software policy (part of the pyTwist software policy), you can upgrade to newer versions of Agent Tools by performing a remediation after upgrading the core.

When the SA core is upgraded, the Python SA API Access software policy is also updated; any old versions of Agent Tools are removed and new versions are attached to the policy. After the SA Core upgrade (during which Agent Tools will be automatically upgraded as part of the core upgrade), you can then upgrade Agent Tools on the Managed Servers by performing the following tasks:

1. Select the managed servers that have had Agent Tools installed. You can see a list of the servers and groups attached to the Python SA API Access software policy by opening the policy itself.

2. Right click on the selected servers and choose **Remediate**.
3. Select the **Python Opsware API Access** software policy.
4. The old versions of the pyTwist and Agent Tools packages are removed, and the new versions are installed.

Data migration

Since Agent Tools keeps no persistent data on the managed server, there's no requirement for data migration or preservation.

Agent Tools scripts

Usage

```
<scriptname>.py|bat|sh --arguments
```

Agent Tool scripts

Script	Function
get_all_cust_attr	<p>Retrieves all custom attributes for a server record.</p> <p>Usage: get_all_cust_attr.py [--localonly] [--mode=python shell pretty]</p> <p>The mode determines the format for the output (such as Python dictionary, shell statements, etc.). Pretty is the default.</p> <p>Note: Shell mode does not work when there are multi-line custom attributes.</p>
get_cust_attr	<p>Retrieves the value of a single custom attribute.</p> <p>Usage: get_cust_attr.py [--localonly] <custom attribute name></p>
set_cust_attr	<p>Sets the value of a single custom attribute on the server.</p> <p>Usage: set_cust_attr.py <custom attribute name> <custom attribute value> --valuefile <path to file with value in it></p>
del_cust_attr	<p>Deletes a custom attribute from the server's record in the database.</p> <p>Usage: del_cust_attr.py <custom attribute name></p>
get_	<p>Retrieves the value of a single custom field.</p>

Agent Tool scripts, continued

Script	Function
cust_field	Usage: get_cust_field.py <custom field name>
set_cust_field	Sets the value of a single custom field on the server. Usage: set_cust_field.py <custom field name> <custom field value> --valuefile <path to file with value in it>
get_customer	Retrieves the customer name that the server is associated with. Usage: ./get_customer.py
set_customer	Sets the customer name that the server is associated with. Usage: set_customer.py <customer name>
get_facility	Retrieves the name of the Facility that the server is associated with. Usage: ./get_facility.py
get_info	Prints out all fields for a server (in a format similar to the server's info file in OGSH). Usage: get_info.py
get_history	Prints out server specific events. Usage: get_history.py --startdate <start date in seconds since epoch> [--enddate <end date in seconds since epoch>] [--username <SAS user name>] [--password <SAS password>]
sub_text_file	Reads in a text file, looks in the file for tokens/parameters, replaces them with the value of custom attributes, and prints the amended file to stdout. See below for more info on the expected file format. Usage: sub_text_file.py [--localonly] <path to file with tokens in it>

Formatting for the sub_text_file script

Text files passed to the sub_text_file script can have any content, however, the script looks for any lines with two @ characters and will treat the string between and including the @ character pairs as a token. You can have a single @ character on a line, it will be ignored, however a second @ character on the same line will cause any text between the two @ characters to be treated as a token.

The tokens are replaced with the value of the custom attribute specified between the @ signs. For example, the string @dns_server@, is replaced with the value of the custom attribute dns_server. If this custom attribute does not exist or its value is empty, the token is replaced with an empty string.

Take a text file that contains the entry:

IP: @monitoring_server_ip@

The script will output will look similar to the following:

IP: 82.159.202.117

Where IP is the value retrieved by monitoring_server_ip.

Output

The sub_text_file script outputs to stdout. You can redirect the output to a file if needed. You can also use a .template file stored in your zip file to format the output. For example:

```
$AGENTTOOLSPATH/sub_text_file.sh petstore_config.template > petstore_config.cfg
```

Sample Agent Tool scripts

The following are simple examples of using Agent Tools scripts.

UNIX/Linux

This example puts a message containing the name of the facility in the Message of the Day (MOTD) that users see when they log into the UNIX server.

```
. /etc/opt/opsware/pytwist/pytwist.conf
facility_name=`$AGENTTOOLSPATH/get_facility.sh`
echo "You have connected to a server in the $facility_name facility. For hardware
information on this server as stored in Opsware, run $AGENTTOOLSPATH/get_info.sh."
> /etc/motd
```

Windows

This Windows example puts a text file on all users' desktops with information about the server.

```
call "C:\Program Files\Common Files\Opsware\etc\pytwist\
pytwist_conf.bat"

call "%AGENTTOOLSPATH%\get_info.bat" > "%SYSTEMDRIVE%\Documents and Settings\All
Users\Desktop\server_info_from_Opsware.txt"
```

1. Do not hard code the path to Agent Tools Instead you must do the following:

Source the PyTwist configuration file:

UNIX:

```
./etc/opt/opsware/pytwist/pytwist.conf
```


Windows:

```
call  
C:\Program Files\Common Files\Opware\etc\pytwist  
\pytwist_conf.bat
```

2. Use the environment variable:

UNIX:

```
$AGENTTOOLSPATH
```

Windows:

```
%AGENTTOOLSPATH%
```

Using this method will prevent errors in your scripts should the path to Agent Tools change in future.

Microsoft Windows PowerShell - SA integration

Windows PowerShell is an extensible command shell for system administrators and programmers, integrated with Microsoft's .Net 2.0 Framework Class Library. It uses the .NET common language runtime and the .NET Framework, and accepts and returns .NET objects. This enhances the tools and methods available to manage and configure of Windows.

Windows PowerShell provides numerous *cmdlets*, which are built into the shell and provide a wide range of functionality. Cmdlets can be used individually or in combination to perform more complex tasks.

Windows PowerShell not only enables access to a computer's file system, PowerShell *Providers* allow you to access data stores like the registry and digital signature certificate stores. A *Provider* is a software module that provides a uniform interface between a service and a data source.

Before you attempt to use the Windows PowerShell with SA, it is assumed that you are familiar with and comfortable using Microsoft Windows PowerShell. If you need background or instruction in using PowerShell, see <http://www.microsoft.com>.

Caution: Because the included cmdlets can modify data on your managed servers, it is important that you have a solid understanding of Windows PowerShell and its use.

Windows PowerShell integration with SA

SA provides initial integration with Microsoft Windows PowerShell on managed servers running Windows. PowerShell is available from SA user interfaces and SA data is available from within the standard PowerShell environment or from within any PowerShell Runspace. A *PowerShell Runspace* is a hosting environment for the PowerShell runtime system.

The following PowerShell cmdlets are available with SA:

- Get-SASServer
- Set-SASServer
- Get-SASJob

SA also includes a PowerShell *SAS Provider* (a component that provides access to the objects in an SA core in a PowerShell environment).

The following topics are discussed in this section:

- ["Windows PowerShell integration with SA" above](#)
- ["Integrated PowerShell/SA cmdlets" below](#)
- ["Installation requirements" on the next page](#)
- ["Installation" on the next page](#)
- ["Microsoft Windows PowerShell integration with SA features" on page 132](#)
- ["Sample sessions" on page 133](#)

Integrated PowerShell/SA cmdlets

The lists below and describes the integrated PowerShell/SA cmdlets included with SA.

PowerShell cmdlets

Cmdlet	Description	Arguments
Get-SASServer	Retrieves server data from specified server(s)	-Credential <PSCredential> -Core <Hostname IPAddress> -Name < ListOfHostnameFragments>

PowerShell cmdlets, continued

Cmdlet	Description	Arguments
		 -Id <ListOfServerIDs>
Get-SASJob	Retrieves data for specified jobs	-Credential <PSCredential> -Core <Hostname IPAddress> -JobFilter <ListOfJobIDs>
Set-SASServer	Retrieves a list of managed servers	-Credential <PSCredential> -Core <Hostname IPAddress> -Server <ServerV0>

Caution: If the target core is running a minimum protocol version of TLSv1.x, the Powershell version (the bound underlying .NET Framework version) must support it. For more information see, [https://msdn.microsoft.com/en-us/library/system.net.security.protocoltype\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.net.security.protocoltype(v=vs.110).aspx).

Installation requirements

An MSI installer package containing the cmdlets and PowerShell SA Provider assemblies, configuration and setup files for installation on a System Administrator's Windows desktop.

Operating System support

- Windows Server 2003
- Windows Server 2008
- Windows Server 2008 R2 x64
- Windows Server 2012

Installation

To implement Microsoft Windows PowerShell/SA integration, you must perform the following tasks:

- Locate the Microsoft Windows PowerShell/SA Connector MSI package in the OCC **Library>Software Policies**.

- Run the MSI to install the assemblies that define the SA-specific cmdlets and SA Provider. The file `readme.rtf` provides last minute information. The Microsoft Windows PowerShell initialization script, `profile.ps1` (similar to `.bashrc`) and a set of sample PowerShell scripts that show how to use PowerShell in an SA environment are also installed.

By default, the MSI installs the connector into `C:\Program Files\Opsware\PsSas`.

The file, `SAS-WSAPI.ps1`, describes accessing the WS-API directly from PowerShell, without the need for cmdlets.

Microsoft Windows PowerShell integration with SA features

Microsoft Windows PowerShell is available as an option in the following areas:

- ["Remote access to managed servers" below](#)
- ["Audit and snapshots rules" below](#)
- ["DSE script integration" on the next page](#)

Remote access to managed servers

From the SA Client, you can open a remote PowerShell session for any managed server (not available for a group of servers), as you would when opening a remote terminal.

1. Launch the SA Client.
2. From the Navigation pane, select **Devices>All Managed Servers**.
3. Select a Managed Server and open it.

In the Device Explorer window, from the **Actions** menu, select **Launch Remote PowerShell**.

You cannot run a script that contains *WMI calls* while logged in to a remote PowerShell session. If you try to run a script containing WMI call, you will get an `Access Denied` error, even if you are a member of a group with the necessary permissions to run that script.

Audit and snapshots rules

Microsoft PowerShell is integrated with SA auditing. While configuring a custom script rule, Microsoft PowerShell scripts are now an option along with batch, Python 2 and Visual Basic. For details about audit, see the Server Automation Administration Guide on the HPE SSO portal.

DSE script integration

For Managed Servers, you can set up PowerShell scripts that call SA APIs using Pytwist so that end users can invoke the scripts as DSEs or ISM controls. For more information about writing scripts that invoke Pytwist APIs, see ["Python API access with Pytwist" on page 66](#).

Sample sessions

This section provides four scenarios that demonstrate using Windows PowerShell/ SA integration.

- ["Scenario 1" below](#) demonstrates extracting managed server data from an SA Core, modifying it, and writing it back to the core.
- ["Scenario 2" on page 138](#) demonstrates exporting SA managed server data to an Excel spreadsheet using Windows PowerShell/SA integration.
- ["Scenario 3" on page 140](#) demonstrates mounting the SA core as a Windows PowerShell PSdrive and navigating around the virtual file system.
- ["Scenario 4" on page 144](#) demonstrates listing all the types of SA objects available to a Windows PowerShell environment.

Scenario 1

Authenticating to an SA Core, obtaining data about a managed server, modifying the data, and writing the data back to the SA Core.

1. Open a PowerShell prompt from the desktop icon.
2. Store the SA Core credentials securely in a PowerShell shell variable. See the following figure.

Storing the SA Credentials in a PowerShell variable



```
C:\Documents and Settings\paul\Desktop\powershell.exe
Windows PowerShell
Copyright (C) 2006 Microsoft Corporation. All rights reserved.

PS C:\documents and settings\Opsware> $creds = get-credential

cmdlet get-credential at command pipeline position 1
Supply values for the following parameters:
Credential
User: student35
Password for user student35: *****

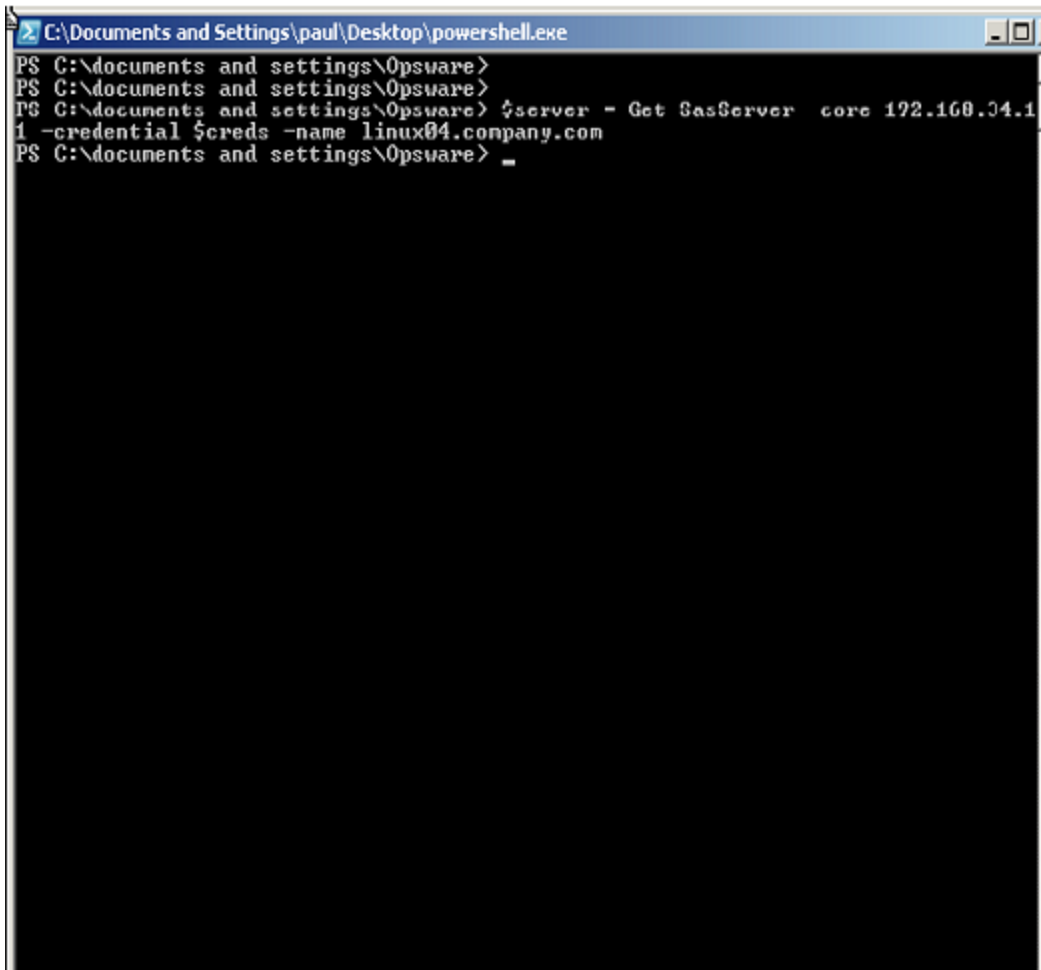
PS C:\documents and settings\Opsware> $creds

UserName                                     Password
-----
student35                                     System.Security.SecureString

PS C:\documents and settings\Opsware>
```

- Using the Get-SasServer cmdlet, you can retrieve the SA record representing a server as shown in the following figure.

Using the Get-SasServer cmdlet



```
C:\Documents and Settings\paul\Desktop\powershell.exe
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $server = Get-SasServer -core 192.168.34.1
1 -credential $creds -name linux04.company.com
PS C:\documents and settings\Opsware> _
```

The returned object is stored in a shell variable.

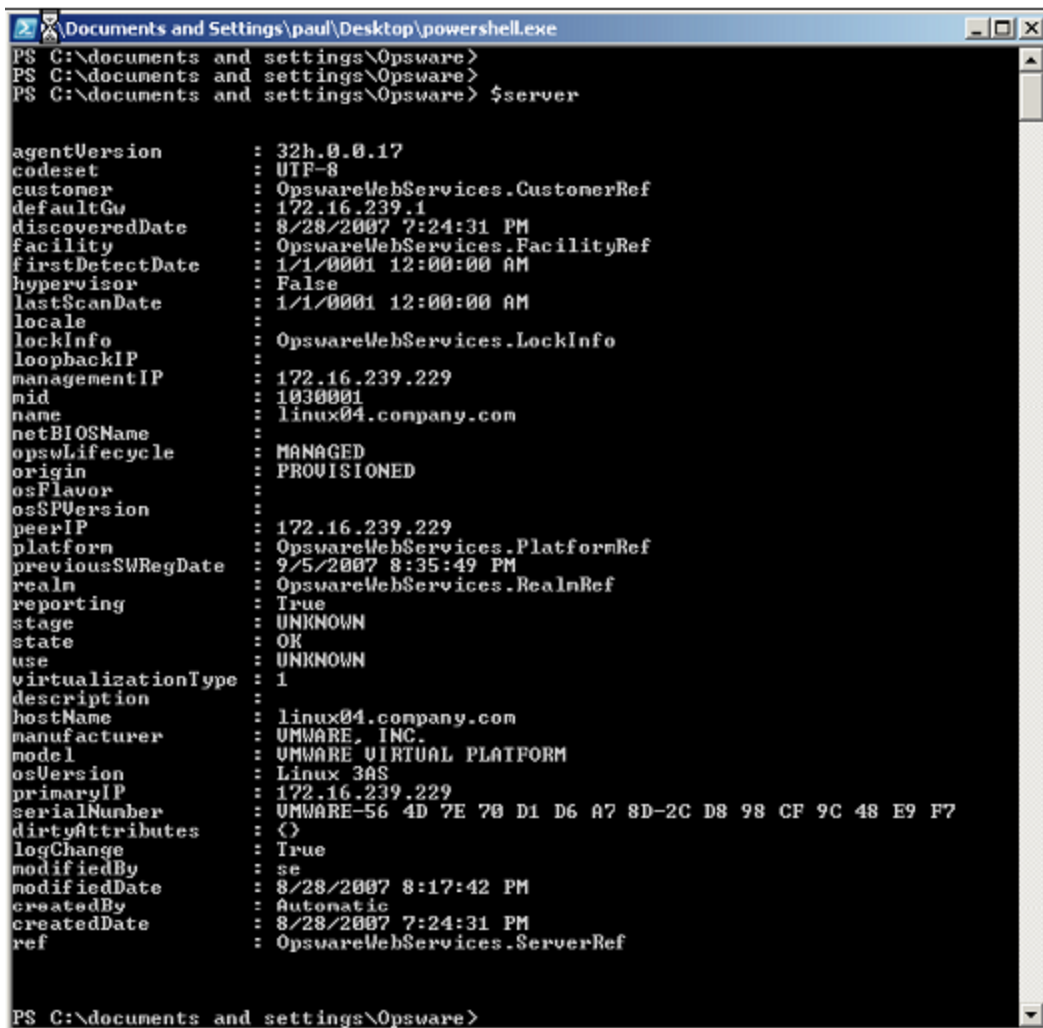
The `Get-SasServer` cmdlet takes a parameter to identify the Core from which the server data is to be retrieved, a parameter to supply credentials to the core for the operation, identifying and authenticating the user account in whose identity the operation is to be attempted, and a parameter to identify the server being requested.

More information on the `Get-SasServer` cmdlet arguments or the arguments for any cmdlet can be obtained by using the PowerShell `Get-Help` base cmdlet, for example:

```
Get-Help Get-SasServer -detailed
```

4. You can now examine the properties of the returned object by entering the name of the shell variable. See the following figure.

Examining SA Server properties



```
Documents and Settings\paul\Desktop\powershell.exe
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $server

agentVersion      : 32h.0.0.17
codeset           : UTF-8
customer          : OpswareWebServices.CustomerRef
defaultGw         : 172.16.239.1
discoveredDate    : 8/28/2007 7:24:31 PM
facility           : OpswareWebServices.FacilityRef
firstDetectDate   : 1/1/0001 12:00:00 AM
hypervisor        : False
lastScanDate      : 1/1/0001 12:00:00 AM
locale            :
lockInfo          : OpswareWebServices.LockInfo
loopbackIP       :
managementIP     : 172.16.239.229
mid               : 1030001
name              : linux04.company.com
netBIOSName      :
opsLifecycle      : MANAGED
origin            : PROVISIONED
osFlavor          :
osSPVersion       :
peerIP           : 172.16.239.229
platform          : OpswareWebServices.PlatformRef
previousSWRegDate : 9/5/2007 8:35:49 PM
realm             : OpswareWebServices.RealmRef
reporting         : True
stage             : UNKNOWN
state             : OK
use               : UNKNOWN
virtualizationType : 1
description       :
hostName          : linux04.company.com
manufacturer      : VMWARE, INC.
model             : VMWARE VIRTUAL PLATFORM
osVersion         : Linux 3AS
primaryIP         : 172.16.239.229
serialNumber      : VMWARE-56 4D 7E 70 D1 D6 A7 8D-2C D8 98 CF 9C 48 E9 F7
dirtyAttributes   : <>
logChange         : True
modifiedBy        : se
modifiedDate      : 8/28/2007 8:17:42 PM
createdBy         : Automatic
createdDate       : 8/28/2007 7:24:31 PM
ref               : OpswareWebServices.ServerRef

PS C:\documents and settings\Opsware>
```

5. List the object's properties, the types of the properties and the methods that can be called on the object from a PowerShell script as shown in following figure.

Listing an object's properties


```

C:\Documents and Settings\paul\Desktop\powershell.exe
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $server.gettype()

IsPublic IsSerial Name                                     BaseType
-----
True     False   ServerVO                                         OpswareWebService...

PS C:\documents and settings\Opsware> $server | Get-Member

TypeName: OpswareWebServices.ServerVO

Name           MemberType Definition
-----
Equals         Method      System.Boolean Equals(Object obj)
GetHashCode    Method      System.Int32 GetHashCode()
GetType        Method      System.Type GetType()
ToString       Method      System.String ToString()
agentVersion   Property    System.String agentVersion {get;set;}
codeset        Property    System.String codeset {get;set;}
createdBy      Property    System.String createdBy {get;set;}
createdDate    Property    System.DateTime createdDate {get;set;}
customer       Property    OpswareWebServices.CustomerRef customer {get...
defaultGw      Property    System.String defaultGw {get;set;}
description    Property    System.String description {get;set;}
dirtyAttributes Property    System.String[] dirtyAttributes {get;set;}
discoveredDate Property    System.DateTime discoveredDate {get;set;}
facility        Property    OpswareWebServices.FacilityRef facility {get...
firstDetectDate Property    System.DateTime firstDetectDate {get;set;}
hostName       Property    System.String hostName {get;set;}
hypervisor     Property    System.Boolean hypervisor {get;set;}
lastScanDate   Property    System.DateTime lastScanDate {get;set;}
locale         Property    System.String locale {get;set;}
lockInfo       Property    OpswareWebServices.LockInfo lockInfo {get;set;}
logChange      Property    System.Boolean logChange {get;set;}
loopbackIP     Property    System.String loopbackIP {get;set;}
managementIP   Property    System.String managementIP {get;set;}
manufacturer    Property    System.String manufacturer {get;set;}
mid            Property    System.String mid {get;set;}
model          Property    System.String model {get;set;}
modifiedBy     Property    System.String modifiedBy {get;set;}
modifiedDate   Property    System.DateTime modifiedDate {get;set;}
name           Property    System.String name {get;set;}
netBIOSName    Property    System.String netBIOSName {get;set;}
opsuLifecyle   Property    System.String opsuLifecyle {get;set;}
origin         Property    System.String origin {get;set;}
osFlavor       Property    System.String osFlavor {get;set;}
osSPVersion    Property    System.String osSPVersion {get;set;}
osVersion      Property    System.String osVersion {get;set;}
peerIP         Property    System.String peerIP {get;set;}
platform       Property    OpswareWebServices.PlatformRef platform {get...
previousSRegDate Property    System.DateTime previousSRegDate {get;set;}
primaryIP      Property    System.String primaryIP {get;set;}
realm          Property    OpswareWebServices.RealmRef realm {get;set;}
ref            Property    OpswareWebServices.ObjRef ref {get;set;}
reporting      Property    System.Boolean reporting {get;set;}
serialNumber   Property    System.String serialNumber {get;set;}
stage          Property    System.String stage {get;set;}
state          Property    System.String state {get;set;}
use            Property    System.String use {get;set;}
virtualizationType Property    System.Int64 virtualizationType {get;set;}
RunPSScriptBlock ScriptMethod System.Object RunPSScriptBlock();

PS C:\documents and settings\Opsware> _
  
```

- You can modify the object's **Description** attribute in Windows PowerShell, then call the Set-SasServer cmdlet and pass the modified ServerVO object to the cmdlet. This cmdlet will take the ServerVO object and update the managed server record in the SA Core. The Set-SasServer cmdlet takes parameters that identify the SA Core to which the updated data is to be written and credentials identifying the SA user account under whose identity the operation is executed.

At the end of the update operation, the updated ServerVO is returned to Windows PowerShell and the properties are displayed at the prompt as shown in the following figure.

Modifying an object's description

```
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $server.description = "Modified by student
35 from PowerShell"
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $server.dirtyAttributes = "description"
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $server | Set-SasServer -core 192.168.34.1
-credential $creds

agentVersion      : 32h.0.0.17
codeset           : UTF-8
customer          : OpswareWebServices.CustomerRef
defaultGw         : 172.16.239.1
discoveredDate    : 8/28/2007 7:24:31 PM
facility           : OpswareWebServices.FacilityRef
hypervisor        : False
locale            :
lockInfo          : OpswareWebServices.LockInfo
loopbackIP       :
managementIP     : 172.16.239.229
nid              : 1030001
name              : linux04.company.com
netBIOSName      :
opsWLifecycle     : MANAGED
origin            : PROVISIONED
osFlavor         :
osPVersion       :
peerIP           : 172.16.239.229
platform         : OpswareWebServices.PlatformRef
previousSWRegDate : 7/5/2007 8:35:49 PM
realm            : OpswareWebServices.RealmRef
reporting        : True
stage            : UNBNCVN
state            : OK
use              : UNBNCVN
virtualizationType : 1
description       : Modified by student35 from PowerShell
hostName         : linux04.company.com
manufacturer     : UMWARE, INC.
model            : UMWARE VIRTUAL PLATFORM
osVersion        : Linux 3AS
primaryIP        : 172.16.239.229
serialNumber     : UMWARE-56 4D 7E 70 D1 D6 A7 8D-2C D8 98 CF 9C 48 E9 F7
dirtyAttributes   : <>
logChange        : True
modifiedBy       : student35
modifiedDate     : 7/6/2007 2:00:56 PM
createdBy        : Automatic
createdAt        : 8/28/2007 7:24:31 PM
ref              : OpswareWebServices.ServerRef

PS C:\documents and settings\Opsware> _
```

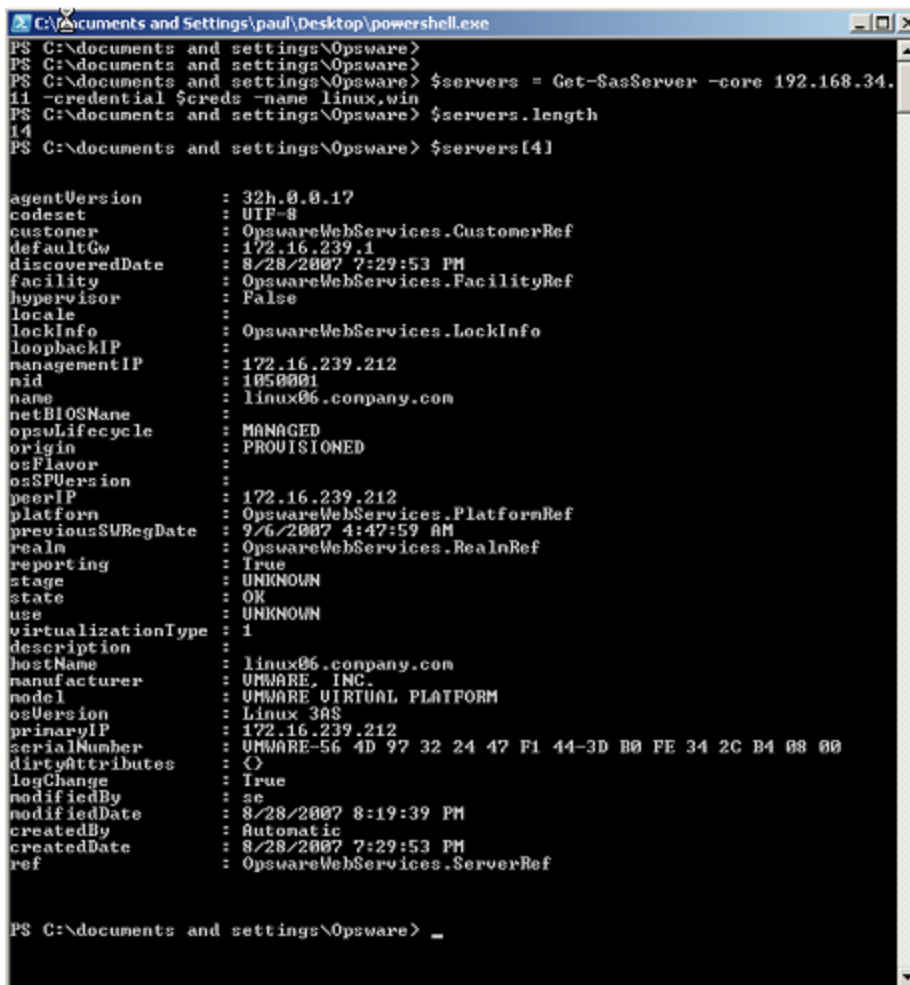
Scenario 2

This scenario demonstrates retrieving all managed server data from the SA Core and displaying it in Microsoft Excel.

1. Use the Get-SasServer cmdlet to retrieve ServerVOs for each Linux and Windows managed server from the SA Core. In the session below, the -name parameter is used to supply a list of name matching filters, for example, -name linux,win, to the SA Core.

The Get-SasServer cmdlet returns an array of ServerVOs that is, in this example, 14 items in length. You can index into this array to examine any one of the ServerVO objects. See the following figure.

Using the Get-SasServer cmdlet with a name filter



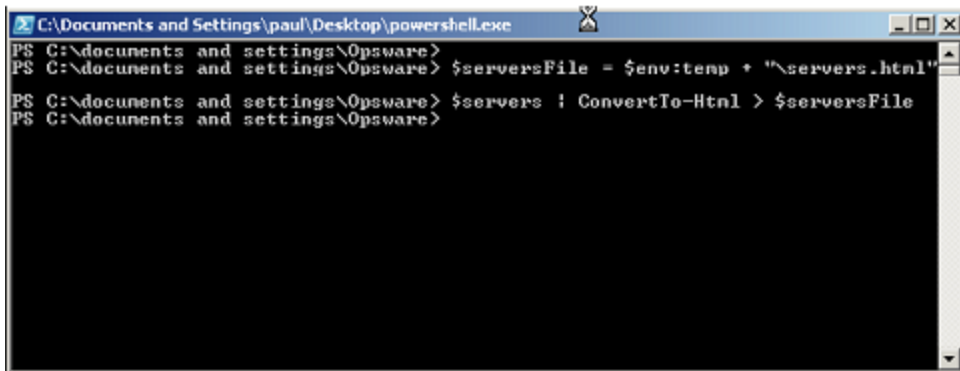
```
C:\Documents and Settings\paul\Desktop\powershell.exe
PS C:\documents and settings\Opware>
PS C:\documents and settings\Opware>
PS C:\documents and settings\Opware> $servers = Get-SasServer -core 192.168.34.
11 -credential $creds -name linux.win
PS C:\documents and settings\Opware> $servers.length
14
PS C:\documents and settings\Opware> $servers[4]

agentVersion      : 32h.0.0.17
codaset           : UTF-8
customer          : OpwareWebServices.CustomerRef
defaultGw         : 172.16.239.1
discoveredDate    : 8/28/2007 7:29:53 PM
facility           : OpwareWebServices.FacilityRef
hypervisor        : False
locale            :
lockInfo          : OpwareWebServices.LockInfo
loopbackIP        :
managementIP      : 172.16.239.212
nid               : 1050001
name              : linux06.company.com
netBIOSName       :
opaulLifecycle    : MANAGED
origin            : PROVISIONED
osFlavor          :
osSPUVersion      :
peerIP            : 172.16.239.212
platform          : OpwareWebServices.PlatformRef
previousSURegDate : 9/6/2007 4:47:59 AM
realm             : OpwareWebServices.RealmRef
reporting         : True
stage             : UNKNOWN
state             : OK
usage             : UNKNOWN
virtualizationType : 1
description       :
hostname          : linux06.company.com
manufacturer      : VMWARE, INC.
model             : VMWARE VIRTUAL PLATFORM
osVersion         : Linux 3AS
primaryIP         : 172.16.239.212
serialNumber      : VMWARE-56 4D 97 32 24 47 F1 44-3D B0 FE 34 2C B4 08 00
dirtyAttributes   : {}
logChange         : True
modifiedBy        : se
modifiedDate      : 8/28/2007 8:19:39 PM
createdBy         : Automatic
createdDate       : 8/28/2007 7:29:53 PM
ref               : OpwareWebServices.ServerRef

PS C:\documents and settings\Opware> _
```

2. Now you can format the ServerVO data as HTML and save to a temporary file. The temporary file is created in the TEMP directory. In a PowerShell session, to get the value of the %TEMP% environment variable, enter \$env:temp. See the following figure.

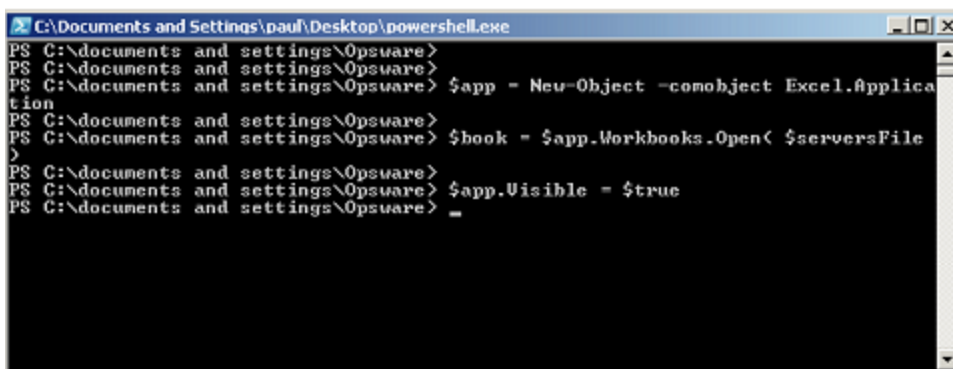
Converting ServerVO Data to HTML and saving to a temporary file



```
C:\Documents and Settings\paul\Desktop\powershell.exe
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $serversFile = $env:temp + "\servers.html"
PS C:\documents and settings\Opsware> $servers | ConvertTo-Html > $serversFile
PS C:\documents and settings\Opsware>
```

- Using the New-Object base Windows PowerShell cmdlet you can launch Microsoft Excel, then create a new workbook inside this instance of Excel, and populate the workbook from the contents of the temporary file. Finally, set the running Excel instance to be visible. This will cause Excel to come to the foreground. Now you can sort the data by date, column value, etc., to determine, for example, the date on which each server came under management in the Core. See the following figure.

Using the New-Object cmdlet to launch Microsoft Excel



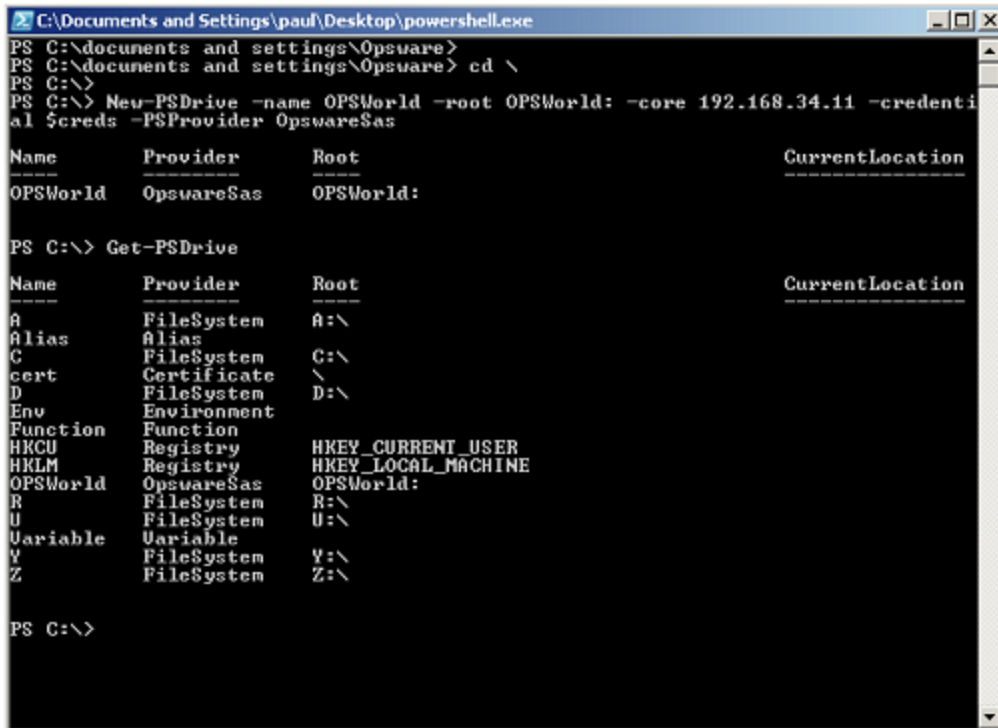
```
C:\Documents and Settings\paul\Desktop\powershell.exe
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $app = New-Object -comobject Excel.Application
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $book = $app.Workbooks.Open($serversFile)
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> $app.Visible = $true
PS C:\documents and settings\Opsware> _
```

Scenario 3

This scenario demonstrates mounting the SA Core as a Windows PowerShell PSDrive, navigating to the SA **Jobs** folder and retrieving its contents.

- Mount the SA Core as a Windows PowerShell PSDrive. PowerShell allows different data stores or repositories to be navigated as if they were a file system. In this scenario, you *mount* the SA Core, specifically the managed environment data store, as if it were a drive named OPSWorld. The windows PowerShell base system then calls the PowerShell SAS Provider, -PSProvider OpswareSas, whenever data is read from or written to this virtual file system — or when the file system is navigated by a client. See the following figure.

Mounting the SA Core as a Windows PowerShell PSDrive



```
C:\Documents and Settings\paul\Desktop\powershell.exe
PS C:\documents and settings\Opsware>
PS C:\documents and settings\Opsware> cd \
PS C:\>
PS C:\> New-PSDrive -name OPSWorld -root OPSWorld: -core 192.168.34.11 -credential $creds -PSProvider OpswareSas
```

Name	Provider	Root	CurrentLocation
OPSWorld	OpswareSas	OPSWorld:	

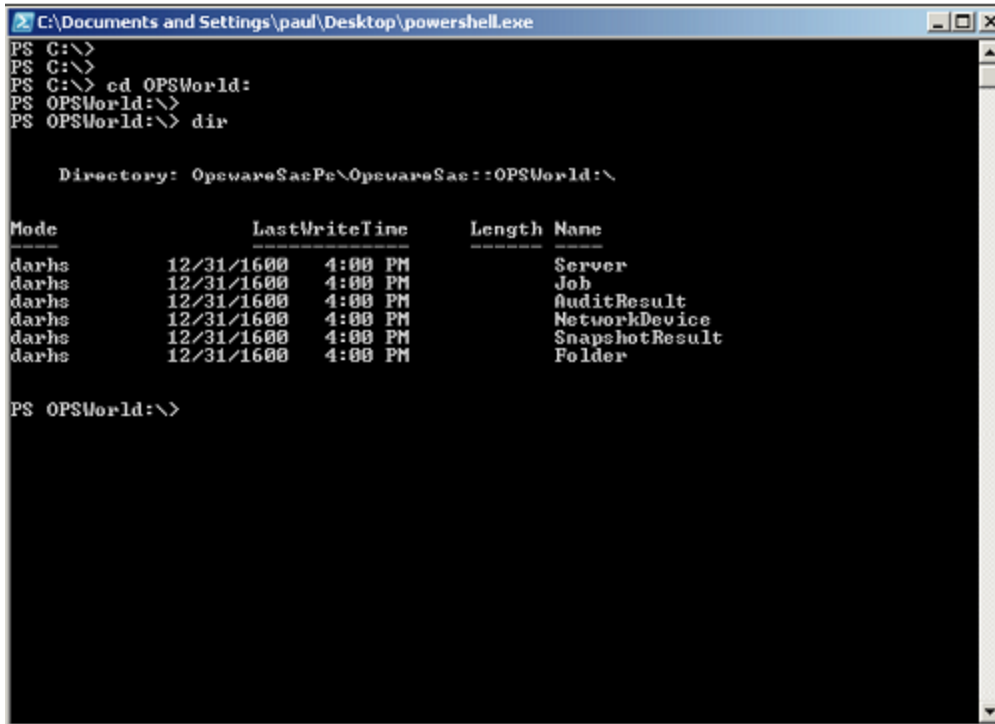
```
PS C:\> Get-PSDrive
```

Name	Provider	Root	CurrentLocation
A	FileSystem	A:\	
Alias	Alias		
C	FileSystem	C:\	
cert	Certificate	\	
D	FileSystem	D:\	
Env	Environment		
Function	Function		
HKCU	Registry	HKEY_CURRENT_USER	
HKLM	Registry	HKEY_LOCAL_MACHINE	
OPSWorld	OpswareSas	OPSWorld:	
R	FileSystem	R:\	
U	FileSystem	U:\	
Variable	Variable		
V	FileSystem	V:\	
Z	FileSystem	Z:\	

```
PS C:\>
```

2. Change directory to the newly mounted drive and obtain a directory listing. `dir` is a PowerShell alias for the `Get-ChildItem` cmdlet. See the following figure.

DIR as an alias for the Get-Child cmdlet



```
C:\Documents and Settings\paul\Desktop\powershell.exe
PS C:\>
PS C:\>
PS C:\> cd OPSWorld:
PS OPSWorld:\>
PS OPSWorld:\> dir

    Directory: OpwareSasPe\OpwareSas::OPSWorld:\

Mode                LastWriteTime         Length Name
----                -
darhs             12/31/1600      4:00 PM      Server
darhs             12/31/1600      4:00 PM        Job
darhs             12/31/1600      4:00 PM    AuditResult
darhs             12/31/1600      4:00 PM    NetworkDevice
darhs             12/31/1600      4:00 PM    SnapshotResult
darhs             12/31/1600      4:00 PM        Folder

PS OPSWorld:\>
```

3. Change directory to the **Jobs** folder, get a directory listing, and save the directory listing as a shell variable. This shell variable will contain an array of JobInfoVO objects from the Core into which you can index.

Save a Directory Listing as a PowerShell variable

```
C:\Documents and Settings\paul\Desktop\powershell.exe
PS OPSWorld:\>
PS OPSWorld:\>
PS OPSWorld:\> cd Job
PS OPSWorld:\Job>
PS OPSWorld:\Job> $jobs = dir
PS OPSWorld:\Job>
PS OPSWorld:\Job> $jobs.length
13
PS OPSWorld:\Job>
PS OPSWorld:\Job> $jobs[2]

PSPath           : OpswareSasPs\OpswareSas::OPSWorld:\Job
PSParentPath     : OpswareSasPs\OpswareSas::OPSWorld:
PSChildName      : Job
PSDrive          : OPSWorld
PSProvider       : OpswareSasPs\OpswareSas
PSIsContainer    : True
blockedReason    :
canceledReason   :
description      : May script: opsware.virtualization.scan_hypervisors
deviceGroups     : <>
endDate          : 8/29/2007 1:17:49 PM
notification     :
schedule         :
serverInfo       : <>
staleDate        : 1/1/0001 12:00:00 AM
startDate        : 8/29/2007 1:17:41 PM
status           : 6
type             :
userName        : $spin
userTag          :
ref              : OpswareWebServices.JobRef

PS OPSWorld:\Job> <$jobs[2]>.GetType()

IsPublic IsSerial Name                                     BaseType
-----
True     False   JobInfo00                                               OpswareWebService...
```

4. Change directory to the C: drive and remove the OPSWorld PSDrive.

Removing the OPSWorld PSDrive

```
C:\Documents and Settings\paul\Desktop\powershell.exe
PS OPSWorld:\>
PS OPSWorld:\>
PS OPSWorld:\> cd Job
PS OPSWorld:\Job>
PS OPSWorld:\Job> $jobs = dir
PS OPSWorld:\Job>
PS OPSWorld:\Job> $jobs.length
13
PS OPSWorld:\Job>
PS OPSWorld:\Job> $jobs[2]

PSPath           : OpswareSasPs\OpswareSas::OPSWorld:\Job
PSParentPath     : OpswareSasPs\OpswareSas::OPSWorld:
PSChildName      : Job
PSDrive          : OPSWorld
PSProvider       : OpswareSasPs\OpswareSas
PSIsContainer    : True
blockedReason    :
canceledReason   :
description      : May script: opsware.virtualization.scan_hypervisors
deviceGroups     : <>
endDate          : 8/29/2007 1:17:49 PM
notification     :
schedule         :
serverInfo       : <>
staleDate        : 1/1/0001 12:00:00 AM
startDate        : 8/29/2007 1:17:41 PM
status           : 6
type             :
userName         : $spin
userTag         :
ref              : OpswareWebServices.JobRef

PS OPSWorld:\Job> (<$jobs[2]).GetType()

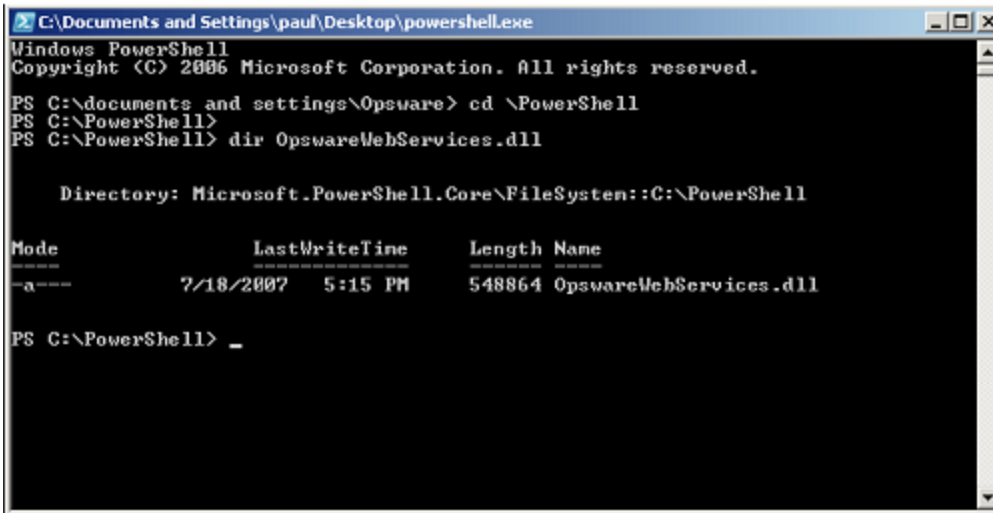
IsPublic IsSerial Name                                     BaseType
-----
True     False   JobInfoV0                                               OpswareWebService...
```

Scenario 4

This scenario describes examining all the types of SA objects available inside the Windows PowerShell environment.

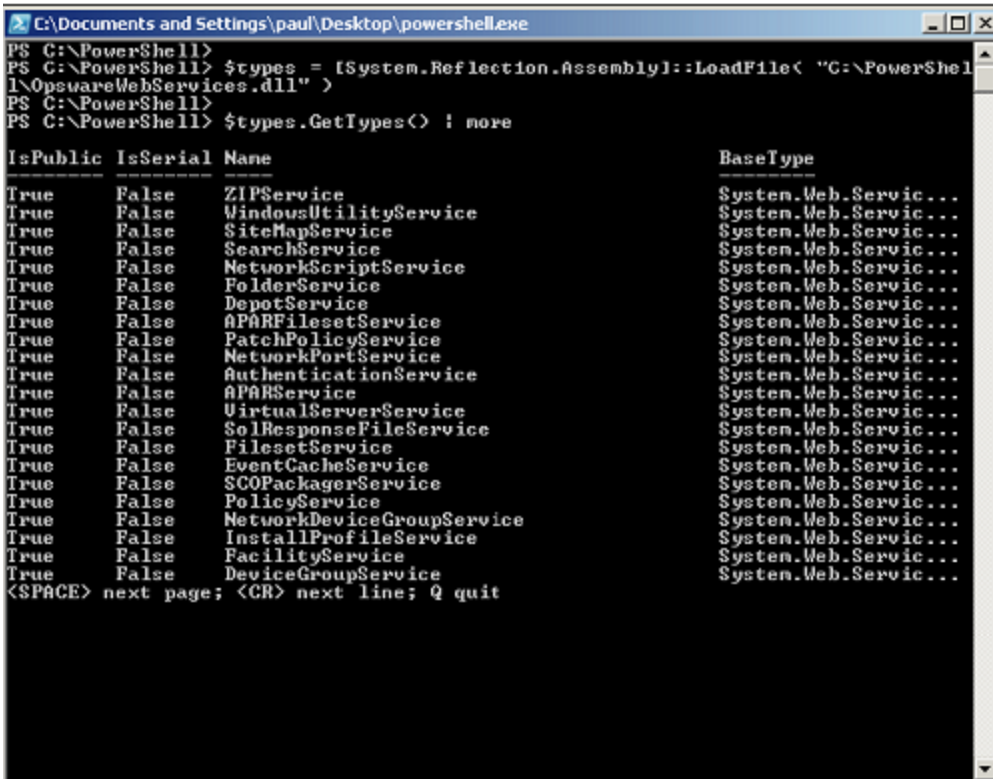
1. Locate the .NET assembly containing the PowerShell SAS Provider and cmdlets. See the following figure.

Locating the .NET Assembly containing the PowerShell SAS Provider and cmdlets



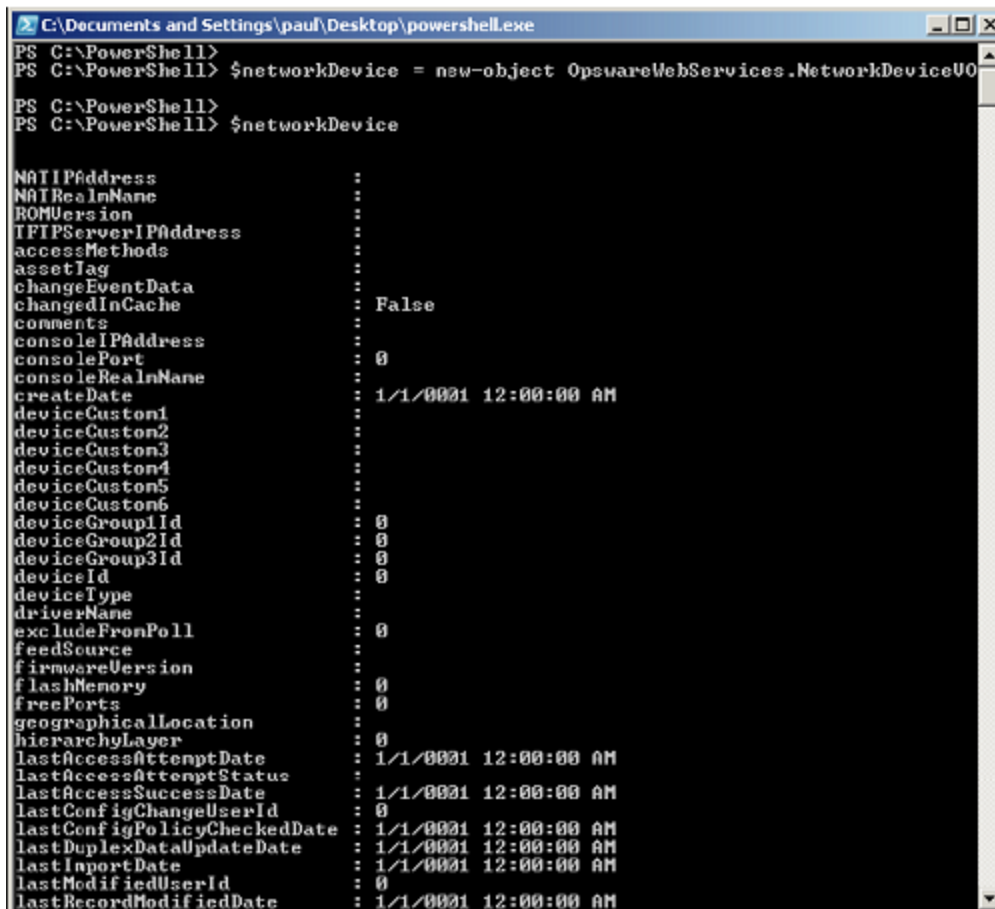
- Using .NET Reflection, load the .NET assembly and examine the loaded types. This displays all the SA types that are available for use in the Windows PowerShell environment. See the following figure.

Loading the .NET Assembly and examining the types



3. Create an instance of a NetworkDeviceVO. This is a nascent NetworkDeviceVO, showing all of the attributes of a network device available for scripting, reporting etc. in the PowerShell environment. See the following figure.

Creating an Instance of a NetworkDeviceVO



```
C:\Documents and Settings\paul\Desktop\powershell.exe
PS C:\PowerShell>
PS C:\PowerShell> $networkDevice = new-object OpswareWebServices.NetworkDeviceVO
PS C:\PowerShell>
PS C:\PowerShell> $networkDevice

NATIPAddress      :
NATRealmName     :
ROMVersion       :
TFTPServerIPAddr :
accessMethods    :
assetTag         :
changeEventData  :
changedInCache   : False
comments        :
consoleIPAddress :
consolePort      : 0
consoleRealmName :
createDate       : 1/1/0001 12:00:00 AM
deviceCustom1    :
deviceCustom2    :
deviceCustom3    :
deviceCustom4    :
deviceCustom5    :
deviceCustom6    :
deviceGroupId    : 0
deviceGroup2Id   : 0
deviceGroup3Id   : 0
deviceId         : 0
deviceType       :
driverName       :
excludeFromPoll  : 0
feedSource       :
firmwareVersion  :
flashMemory      : 0
freePorts        : 0
geographicalLocation :
hierarchyLayer   : 0
lastAccessAttemptDate : 1/1/0001 12:00:00 AM
lastAccessAttemptStatus :
lastAccessSuccessDate : 1/1/0001 12:00:00 AM
lastConfigChangeUserId : 0
lastConfigPolicyCheckedDate : 1/1/0001 12:00:00 AM
lastDuplexDataUpdateDate : 1/1/0001 12:00:00 AM
lastImportDate   : 1/1/0001 12:00:00 AM
lastModifiedUserId : 0
lastRecordModifiedDate : 1/1/0001 12:00:00 AM
```

Java RMI clients

A Java Remote Invocation (RMI) client can call the methods of the SA API from a server that has network access to the SA core. The server running the client does not have to be an SA core or managed server. When it connects to the core, the client specifies an SA user name and password, much like an end user logging on with the SA Client. The group that the user belongs to determines which SA resources and tasks are available to the client.

This topic is intended for software developers who are familiar with SA fundamentals and the Java programming language.

Setup for Java RMI clients

Before developing Java RMI clients for the SA API, perform the following steps:

1. Install an SA core in a development environment. Do not use a production core.
2. Obtain a development server where you will build and run the Java RMI client.
3. On the development server, install the Java SE 7 SDK.
4. Verify that the development server has a network connection to the SA core server that runs the OCC component.
5. Download the `opswclient.jar` file from the SA core server to your development server. The `opswclient.jar` file contains the Java RMI stubs for the SA API. You include the `opswclient.jar` in the `classpath` option when compiling and running Java RMI clients.
6. To download `opswclient.jar` do one of the following:
 - a. Specify the following URL, where `occ_host` is the core server running the OCC component:

```
https://occ_host/twister/opswclient.jar
```

- b. Go to the following directory: `/opt/opsware/twist/extlib/client`.

You also need the `spinclient-latest.jar` and the `opsware_common-latest.jar` files. These files can be obtained from a running SA Core in:

```
/opt/opsware/twist/lib/
```

You must also add these `.jar` files to the `classpath` parameter when compiling and running these examples.

Sample Java RMI

This section describes a simple Java RMI client named `GetServerInfo`.

The `GetServerInfo` client searches for managed servers by full or partial host name, which you specify as a command-line argument. For each managed server found, the client prints out the server's name, management IP address, and OS version.

The `GetServerInfo` client performs the following steps:

1. Connects to SA:

```
// Set the JNDI provider
client.setJNDIProvider( "https" , host , ( short ) 1032 , null, newString[] {
    OPSWARE_CA_CERT_PATH } , null ) ;
// Force a reconnection
client.getContext( true ) ;
// Set and authenticate the user
client.setAPIUser( new APIUserImpl(username , password)) ;
```

2. Gets a reference to the ServerService interface:

```
serverSvc = (ServerService)OpwareClient.getService
(ServerService.class);
```

3. Invokes methods on ServerService:

```
ServerRef[] serverRefs = serverSvc.findServerRefs(filter);
. . .
ServerVO[] serverVOs = serverSvc.getServerVOs(serverRefs);
. . .
System.out.println(serverVOs[i].getName());
```

Compiling and running the GetServerInfo example

Before compiling and running the example, perform the following tasks:

1. Obtain the `opsware_common-latest.jar`, `spinclient-latest.jar` and `opswclient.jar` files, as described in ["Setup for Java RMI clients" on the previous page](#).
2. Download the ZIP file that contains the demo program `GetServerInfo.java` file. For information about downloading the demo file, see ["Platform Developer Guide examples" on page 28](#).
3. To compile the client, specify the `opsware_common-latest.jar`, `spinclient-latest.jar` and `opswclient.jar` files for the `classpath` parameter:

```
javac -classpath :path/opswclient.jar:path/opsware_common-
latest.jar:path/spinclient-latest.jar GetServerInfo.java
```

4. To run the client, enter the following command, where *target* is the full or partial name of a server managed by SA. Note: the Java classpath separator for windows is `;`.

```
java -classpath .:path/opswclient.jar:path/opsware_common-
latest.jar:path/spinclient-latest.jar \GetServerInfo [options] target
```

In the following example, `GetServerInfo` connects to SA on host `c44` (where the OCC core component runs) and port 443. The program displays information for managed servers with

hostnames that contain the string opsw.

```
java -classpath ./home/jdoe/opswclient.jar:/home/jdoe/opsware_common-  
latest.jar:/home/jdoe/spinclient-latest.jar \GetServerInfo --host  
c44.dev.example.com --port 443 opsw
```

5. Respond to the prompts for the SA user name and password. The SA user must have read permissions for the servers that match the *target* specified on the command line.

Possible issue on Windows

The SA Java RMI Client, opswclient.jar, might not work on Windows if your SA certificates are using an SHA-224 signature algorithm. This is caused by the following JDK change: [Remove SHA224 from the default support list if SunMSCAPI enabled](#).

Workaround: Disable the SunMSCAPI security provider. This restores support for SHA-224 in the JDK. To disable the SunMSCAPI provider either:

- edit <JRE_HOME>/lib/security/java.security and comment out the line that defines the SunMSCAPI provider OR
- disable the SunMSCAPI provider programatically by using the `java.security.Security.removeProvider()` method.

Web Services clients

The SA API supports Web Services, a programming environment built on open industry standards such as SOAP (Simple Object Access Protocol) and WSDL (Web Services Definition Language). You can create Web Services clients in a variety of programming languages such as Perl and C# (as shown later in this section) or with Web Services-enabled development environments such as Microsoft Visual Studio .NET.

This topic is intended for software developers who are familiar with SA fundamentals and Web Services development.

- ["Programming language bindings provided in this release" on the next page](#)
- ["URLs for service locations and WSDLs" on the next page](#)
- ["Security for Web Services clients" on page 151](#)

- ["Overloaded operations" on the next page](#)
- ["Java interface support" on the next page](#)
- ["Unsupported data types" on the next page](#)
- ["Invoke setDirtyAttributes when creating or updating VOs" on page 152](#)
- ["Compatibility with SA Web Services API 2.2" on page 153](#)

Programming language bindings provided in this release

This release of SA includes Web Services client stubs for C#. Web Services clients written in Perl do not require client stubs.

This release does not include Web Services client stubs for Java or Python. However, Java clients can access the SA API through RMI and Python clients through Pytwist, as described in the preceding sections.

URLs for service locations and WSDLs

Clients access the Web Services at URLs with the following syntax, where *host* is the server running the OCC core component and *port* is for the HTTPS proxy. (The default proxy port is 443). The *packageName* corresponds to the Java library that the service belongs to.

```
https://host:port/osapi/packageName/WebServiceName
```

The WSDL files are at URLs with the following syntax:

```
https://host:port/osapi/packageName/WebServiceName?WSDL
```

For example, the following URLs point to the FolderService location and WSDL:

```
https://occ.c38.example.com:443/osapi/com/opsware/folder/FolderService
```

```
https://occ.c39.example.com:443/osapi/com/opsware/folder/FolderService?wsdl
```

The SOAP binding style is RPC (Remote Procedure Call) and the transport protocol is HTTPS.

Security for Web Services clients

Like other clients of the SA API, Web Services clients must be authenticated and authorized to perform operations in SA. Communication between clients and the Web Services component in the SA core is encrypted. Access is restricted to HTTPS clients through the HTTPS proxy port of the OCC core component. (The default port is 443.)

Overloaded operations

The SA API has overloaded operations, but the WSDL 2.0 specifications do not support overloading. An overloaded operation in the SA API is exposed by the Web Service as a single operation.

Java interface support

The SA API uses Java interfaces, but Web Services does not support interfaces. As a workaround, the WSDL files map interfaces to `xsd:anyType`. For clients coded in object-oriented programming languages such as C#, if an API method returns an interface, the return type must be cast to a concrete class. Arrays of interfaces are converted to `Object[]`; specific types of the array members are preserved through serialization/deserialization. For a C# code example, see ["Handle interface return types" on page 164](#).

Unsupported data types

The following data types are used by the SA API but are not supported by SOAP:

```
java.util.Properties  
com.opsware.common.ModifiableMap  
com.opsware.acm.ValueSet  
com.opsware.swmgmt.PolicyOverrideFilter
```

Methods omitted from Web Services

The following SA API methods use unsupported data types as parameters or return types. As a result, they are not exposed as operations in the Web Services.

```
com.opsware.custattr.CustomAttribute.getCustAttrs  
com.opsware.custattr.CustomAttribute.setCustAttrs  
com.opsware.custattr.CustomField.getCustomFields  
com.opsware.custattr.CustomField.setCustomFields  
com.opsware.pkg.Patch.getPolicyOverrideRefs
```

Partial support for `java.util.Map`

Axis converts `java.util.Map` to `apachesoap:Map`, which is a collection of key-value pairs. With .NET, this conversion does not work. C# clients, for example, will receive an empty array of key-value pairs. However, this conversion does work with `Soap::Lite` in Perl. Therefore, SA API methods that use `java.util.Map` are available as operations in the Web Services.

The following methods use `java.util.Map` as parameters or return types:

```
com.opsware.acm.GroupConfigurable.getApplicationInstances  
com.opsware.acm.ServerConfigurable.getCustAttrsWithRC  
com.opsware.compliance.sco.CMLSnapshott.getValueSet  
com.opsware.compliance.sco.CMLSnapshott.setValueSet  
com.opsware.compliance.sco.SnapshottResultService.remediateCMLSnapshott  
com.opsware.custattr.VirtualColumnVO.getConfigInfo  
com.opsware.custattr.VirtualColumnVO.setConfigInfo
```

Methods in VOs with unsupported data types

The following methods of VOs use unsupported data types as parameters or return types:

```
com.opsware.acm.ApplicationInstanceVO.getValueSet  
com.opsware.acm.ApplicationInstanceVO.setValueSet  
com.opsware.acm.ConfigurableVO.getValueSet  
com.opsware.acm.ConfigurableVO.setValueSet  
com.opsware.virtualization.VirtualConfigNode.getProperties  
com.opsware.virtualization.VirtualConfigNode.setProperties  
com.opsware.virtualization.VirtualServerConfig.getProperties  
com.opsware.virtualization.VirtualServerConfig.setProperties
```

Invoke `setDirtyAttributes` when creating or updating VOs

Web Services clients must invoke `setDirtyAttributes` before invoking a create or update method on a service. The `setDirtyAttributes` method explicitly marks the attributes (fields) of a VO that

need to be set by the create or update invocation. The attribute names specified by `setDirtyAttributes` are case sensitive.

For example, to modify the `description` attribute of a `FolderVO` object, the following code invokes `setDirtyAttributes` before it invokes `update`:

```
// fs is FolderService
FolderVO folderVO = fs.getFolderVO(folderRef);
folderVO.setDescription("credit card processing");
folderVO.setDirtyAttributes(new String[]{"description"});
fs.update(folderRef, folderVO, true, true);
```

Invoking `setDirtyAttributes` is required for Web Services clients because of the way Axis deserializes XML objects from XML. If `setDirtyAttributes` is not invoked, Axis calls setters on all attributes of the VO, including read-only attributes, resulting in a `ReadOnlyException`.

Compatibility with SA Web Services API 2.2

The SA Web Services API 2.2 is not compatible with the SA API described in this guide. The method signatures, services, WSDLs, and port bindings are not the same. If you are creating new Web Services clients, be sure to use the SA API, not the SA Web Services API 2.2.

Perl Web Services clients

This section contains step-by-step instructions and sample code for creating Perl Web Services clients that access the SA API.

- ["Required software for Perl clients" below](#)
- ["Running the Perl demo program" on the next page](#)
- ["Sample Perl code" on page 155](#)
- ["Construction of Perl objects for Web Services" on page 158](#)

Required software for Perl clients

Your development environment must have the following Perl modules:

- Crypt-SSLeay-0.57
- IO-Socket-SSL-1.31
- Net-SSLeay-1.35
- HTML-Parser-3.64
- MIME-Base64-3.08
- URI-1.40
- libwww-perl-5.833
- SOAP-Lite-0.710

Depending on your Perl version, newer versions of these modules could be required.

Caution: If the "500 SSL negotiation failed:" error persists, then OpenSSL needs to be updated to version 1.0.1 or higher. For the RHEL family, OpenSSL needs to be updated to version 1.0.1e-30 or higher.

Running the Perl demo program

To run the demo program, perform the following steps:

1. From the support site, obtain the **SA_Platform_Developer_Guide_examples.zip** file bundled with the **All Manuals Download SA 10.5** folder.
2. Obtain the [demo program](#) `uapisample.pl` file from `SA_Platform_Developer_Guide_examples.zip\SA_Platform_Developer_Guide_examples\api_examples\web_services\perl`.
3. Edit the `uapisample.pl` file, changing the hard-coded values for host, username, password, and object IDs such as `serverID`.
4. Run `uapisample.pl`.
5. If you receive a "Certificate Verify Failed" error, you should uncomment the following line from the sample file and provide a valid path to the certificate file:

```
#$ENV{HTTPS_CA_FILE} = "path_to/opsware-ca.crt";
```

You can find the certificate file from an SA Core in:

```
/var/opt/opsware/crypto/twist/opsware-ca.crt
```

Sample Perl code

The following code snippets are from `uapisample.pl`, a Perl program contained in the [ZIP file](#) you downloaded previously.

Set up the Service URI

```
# Construct the URI for the service.
#
my $username = "integration";
my $password = "integration";
my $protocol = "https";
my $host = "occ.c38.dev.example.com";
my $port = "443";
my $contextUri = "osapi/com/opsware/";
my $folderServiceName = "folder/FolderService";
my $folderUri = "http://www.example.com/" . $contextUri .
$folderServiceName;
# Create a proxy to the FolderService.
#
my $folderProxy = $protocol . "://" . $username . ":" . $password . "@" .
$host . ":" . $port . "/" . $contextUri . $folderServiceName;
```

Initiate a new service

```
my $folderPort = SOAP::Lite
-> uri($folderUri)
-> proxy($folderProxy);
```

Invoke a service method

```
my $root = $folderPort->getRoot()->result();
print 'Got root folder: ' . $root->{'name'} . "\n";
# Alternative:
my $root = $folderPort->SOAP::getRoot();
print 'Got root folder: ' . $root->{'name'} . "\n";
```

Get a VO

```
$rootVO = $folderPort->getFolderVO(SOAP::Data->name('self')
->value(\SOAP::Data->name('id')->type('long')->value(0)))
->result();
# The preceding call to getFolderVO does not pass a FolderRef
# parameter. If a method such as FolderService.remove accepts a
# FolderRef parameter, use the following code:
#
my $folderToBeRemoved = SOAP::Data->name('self')
->attr({'xmlns:ns_fs' => 'http://folder.example.com/FolderService'}) -
```

```
>type('ns_fs:FolderRef')->value(\SOAP::Data->name('id')->type('long') ->value(123456));
$folderPort->remove($folderToBeRemoved);
# To see the Perl representation of the returned VO, you can use
# the Dumper method. This will help you understand how to
# construct the dirty attributes of a VO for a create or update
# method.
#
use Data::Dumper;
print Dumper($folderVO);
```

Get an array

```
# Construct $folder, the FolderRef before getting the array.
#
my $folder = SOAP::Data->name('self') ->attr({ 'xmlns:ns_fs' => 'http://
folder.example.com'}) ->type('ns_fs:FolderRef')->value(\SOAP::Data-
>name('id')->type('long') ->value($root->{'id'}));
# The getChildren method returns an array of FNodeReference
# objects.
#
my $children = $folderPort->getChildren($folder, SOAP::Data->name('type')-
>type('string')->value(''))->result();
foreach $child (@{$children}){
print 'Get child: ' . $child->{'name'} . "\n";
}
```

Construct an object array

```
# For a function that takes an object array as a parameter,
# such the getVOs method, take the following approach:
# First, construct the Array object elements individually
# and put them in an array.
#
my @refs = [];
foreach my $ref (@{$myRefs}){
    # Assume myRefs was returned from a previous
    # Web Services call.
    my $object = SOAP::Data->name('FacilityRef')
        ->value(\SOAP::Data->name('id')
            ->type('long')
            ->value($ref->{'id'}
        )
        )->attr({ 'xmlns:facility' => 'http://locality.example.com'})
        ->type('facility:FacilityRef');
    push @refs, $object;
}
# Second, construct an Array Object and put the array in it.
#
my $selves = SOAP::Data->name("selves" =>\SOAP::Data->name("element" => @refs)-
```

```
>type("facility:FacilityRef"))
    ->attr({ 'xmlns:facility' => 'http://locality.example.com'})
    ->type("facility:ArrayOfFacilityRef");
```

Update or create a VO

```
# This example updates the description attribute of a ServerVO.
#
my $serverID = 40038;
my $server = SOAP::Data->name('self')->value(\SOAP::Data->name('id')->type('long')->value($serverID));
# Don't forget to set dirtyAttributes for the attributes
# you want to update. You also need dirtyAttributes for
# create methods that pass a VO.
#
my @dirtyAttrs = ('description');
my $serverVO = SOAP::Data->name('vo') ->attr({ 'xmlns:ns_ss' => 'http://server.example.com'}) ->value(\SOAP::Data->value( SOAP::Data->name('description')->value('PERL_UPDATE_DESC')->type('string'), SOAP::Data->name('logChange')->value('false')->type('boolean'), SOAP::Data->name('dirtyAttributes' => \SOAP::Data->name("element" => @dirtyAttrs)->type("string")) ->type("ns_ss:ArrayOf_soapenc_string"), ));
my $force = SOAP::Data->name('force')->value('true')->type('boolean');
my $refetch = SOAP::Data->name('refetch')->value('true')->type('boolean');
# Call the update method.
#
print 'Invoking method serverWSPort.update...', "\n";
my $updatedServerVO = $serverWSPort->update(
    $server,
    $serverVO,
    $force,
    $refetch)->result();
print "New description: ", $updatedServerVO->{'description'}, "\n";
```

Handle SOAP faults

```
# Make sure that you turn off on_fault subroutine in the
# "use SOAP::Lite ..." statement.
#
# The fault member of a SOAP return will be set if the Web
# Service call throws an exception.
# The following code tries to get a folder that does not exist:
#
my $testVO = $folderPort->getFolderVO(SOAP::Data->name('self') ->value(\SOAP::Data->name('id')->type('long')->value(123456)));
if($testVO->fault){
    print $testVO->faultstring . "\n";
    # This will print the error msg.
    print "ExceptionName: " . getExceptionName($testVO) . "\n"; # A
    NotFoundException should be displayed here
}
```

```
        # The code that deals with the error goes here....
    }
    . . .
# The following subroutine extracts the exception name from the
# returned faultdetail.
#
sub getExceptionName {
    my $fault = shift; #get the fault object
    if($fault->faultdetail->{'fault'}){
        return ref($fault->faultdetail->{'fault'});
    }
}
. . .
# As shown in the preceding code, it's easier to handle SOAP
# faults if you execute functions like this:
#
# my $data = $port->function(...);
# Not like this:
# $port->SOAP::function(...);
# $port->function(...)->result;
```

Construction of Perl objects for Web Services

Before calling a Web Services operation, a Perl client must set up the data structures that are required for the input parameters. The information you need for setting up the data structures is in the API documentation (javadocs) and the service's WSDL file. The Perl code example in this section shows how to construct the input parameter for the `getServerVO` operation. The step-by-step instructions after the code show where to get the information about the input parameter from the API documentation and the WSDL file.

Source code for calling `getServerVO`

The following Perl code sets up the input parameter `self` and then calls the `getServerVO` operation. This call retrieves the VO (value object) for the managed server of ID 12345.

```
# Create a top-level SOAP::Data object
#
$self = SOAP::Data->name('self')
# The namespace corresponds to the schema of the data type
# of the SOAP::Data object. The name chosen (ns_ss) is
# arbitrary.
#
$self->attr({'xmlns:ns_ss =>
'http://server.example.com/ServerService'});
# Specify the type (ServerRef) for the parameter self, using the
# name of the namespace from the preceding statement.
```

```
#
$self->type('ns_ss:ServerRef');
# Create the value for the parameter. The value is a pointer
# to a SOAP::Data object. The number 12345 is the SA ID of a managed server.
#
my $id = SOAP::Data->name('id')->type('long')->value(12345);
# From the self object, point to the value.
#
$self->value(\$id);
# Finally, call getServerVO:
#
my $data = $serverPort->getServerVO($self);
if($data->fault){
    # Handle exceptions here ...
}
else{
    my $serverVO = $data->result;
}
. . .
```

Location of information for getServerVO setup

To get the information needed to write the code for the call to `getServerVO`, perform the following steps:

1. In a browser, go to the API documentation (javadocs) at the following URL:

```
https://occ_host:1032/twister/docs/index.html
```

The `occ_host` is the IP address or host name of the core server running the Command Center component. (For instructions on invoking methods with the Twister, see ["API Documentation and the Twister" on page 27.](#))

2. Examine the API documentation to determine the input parameters and return value of the method.

The `getServerVO` method is defined in the interface `com.opsware.server.ServerService`. In the following method signature, note that `getServerVO` accepts a `ServerRef` as a parameter and returns a `ServerVO`:

```
public ServerVO getServerVO(ServerRef self)
    throws java.rmi.RemoteException,
           NotFoundException,
           AuthorizationException
```

3. In a browser, specify the following URL to open the WSDL file for the `ServerService`:

`https://occ_host/osapi/com/opsware/server/ServerService?wsdl`

4. In the WSDL file, locate the namespace for the ServerService:

```
<schema targetNamespace="http://server.example.com"
xmlns="http://www.w3.org/2001/XMLSchema">
```

The following Perl statement (from the code listed previously) specifies the namespace:

```
$self->attr({'xmlns:ns_ss =>
'http://server.example.com/ServerService'});
```

5. In the WSDL file, locate the `getServerVO` operation and note the input message name `getServerVORequest`.

```
<wsdl:operation name="getServerVO" parameterOrder="self">
  <wsdl:input message="impl:getServerVORequest" name="getServerVORequest"/>
  <wsdl:output message="impl:getServerVOResponse" name="getServerVOResponse"/
>
  <wsdl:fault message="impl:NotFoundException" name="NotFoundException"/>
  <wsdl:fault message="impl:AuthorizationException"
name="AuthorizationException"/>
</wsdl:operation>
```

6. In the WSDL file, locate the `getServerVORequest` message:

```
<wsdl:message name="getServerVORequest">
  <wsdl:part name="self" type="impl:ServerRef"/>
</wsdl:message>
```

The `getServerVORequest` message element defines the name (`self`) and type (`ServerRef`) of the input parameter of `getServerVO`. The following Perl statement specifies `ServerRef`:

```
$self->type('ns_ss:ServerRef');
```

7. In the WSDL file, locate the `complexType` for `ServerRef`:

```
<complexType name="ServerRef">
  <complexContent>
    <extension base="tns1:ObjRef">
      <sequence>
        <element name="secureResourceTypeName" nillable="true"
type="soapenc:string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Note that `ServerRef` extends `ObjRef`.

8. In the WSDL file, locate the complexType for ObjRef:

```
<complexType abstract="true" name="ObjRef">
  <sequence>
    <element name="id" type="xsd:long"/>
    <element name="idAsLong" nillable="true" type="soapenc:long"/>
    <element name="name" nillable="true" type="soapenc:string"/>
  </sequence>
</complexType>
```

In ObjRef, note the name (`id`) and type (`long`). These data types are specified in the following Perl statement:

```
my $id = SOAP::Data->name('id')->type('long')->value(12345);
```

C# Web Services clients

This section contains step-by-step instructions and sample code for creating C# Web Services clients that access the SA API.

- ["Required software for C# clients" below](#)
- ["Obtaining the C# client stubs" on the next page](#)
- ["Building the C# demo program" on the next page](#)
- ["Running the C# demo program" on page 163](#)
- ["Sample C# code" on page 164](#)
- ["Password security with C#" on page 166](#)

Required software for C# clients

To develop C# Web Services clients, your development environment must have the following software:

- Microsoft .NET Framework SDK version 1.1
- C# client stubs for SA API

Obtaining the C# client stubs

SA provides a stub file for each service, for example, `FolderService.cs`. All stubs have the same namespace: `OpwareWebServices`. In addition to the stubs, SA provides `shared.cs`, the file that contains shared classes such as `ServerRef`.

To obtain a ZIP file containing the C# stubs, specify the following URL, where `occ_host` is the core server running the OCC component:

```
https://occ_host:1032/twister/opswcsharpclient.zip
```

The constants defined in services and objects are not defined in the C# stubs. To get information about the constants, use the API documentation (javadocs), as described in ["Constant field values" on page 27](#).

Building the C# demo program

To build the demo program:

1. From the support site, obtain the **SA_Platform_Developer_Guide_examples.zip** file bundled with the **All Manuals Download SA 10.5** folder. For information about downloading the demo file see, ["Platform Developer Guide examples" on page 28](#).

The **SA_Platform_Developer_Guide_examples.zip** contains the following demo program files at the `SA_Platform_Developer_Guide_examples\api_examples\web_services\csharp` location:

- `App.config` - Application settings
 - `WebServicesDemo.cs` - Client code that invokes service methods
 - `MyCertificateValidation.cs` - Certificate validation class
2. Create the following directory:
`C:\wsapi`
 3. From the Visual Studio 2008 Start Page, select New Project and create a project with the following values:
 - Project Type: Visual C# Projects
 - Template: Console Application

- Name: WSAPIDemo
- Location: C:\wsapi

This action creates the new directory C:\wsapi\WSAPIDemo, which contains some files.

4. In the new project, delete the default program and AssemblyInfo.cs from the list of objects.
5. Copy the files you obtained in step 1 into the C:\wsapi\WSAPIDemo directory.
6. Download the client stubs from the URL specified in "[Obtaining the C# client stubs](#)" on the [previous page](#).
7. Copy the C# client stubs into the C:\wsapi\WSAPIDemo directory.
8. Add the files copied in the preceding two steps to the WSAPIDemo project:
 - In Visual Studio, from the Project menu, select Add Existing Item.
 - Browse to the directory C:\wsapi\WSAPIDemo, and select all the demo files (.cs and .config).
9. Add a reference to System.Web.Services.dll:
 - In Visual Studio, from the Project menu, select Add Reference.
 - Under the .NET tag, browse to Component with Name: System.Web.Services.dll.
 - Click System.Web.Services.dll, click Select, and then click OK.
10. If you used a different template when creating the project, you might need to add references to System, System.XML, and System.Data. Check the Project References to determine if you need to add these references.
11. In the App.config file, change the values for username, password, host, and the hardcoded object IDs such as serverID.
12. In Visual Studio, from the Build menu, select Build WSAPIDemo.

Caution: If the target core is running a minimum protocol version of TLSv1.x, the Powershell version (the bound underlying .NET Framework version) must support it. For more information see, [https://msdn.microsoft.com/en-us/library/system.net.securityprotocoltype\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.net.securityprotocoltype(v=vs.110).aspx)

Also, it must be explicitly enabled from the C# application.

```
Code sample: System.Net.ServicePointManager.SecurityProtocol |=  
System.Net.SecurityProtocolType.Tls12 | System.Net.SecurityProtocolType.Tls11;"
```

Running the C# demo program

To run the demo program:

1. Open the Visual Studio 2008 command prompt:

```
Start > All Programs > Microsoft Visual Studio 2008 >  
Visual Studio Tools > Visual Studio 2008 Command Prompt
```

2. Change the directory to:

```
C:\wsapi\WSAPIDemo\bin\Debug
```

3. Enter the following command:

```
WSAPIDemo.exe
```

Sample C# code

The following code snippets are from `WebServicesDemo.cs`, a C# program contained in the [Zip file](#) you downloaded previously.

Set up certificate handling

```
# This setup is required just once for the client.  
#  
ServicePointManager.CertificatePolicy = new MyCertificateValidation();
```

Assign the URL prefix

```
# This is the URL prefix for all services.  
#  
wsdlUrlPrefix = protocol + "://" + host + ":" + port + "/" + contextUri + "/";
```

Initiate the service

```
FolderService fs = new FolderService();  
fs.Url = wsdlUrlPrefix + "com.opsware.folder/FolderService";
```

Invoke service methods

```
FolderRef root = fs.getRoot();  
FolderVO vo = fs.getFolderVO(root);
```

Handle interface return types

```
    # In the API, FolderVO.getMembers returns an array of  
    # FNodeReference interfaces, but Web Services does not support  
    # interfaces. In the C# stub, the return type of
```

```
# FolderVO.members is Object[]. If a returned Object type will
# be used as a parameter that must be a specific type, then you
# must cast it to that type. For example, the following code
# casts elements of the returned array to FolderRef as
# appropriate.
#
Object[] members = vo.members;
for(int i=0;i<members.Length;i++)
{
Console.WriteLine("Got object: " + members[i].GetType().FullName + " --> " +
((ObjRef)members[i]).name);
    if(members[i] is FolderRef) {
        Console.WriteLine("I am a FolderRef: " +
            ((FolderRef)members[i]).name);
    }
}
}
```

Update or create a VO

```
    # When updating a VO, the changed attributes must be set in
# dirtyAttributes. (The VO passed to a create method has
# the same requirement.)
#
# Note: If you update a VO that was returned from a service
# method invocation, such as getFolderVO, then you must
# set the logChange attribute of the VO to false:
# vo.logChange = false;
#
# The following code changes the name of a folder.
#
Console.WriteLine("Changing name from " + vo.name +
" to yo_csharp.");
vo.name = "yo_csharp";
vo.dirtyAttributes = new String[]{"name"};
# Manually set dirty fields being changed.
#
vo = fs.update(folder, vo, true, true);
Console.WriteLine("Folder name changed to: " + vo.name);
```

Handle exceptions

```
    # .NET converts Web Services faults into SoapExceptions
# without trying to deserialize them into application
# exceptions first. As a result, your code cannot catch
# application exceptions. As a workaround, the C# stubs
# provided by SA include SOAPExceptionParser,
# a class that enables you to get information from
# SOAPExceptions. The following code shows how to get the
```

```
# exception name and error message by calling the getDetail
# method of SoapExceptionParser.
#
try{
// Try to get a non-existent folder here.
} catch(SoapException e){
    SoapExceptionDetail detail =
    SoapExceptionParser.getDetail(e);
    Console.WriteLine("SoapExceptionDetail.name: " +
    detail.exceptionName);
    Console.WriteLine("SoapExceptionDetail.msg: " +
    detail.message);
    ...
}
```

Password security with C#

The FolderService method reads the user and password pair from the file App.config. The following shows an example of this method.

```
        User user = new User();
user.username = "user";
user.password = "password";
FolderService fs = new FolderService();
fs.Url = wsdlUrlPrefix + "com.opsware.folder/FolderService";
fs.user = user;
```

If you do not want to store the password in clear text in the App.config file, you can use the SecureUser class to encrypt the password. The SecureUser class uses the C# SecureString in .NET 2.0. Passwords are stored encrypted in a SecureString. Furthermore, the getPassword() method is only visible internally. SecureUser is a static class, so you only need to set your user name and password once or each time you switch users.

Each service retrieves the user name and password from SecureUser first and then its user member variable and then App.config, for backward compatibility. SecureUser takes either a String or a SecureString for the password. In either case, clients are responsible to clean up the password variable passed to the SecureUser.setUser() method.

At some point the password will need to be converted to a regular C# string in memory, which will only get freed when the next garbage collection occurs. Using SecureUser will only ensure internal password storage is secure.

The following example shows how to set the user name and password securely.

```
        SecureString passwd = new SecureString();  
passwd.AppendChar('p');  
passwd.AppendChar('a');  
passwd.AppendChar('s');  
passwd.AppendChar('s');  
passwd.AppendChar('w');  
passwd.AppendChar('d');  
SecureUser.setUser("username", passwd); // that's it, no need to set up user  
for each service.  
passwd.Dispose(); // resets passwd and frees up memory so no copy remains from  
caller.
```

Pluggable checks

The SA Audit and Remediation feature enables you to define and monitor the compliance information for SA managed servers. Because compliance standards are continuously evolving, SA lets you create specialized custom checks and policies, and extend those provided with SA. A pluggable check is an audit rule, which belongs to one or more audit policies. You create a pluggable check in a command-line environment, upload the check, and then add it to an audit policy with the SA Client.

This section is intended for software developers who are familiar with XML and with the Audit and Remediation feature of SA.

- ["Setup for pluggable checks" below](#)
- ["Pluggable check tutorial" on the next page](#)
- ["Audit and remediation " on page 175](#)
- ["Creating a pluggable check " on page 177](#)
- ["Creating the audit policy " on page 185](#)
- ["Document Type Definition \(DTD\) for config.xml file" on page 186](#)

Setup for pluggable checks

Before developing pluggable checks:

1. Install an SA core in a development environment. Do not use a production core.
2. On a server that has an installed Agent, install OCLI 1.0. For information on the OCLI 1.0, see [Use](#).

Pluggable check tutorial

This tutorial shows how to create a pluggable check named HelloWorld Check. This simple check verifies that the `/var/tmp/helloworld` file exists on a Unix managed server. If the file does not exist, the remediation script of the pluggable check creates the file.

To develop the HelloWorld check:

1. Follow the instructions in "[Setup for pluggable checks](#)" on the previous page. The server where you install OCLI 1.0 will be the development server for this tutorial.
2. The HelloWorld Check example code is included with the [ZIP file](#) that contains the API code examples.
3. Unzip the file you downloaded in the preceding step and verify that the `pluggable_checks/helloworld` directory contains the following files:
 - `config.xml`
 - `gethelloworld.py`
 - `sethelloworld.py`

The HelloWorld check is made up of these three files. The `config.xml` file is a configuration file. The `gethelloworld.py` Python script performs the audit. The `sethelloworld.py` Python script performs the remediation. In the following steps, you package these files into a ZIP file and then import the ZIP file into SA.

4. On your development server, copy the unzipped `helloworld` files to a working directory, for example:

```
cd /home/jdoe/dev
mkdir helloworld
cd helloworld
cp unzip_dest/pluggable_checks/helloworld/* .
```

5. Obtain a Globally Unique ID (GUID). Each pluggable check requires a GUID. You can acquire a valid GUID by using one of the following techniques:
 - Log on to web sites such as the following:
<http://kruithof.xs4all.nl/uuid/uuidgen>
 - Download the free Windows tool `guidgen` from:
<http://www.microsoft.com/downloads/details.aspx?FamilyID=94551F58-484F-4A8C->


```
BB39-ADB270833AFC&displaylang=en
```

If you programmatically create your GUIDs, then your code should conform to RFC4122 (<http://www.ietf.org/rfc/rfc4122.txt>).

6. With a text editor, insert the GUID in the `config.xml` file, for example:

```
<checkGUID>6c7ed38c-d8d6-11db-8314-0800200c9a66</checkGUID>
```

This is the only element in `config.xml` that you need to modify for this tutorial.

7. In the text editor, save `config.xml` with the change you made for the GUID.

Keep the text editor open. Throughout this tutorial, you will examine various elements in `config.xml` to learn how they map to the Python scripts and the SA Client display fields of the HelloWorld Check.

8. In the `config.xml` file, note the following elements, which are related to the audit (get) and remediation (set) scripts of the HelloWorld Check:

```
<!-- The name of the script that performs the check. -->  
<checkGetScriptName>gethelloworld.py</checkGetScriptName>
```

```
<!-- The name of the script that remediates the audit. -->  
<checkSetScriptName>sethelloworld.py</checkSetScriptName>
```

```
<!-- The exit code of the gethelloworld.py script will be checked.-->  
<checkReturntype>EXITCODE</checkReturntype>
```

```
<!-- A string argument is passed to gethelloworld.py. -->  
<checkGetArgumentType>STRING</checkGetArgumentType>
```

```
<!-- The default argument for gethelloworld.py is the name of the file the  
script is checking for. -->  
<checkGetArgumentDefaultValue>/var/tmp/helloworld  
</checkGetArgumentDefaultValue>
```

```
<!-- If the helloworld file exists, the exit code of gethelloworld.py is 0.  
-->  
<checkSuccessExitCodeValue>0</checkSuccessExitCodeValue>  
<!-- If the helloworld file does not exist, the exit code of  
gethelloworld.py is 1. -->  
<checkSuccessExitCodeValue>1</checkSuccessExitCodeValue>
```

9. Examine the `gethelloworld.py` script, which performs the audit by checking for the existence of

the file `/var/tmp/helloworld`. You do not need to edit this script for this tutorial. Later in this tutorial ([step 30](#), when you run the audit in the SA Client, this script executes on a managed server.

The `/var/tmp/helloworld` string is the default argument of the script, as indicated by the value of `<checkGetArgumentDefaultValue>` in `config.xml`. The script's exit code (`result`) corresponds to the values specified for `<checkSuccessExitCodes>`.

Here is the source code for the `gethelloworld.py` script:

```
import sys
import os
import string
if __name__ == "__main__":
    if len(sys.argv) != 2:
        sys.stderr.write("No argument found! Please enter a
            file name!\n")
        sys.exit(220)
    filename = sys.argv[1]
    if os.path.isfile(filename) or os.path.isdir(filename):
        result = 0
    else:
        result = 1
    sys.stderr.write("Debugging: Found result %s\n"
        % result)
    sys.stdout.write("%s\n" % result)
    sys.exit(result)
```

10. Next, examine the remediation script `sethelloworld.py`, which creates the `/var/tmp/helloworld` file. This script runs on a managed server if you decide to remediate the audit in [step 35](#). Do not change the script for this tutorial.

The source code for `sethelloworld.py` follows:

```
import sys
import os
import string
if __name__ == "__main__":
    if len(sys.argv) != 2:
        sys.stderr.write("No argument found!
            Please enter a file name!\n")
        sys.exit(220)
    filename = sys.argv[1]
    if os.path.isfile(filename) or os.path.isdir(filename):
        # Do nothing because the file already exists.
```

```
        pass
    else:
        try:
            fd = open(filename, "w")
            fd.write(" ")
            fd.close()
        except:
            sys.stderr.write("Could not open file %s for
                writing!\n" % filename)
            sys.exit(220)
# Exit successfully with a 0 exit code.
sys.stderr.write("Successfully created file\n")
sys.exit(0)
```

11. Package the HelloWorld Check.

To package the HelloWorld pluggable check, archive the contents of the working directory into a single ZIP file, for example:

```
cd /home/jdoe/dev/helloworld
zip ../helloworld.zip *
```

12. Verify that the ZIP file contains the two Python scripts and the `config.xml` file by entering the following `unzip` command:

```
unzip -t ../helloworld.zip
testing: config.xml OK
testing: gethelloworld.py OK
testing: sethelloworld.py OK
No errors detected in compressed data of ../helloworld.zip.
```

13. Import the pluggable check into SA with the `oupload` command of OCLI 1.0:


```
oupload -C"Customer Independent" \
-t"Server Configuration Check" \
--forceoverwrite --old -O"SunOS 5.8" ../helloworld.zip
```

Note: The platform option (`-O`) is `SunOS 5.8` for all Unix and Linux checks. For Windows checks, the platform option is `Windows 2003`.

If `oupload` does not run successfully, make sure that you have installed the correct version of OCLI 1.0, set the `PATH` environment variable correctly, and included the `login` file in your environment. For details on these requirements, see the OCLI 1.0 in the Using

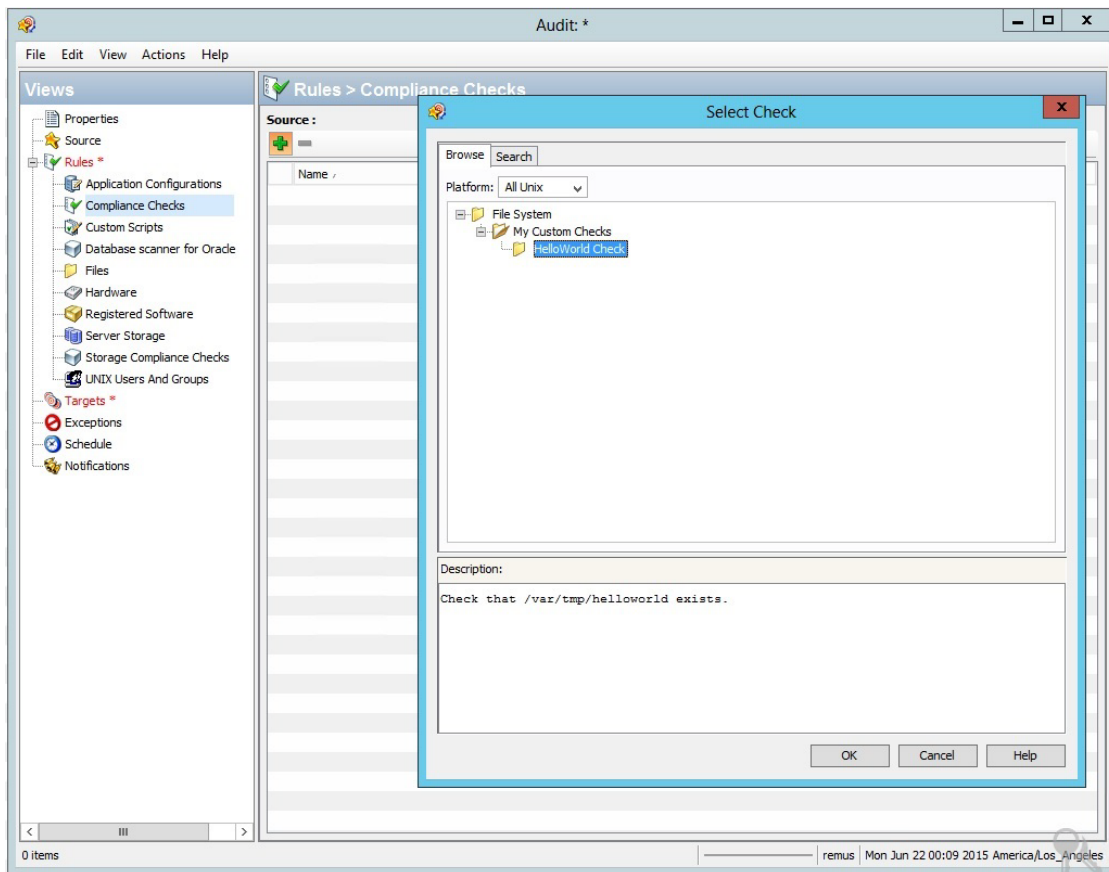
14. Open the SA Client.

In the next few steps, you create a new audit, adding to it the HelloWorld Check you imported with the `upload` command.

15. From the **Tools** menu, select **Update Cache**.
16. From the Navigation pane, select **Library > By Type > Audits and Remediation > Audits > Unix**.
17. From the **Actions** menu, select **New**.
18. In the Audit Window, in the Name field of the Properties pane, enter HelloWorld Audit.
19. In the Views pane, In the Views pane, select **Rules > Compliance Checks**.
20. Click the **Add** button , and then click **File System**.

The Content pane should list the HelloWorld Check under Available for Audit, as shown in the following figure:

HelloWorld check in the rules for a file system



21. In the `config.xml` file, note the following elements, which are related to the information displayed.

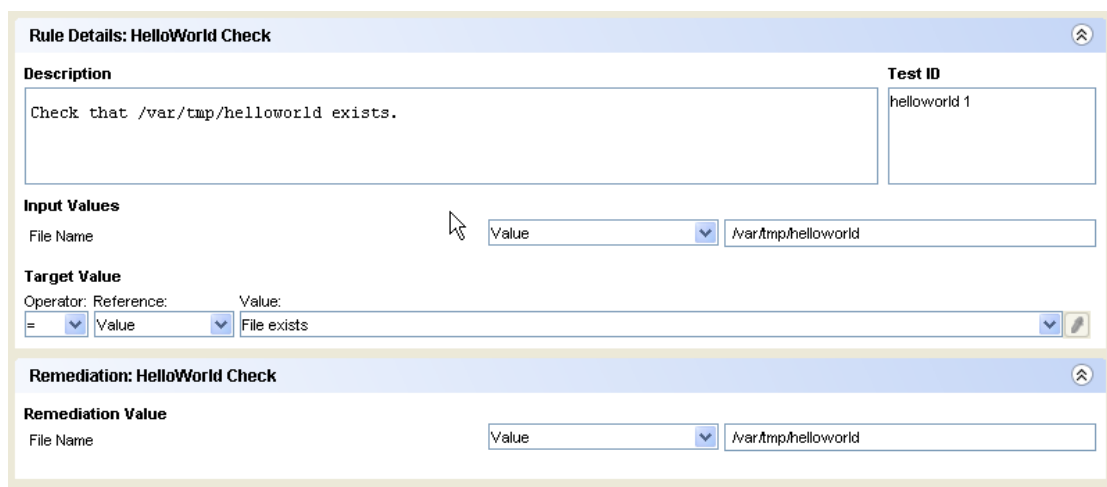
```
<!-- The check name is the rule name shown in the SA Client. -->  
<checkName>HelloWorld Check</checkName>
```

```
<!-- The category corresponds to the rule hierarchy displayed by the SA  
Client. -->  
<checkCategory>File System|My Custom Checks</checkCategory>
```

In the Audit Window of the SA Client, under Available for Audit, select HelloWorld Check and click the plus sign.

The Content pane should list the details for HelloWorld Check, as shown in the following figure.

HelloWorld check rule details



22. In the `config.xml` file, examine the following elements, which are related to the information displayed under the "HelloWorld check rule details" above figure:

```
<!-- The following value appears under Description in the Rule Details of  
the SA Client. -->  
<checkDefaultDescription>  
Check that /var/tmp/helloworld exists.  
</checkDefaultDescription>
```

```
<!-- The following element corresponds to the Test ID in the SA Client. -->  
<checkTestID>helloworld 1</checkTestID>
```

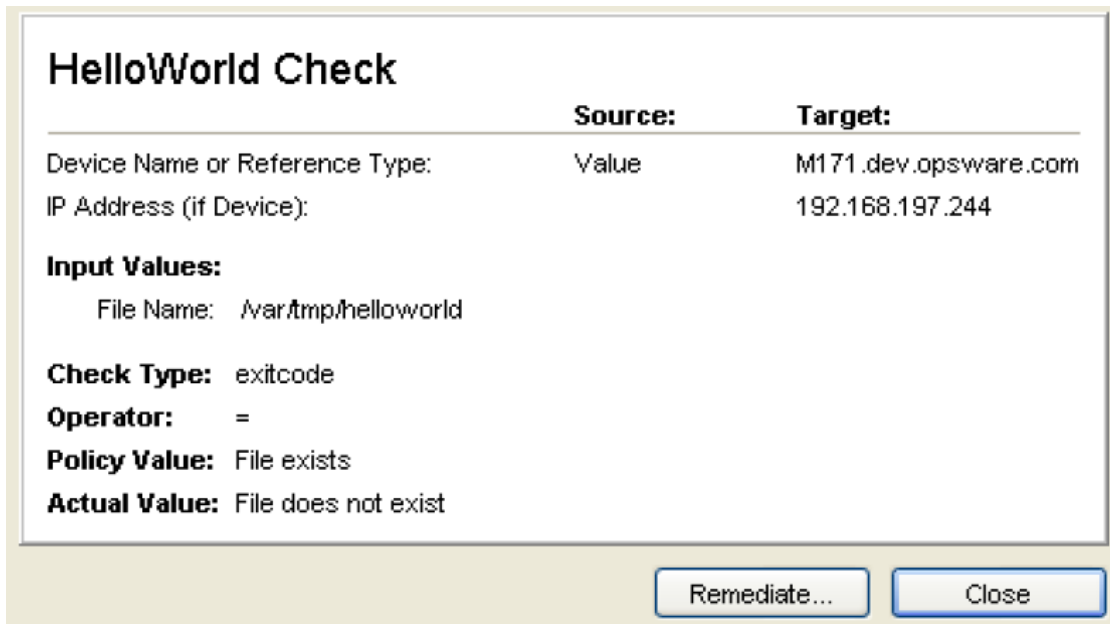
```
<!-- This label is under Input Values in the SA Client. -->  
<checkGetArgumentDefaultLabel>File Name  
</checkGetArgumentDefaultLabel>
```

```
<!-- The default argument to the gethelloworld.py script also appears
```

```
under Input Values in the SA Client. -->  
<checkGetArgumentDefaultValue>/var/tmp/helloworld  
</checkGetArgumentDefaultValue>
```

23. In the Views pane of the SA Client, select **Targets**.
24. In the following steps you add a target server to HelloWorld Audit. In later steps, the `gethelloworld.py` and `sethelloworld.py` scripts will run on the target server.
25. In the Contents pane, click **Add**.
26. In the Select Server window, drill down to a server and click **OK**.
In the Audit window, select **File> Save**.
27.
At this point, the HelloWorld Audit contains the HelloWorld Check (rule) and is associated with a target server.
28. In the Audit window, from the **Actions** menu, select **Run Audit**.
29. Step through the windows of the Run Audit task.
30. In the Run Audit window, click **Start Job**.
This action launches the job that runs the `gethelloworld.py` script on the target server.
31. After the job has completed, click **View Results**.
32. In the Views pane of the Audit Result window, select Policy Rules (1).
33. In the Content pane of the Audit Result window, open HelloWorld Check.
The Difference Details window should appear, as shown in the following figure.

HelloWorld check difference details



34. In the `config.xml` file, note the following elements, which are related to the information displayed in the "HelloWorld check difference details" on the previous page figure:

```
<!-- The following value appears as the Policy Value in the Difference
Details window. -->
<checkSuccessExitCodeDefaultDisplayName>
File exists</checkSuccessExitCodeDefaultDisplayName>
```

```
<!-- The next value appears as the Actual Value in the same window. -->
<checkSuccessExitCodeDefaultDisplayName>
File does not exist</checkSuccessExitCodeDefaultDisplayName>
```

35. If you want to create `/var/tmp/helloworld` on the target server, on the Differences Window, click **Remediate**.

This action runs the `sethelloworld.py` script. For more information, see the Server Automation Administration Guide on the HPE SSO portal.

Audit and remediation

Sarbanes-Oxley (SoX), Information Technology Infrastructure Library (ITIL), and ISO20000 make it urgent to keep server configurations in compliance. The SA Audit and Remediation feature offers you a well-organized set of policies to help you address compliance issues. A graphical interface makes it

easy for you to select and run audits against specified servers, and see how well they comply with professional standards.

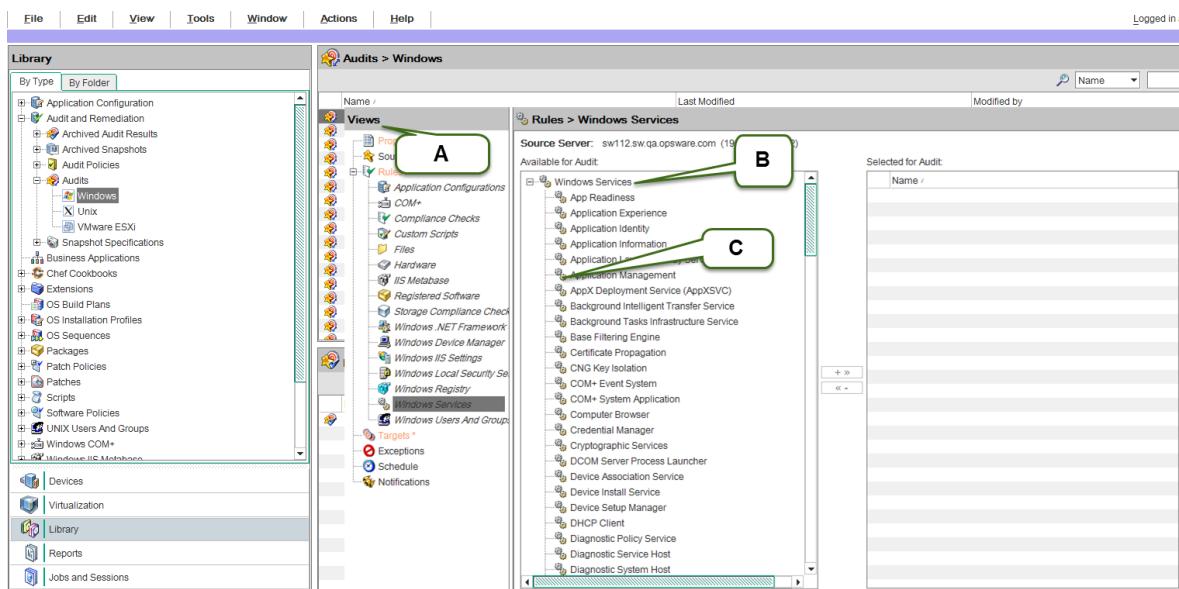
Audit and Remediation also simplifies system administration. For example, you might monitor a class of servers that run a home grown application built by your team, such as a database server or middleware application. As you configure and monitor the servers that run the application, you keep a list that tracks the ideal state of the configuration. Such a list might include file, directory, and network share permissions.

You can create an audit that defines these configurations, then audit the servers after installing the application. The audit results will confirm whether or not the application is installed and has been configured successfully according to your criteria. If the configuration is non-compliant, you can create an ad-hoc audit to troubleshoot the problem. When the audit results indicate an error, you can remediate the server to match your ideal configuration. To ensure that the configuration change works in production, you can set the audit to run on a configurable schedule and have a notification sent upon completion.

Showing a window for selecting an audit, the following figure includes the following callouts:

- **Callout A:** Any category listed in the Views panel may have SA non-modifiable capabilities, or modifiable pluggable checks.
- **Callout B:** This points to the SA capabilities for dealing with Windows services.
- **Callout C:** This lists pluggable checks for working with Windows Services.

Windows Services audit rule



Each check evaluates one rule. Several checks can be bundled together into a policy.

The SA Audit and Remediation feature comes with many out-of-the-box checks. You can run most audits by selecting the desired check. The choice of audits grows continuously as developers design, code, test, and add more checks to the system through the HPE Live Network. These checks are imported as complete policies.

However, since every business has unique challenges and unique resources, you may need to determine compliance against a set of criteria not available for auditing within the SA Audit and Remediation framework. For this reason, the system provides a way to create your own custom pluggable checks.

The Audit and Remediation feature evaluates, by specific rules, the compliance state of servers under SA management. This feature can also remediate the servers that do not match the desired configuration state as defined in the rules. These rules include various server parameters, registry values, file permissions, application configurations, file existence, COM+ objects, and more.

In the Windows environment, web server rules can also be specified with application configuration, which is based upon the Microsoft Internet Information Services (IIS) Web server configuration file, `UrlScan.ini`. SA can compare partial or full values from specific configuration files, select the desired elements from the file, and make sure that these values or configuration file entries exist. For more information, see the Application Configuration.

SA includes many predesigned audit rules. Each defines a desired state of configuration for a server or server groups. Some rules are value-based, providing a comparator (`<`, `>`, `==`, `!=`, `contains`, etc.), a value or set of values, and one or more checks, which spell out the underlying code used to evaluate the state of the audited item or items. The comparison data determines compliance or non-compliance. A rule may also contain remediation values if the check supports remediation.

A rule consists of a single check. You can create new functionality by using custom content objects in the form of pluggable checks. You can also bundle related pluggable checks into audit policies for convenience.

Creating a pluggable check

A pluggable check is code that is downloaded to the managed server or servers and is executed by the Audit and Remediation framework. You can use checks to extend the native Audit and Remediation properties and to provide additional specialized functionality. Each pluggable check includes a customized `config.xml` file and at least one script that compares the audited feature against values specified in the `config.xml` file. A pluggable check may also include a script that sets specified variables in the audited server to the value specified in the `config.xml` file. You can write pluggable check scripts in Python, Visual Basic Scripting (VBS), BAT, or shell script. A pluggable check is packaged as a zip archive.

Most of the CIS checks are direct translations of the CIS benchmarks. More information can be found at <http://www.cisecurity.org>.

Most types of checks fall into one of the following categories:

- Windows Registry checks
- Unix Services checks
- User checks, which may use password or shadow file information

Guidelines for pluggable checks

To simplify server maintenance, adhere to the following guidelines:

- When creating a new pluggable check, pay special attention to the names. Describe the purpose of the check, and replace spaces with an underscore. For example, `Users_Without_Password_Expiration` is self-explanatory. This will help you to find a check quickly when a server acquires several hundred or more checks.
- Write a generic check. This enables you to easily create additional checks of the same execution type with only a few lines of code change. For example, for most CIS2k3 Windows Service Checks, you can change a single line of code to create a new check for a new service.
- When naming the audit (get) and remediation (set) scripts, remove the spaces or underscores from the directory name, and prefix with get or set, as appropriate. For example, `getUsersWithoutPasswordExpiration.sh` is a good name for an audit file. Be consistent on this, even if you think your custom check will not be used by anyone else.
- Pay attention to error checking. Remember that unexpected return values might report an audit as non-compliant when a script failure occurs. Trap the unexpected error or exception, and write out information about it to stdout or stderr to simplify troubleshooting.
- Convert most checks to a simple binary case of True or False when possible.
- Always try to handle not only the specific benchmark case, but also its counterpart. For example, you can easily create a “Disable Service X,” pluggable check at the same time that you create an “Enable Service X” and reuse most of the code. This can be useful if you decide later to test for the opposite condition.
- Use the standard exit codes defined by the framework whenever possible. These are:

```
EXIT_FAILURE=220
EXIT_ERR_USAGE=221
EXIT_ERR_INVALID_OS=222
```

- When returning disabled or enabled in a Boolean type check, return 0 for disabled, 1 for enabled.
- Package each pluggable check as a ZIP archive. A single file system directory contains the files listed in the following [table](#).

Pluggable check contents

File Name	Description
<code>config.xml</code>	(Required) The XML configuration file defining how this pluggable check executes, returns, and ultimately reports compliance or non-compliance.
<code>getName.</code> { <code>py</code> <code>sh</code> <code>BAT</code> <code>vbs</code> }	(Required) The audit script, written in Python, VBS, BAT, or shell, that evaluates the audited object, and returns text and exit codes according to the <code>config.xml</code> definitions.
<code>setName.</code> { <code>py</code> <code>sh</code> <code>BAT</code> <code>vbs</code> }	(Optional) The remediation script, written in Python, VBS, BAT, or shell, that remediates the condition checked by the audit script.
<i>Additional Code, Scripts, or Libraries</i>	(Optional) Helper and supplementary scripts used by either the audit or remediation scripts.

The file names for the audit and remediation scripts do not need to begin with `get` and `set`, but this convention simplifies file maintenance.

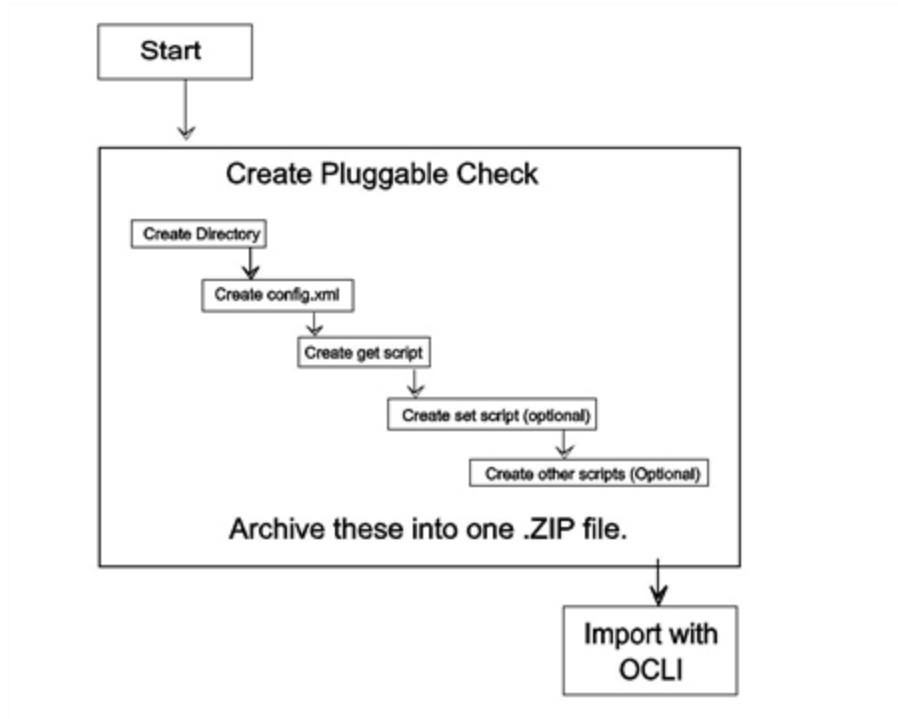
The following example shows a directory structure for a pluggable check:

```
./check_name/  
./check_name/config.xml  
./check_name/getcheckname.py  
./check_name/setcheckname.py
```

Development process for pluggable checks

The following figure shows an overview for the development process, which takes place in a command-line environment.

Development process



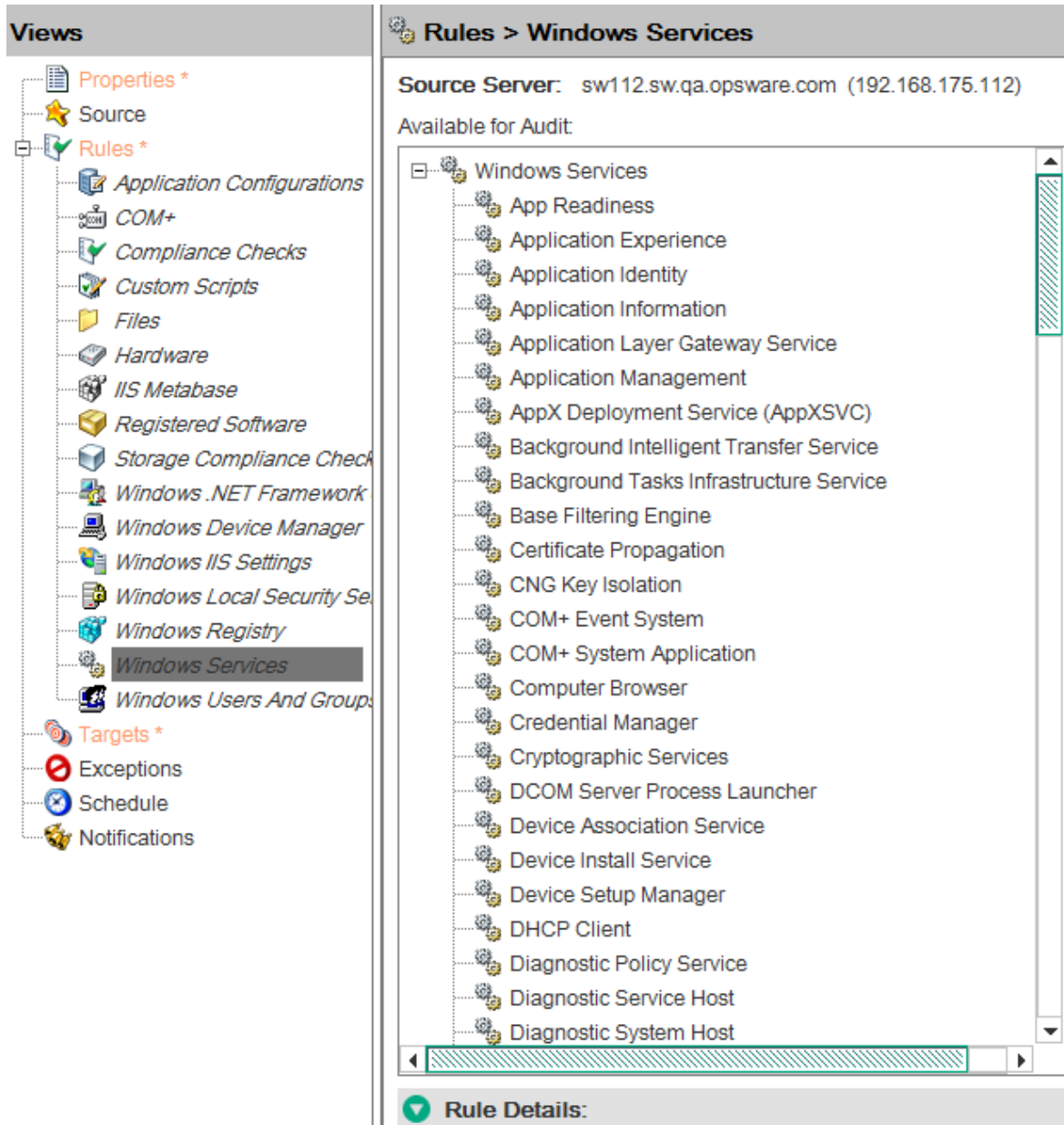
Pluggable check configuration (config.xml)

The config.xml file is a specification file for the pluggable check that contains elements to control how this check appears in the SA Client, default values, value types for comparison, and the category of the check. For example, the following element in the config.xml file determines the pluggable check's rule category in the SA Client:

```
<checkCategory>Windows Services</checkCategory>
```

Standard categories, each indicated with its own icon, include hardware, software, operating systems, users and groups, file systems, and more, as shown in the following [figure](#).

Pluggable check categories in the rule hierarchy



The following listing shows the template for the config.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE checkConfiguration SYSTEM "check.dtd">
<checkConfiguration version="1.0">
<checkName>$CHECKNAME</checkName>
<checkGUID>$CHECKGUID</checkGUID>
<checkDefaultDescription>$CHECKDESCRIPTION</checkDefaultDescription>
<checkRemediationDefaultDescription> $CHECKREMEDIATIONDESCRIPTION </
checkRemediationDefaultDescription>
<checkGetScriptName>$GETSCRIPTNAME</checkGetScriptName>
<checkGetScriptType>PY</checkGetScriptType><!-- Or SH for shell, BAT for Bat,
```

```
VBS for Visual Basic -->
<checkSetScriptName>$SETSCRIPTNAME</checkSetScriptName><!-- Optional -->
<checkSetScriptType>PY</checkSetScriptType><!-- Optional -->
<checkVersion>32b.0-1.0</checkVersion>
<checkReturnTypes>$RETURNTYPE</checkReturnTypes> <!-- EXITCODE, STRING, or
NUMBER -->
<checkTestIDs>
<checkTestID>$CHECKTESTID</checkTestID> <!-- Optional -->
</checkTestIDs>
<checkPlatformTypes>
<checkPlatform>$PLATFORMTYPE</checkPlatform> <!-- Currently Unix or Windows --
>
</checkPlatformTypes>
<checkCategories>
<checkCategory>$CATEGORY</checkCategory> <!-- Top-level GUI category -->
</checkCategories>
<checkGetArguments> <!-- All arguments are optional -->
<checkGetArgument>
<checkGetArgumentType>$GETARGTYPE</checkGetArgumentType> <!-- STRING or NUMBER
-->
    <checkGetArgumentDefaultLabel>$GETDEFAULTLABEL</
checkGetArgumentDefaultLabel>
        <checkGetArgumentDefaultDescription>$GETDEFAULTDESCRIPTION</
checkGetArgumentDefaultDescription>
            <checkGetArgumentDefaultValue>$GETDEFAULTVALUE</
checkGetArgumentDefaultValue>
        </checkGetArgument>
</checkGetArguments>
<checkSetArguments> <!-- Also optional -->
<checkSetArgument>
<checkSetArgumentType>$SETARGTYPE</checkSetArgumentType>
    <checkSetArgumentDefaultLabel>$SETDEFAULTLABEL</
checkSetArgumentDefaultLabel>
        <checkSetArgumentDefaultDescription>$SETDEFAULTDESCRIPTION</
checkSetArgumentDefaultDescription>
            <checkSetArgumentDefaultValue>$SETDEFAULTVALUE</
checkSetArgumentDefaultValue>
        </checkSetArgument>
</checkSetArguments>
<checkSuccessExitCodes> <!-- Only for EXITCODE type checks, generally at least
two entries -->
    <checkSuccessExitCode>
<checkSuccessExitCodeValue>$EXITCODEVALUE</checkSuccessExitCodeValue>
        <checkSuccessExitCodeDefaultDescription>$EXITCODEDESCRIPTION
    </checkSuccessExitCodeDefaultDescription>
        <checkSuccessExitCodeDefaultDisplayName>$EXITCODEDISPLAYNAME
    </checkSuccessExitCodeDefaultDisplayName>
</checkSuccessExitCode>
</checkSuccessExitCodes>
```

```
</checkConfiguration>
```

For more details, see ["Document Type Definition \(DTD\) for config.xml file" on page 186](#).

Audit (get) scripts

You can design the audit script, also known as the get script, to obtain a value from a managed server. The script is executed with optional parameters, as specified in the `config.xml` file. If the script is running an EXITCODE check, the result of the script is compared to the exit codes specified in the `config.xml` file. For STRING and NUMBER return type checks, the result is compared to what is written to STDOUT.

An audit script has a set of pre-defined return codes. You can define additional return codes in the check `config.xml` file.

The audit script may display informational messages. These messages are useful when troubleshooting an audit script failure. Review the following sample Python audit script:

Remediation (set) scripts

You can design the remediation script, also known as the set script, to enact a change on the managed server that would cause the audit script to return success when completed. The script is executed with optional parameters, as specified in the check `config.xml` file.

These set scripts are optional, and can vary in character from being very similar to their counterpart get scripts to entirely different (and longer).

From a shell standpoint, there is nothing special in the script itself, other than the return codes being used. Most checks display some debug output or information messages. This is not normally seen by users, except in the event of a script failure, where the messages are useful for troubleshooting purposes.

As a standard practice, always include at least one parameter to the set script. Also, remember to modify the `config.xml` file so that it displays nicely in the SA Client when adding a set script to an already existing check.

Make sure your remediation scripts exit with exit code 0 to indicate success. All other exit codes will indicate failure of the remediation operation.

Review the following sample Python set script.

```
import sys
import os
import string
if __name__ == "__main__":
# If there are set arguments they will be loaded into
# sys.argv
# Enter the desired set code here. Stdout may be used for
```

```
# debugging.  
# Uses exitcode 0 for success, and all other values for  
# failure.  
# enter condition where set script if successful. for this  
# example, use 'if 1'  
if 1:  
    sys.exit(0)  
else:  
    sys.exit(-1)
```

Other code for pluggable checks

Pluggable checks may also contain code other than the get or set scripts. Libraries, executables or additional scripts can be added to the check, so their set or get scripts can utilize these upon execution.

You can also include additional code in the ZIP file.

Zipping up pluggable checks

After you have created the `config.xml` file, the audit (get) script, and the optional remediation (set) script, create a ZIP archive containing these files. The following shell history shows the creation process in a UNIX environment.

```
# ls  
check_name  
# cd check_name  
# zip ../checkname.zip *  
adding: config.xml  
adding: getcheckname.py  
adding: setcheckname.py  
# unzip -t ../checkname.zip  
testing: config.xml OK  
testing: getcheckname.py OK  
testing: setcheckname.py OK  
No errors detected in compressed data of ../checkname.zip.
```

Importing pluggable checks

Import a pluggable check into an SA core or mesh using the OCLI 1.0 utility, which is documented in the SA Content Utilities. The following shell history provides an example of the import process for Linux:

```
# cp checkname.zip /var/tmp/checks  
# cd /var/tmp/checks  
# cp opsware_32.a.692.0-upload/disk001/packages/Linux/3AS/ocli-32a.2.0.5-  
linux-3AS .  
# chmod 755 ocli-32a.2.0.5-linux-3AS  
# ./ocli-32a.2.0.5-linux-3AS  
# . ./ocli/login.sh  
# export PATH=/opt/opsware/bin:$PATH
```



```
# oupload -C"Customer Independent" -t"Server Configuration Check" --  
forceoverwrite --old -O"SunOS 5.8" your_Pluggable_check.zip
```

The oupload command uses "SunOS 5.8" to specify that the check falls into the generic Unix category in the SA Client. To specify a check for the Windows category, use "Windows 2003."

Creating the audit policy

The audit policy creation procedure is illustrated in figure below:

Procedure for creating an audit policy



Creating an audit policy

Audit policies consist of rules. Each rule consists of one or more checks, which can include the user-created pluggable check. Audit policies and rules are displayed, created and edited in the SA Client. The following figure shows a list of the audit rules available on a model system.

List of audit rules

Name	Last Modified	Modified by
a_reg_soft	Mon May 09 11:46:04 2016	ileana
script	Mon May 09 11:45:00 2016	oleto
TestElena	Mon May 09 11:43:28 2016	ileana
a_from_sn_all	Tue Mar 01 20:49:23 2016	ileana
a_3ap	Tue Mar 01 20:27:10 2016	ileana
a_file	Tue Mar 01 20:23:39 2016	ileana
a_comp_checks	Mon Feb 29 21:05:57 2016	ileana
audit_2	Mon Feb 29 19:44:09 2016	ileana
audit_from_sn_result	Mon Feb 29 19:20:56 2016	ileana

Name	Source	Compliant Rules	Non-Compliance	Failed Rules	Created
script	-	0	0	1	Mon 9

For detailed information on creating an audit policy, see the Server Automation Administration Guide on the HPE SSO portal.

Exporting the audit policy

To move a new audit policy to other SA cores, export it from one and import it to another using the DCML Exchange Tool (DET) command-line utility. Use this tool to populate a newly-installed SA core with content, such as policies, from an existing core. For detailed instructions on this procedure, see the Server Automation Administration Guide on the HPE SSO portal.

Document Type Definition (DTD) for config.xml file

This file governs SA Client display names and descriptions, default values, comparisons to be performed upon values returned by the check code, the category of the SA Client displaying these values, and more.

Two elements in the default config.xml file, checkGetArguments and checkSetArguments, are used to pass data values to the scripts at execution time. If your programmable check does not require any arguments, delete these elements from your config.xml file.

The following DTD for config.xml is dynamically generated by SA:

```
<!ELEMENT checkConfiguration (checkName, checkGUID, checkDefaultDescription,
checkRemediationDefaultDescription?, checkGetScriptName?,
checkGetScriptType?, checkSetScriptName?, checkSetScriptType?, checkVersion,
checkAllowRemediationOnFailure?, checkReturnType, checkTestIDs?,
checkPlatformTypes, checkExclusivePlatforms?, checkExcludePlatforms?,
checkCategories, checkGetArguments?, checkSetArguments?,
checkComparisonDefaults?, checkCompareValidValues?, checkSuccessExitCodes?)>
<!ATTLIST checkConfiguration version CDATA #REQUIRED>
<!ELEMENT checkName (#PCDATA)>
<!ELEMENT checkGUID (#PCDATA)>
<!ELEMENT checkDefaultDescription (#PCDATA)>
<!ELEMENT checkRemediationDefaultDescription (#PCDATA)>
<!ELEMENT checkGetScriptName (#PCDATA)>
<!ELEMENT checkGetScriptType (#PCDATA)>
<!ELEMENT checkSetScriptName (#PCDATA)>
<!ELEMENT checkSetScriptType (#PCDATA)>
<!ELEMENT checkVersion (#PCDATA)>
<!ELEMENT checkAllowRemediationOnFailure (#PCDATA)>
<!ELEMENT checkReturnType (#PCDATA)>
<!ELEMENT checkTestIDs (checkTestID+)>
<!ELEMENT checkTestID (#PCDATA)>
```

```
<!ELEMENT checkPlatformTypes (checkPlatform+)>
<!ELEMENT checkPlatform (#PCDATA)>
<!ELEMENT checkExclusivePlatforms (checkExclusivePlatform+)>
<!ELEMENT checkExclusivePlatform (#PCDATA)>
<!ELEMENT checkExcludePlatforms (checkExcludePlatform+)>
<!ELEMENT checkExcludePlatform (#PCDATA)>
<!ELEMENT checkCategories (checkCategory+)>
<!ELEMENT checkCategory (#PCDATA)>
<!ELEMENT checkGetArguments (checkGetArgument+)>
<!ELEMENT checkGetArgument (checkGetArgumentType,
checkGetArgumentDefaultLabel, checkGetArgumentDefaultDescription,
checkGetArgumentDefaultValue?, checkGetArgumentValidValues?)>
<!ELEMENT checkGetArgumentType (#PCDATA)>
<!ELEMENT checkGetArgumentDefaultLabel (#PCDATA)>
<!ELEMENT checkGetArgumentDefaultDescription (#PCDATA)>
<!ELEMENT checkGetArgumentDefaultValue (#PCDATA)>
<!ELEMENT checkGetArgumentValidValues (checkGetArgumentValidValue+)>
<!ELEMENT checkGetArgumentValidValue (checkGetArgumentValidValueItem,
checkGetArgumentValidValueDisplayName)>
<!ELEMENT checkGetArgumentValidValueItem (#PCDATA)>
<!ELEMENT checkGetArgumentValidValueDisplayName (#PCDATA)>
<!ELEMENT checkSetArguments (checkSetArgument+)>
<!ELEMENT checkSetArgument (checkSetArgumentType,
checkSetArgumentDefaultLabel, checkSetArgumentDefaultDescription,
checkSetArgumentDefaultValue?, checkSetArgumentValidValues?)>
<!ATTLIST checkSetArgument populateFromRule CDATA #IMPLIED>
<!ELEMENT checkSetArgumentType (#PCDATA)>
<!ELEMENT checkSetArgumentDefaultLabel (#PCDATA)>
<!ELEMENT checkSetArgumentDefaultDescription (#PCDATA)>
<!ELEMENT checkSetArgumentDefaultValue (#PCDATA)>
<!ELEMENT checkSetArgumentValidValues (checkSetArgumentValidValue+)>
<!ELEMENT checkSetArgumentValidValue (checkSetArgumentValidValueItem,
checkSetArgumentValidValueDisplayName)>
<!ELEMENT checkSetArgumentValidValueItem (#PCDATA)>
<!ELEMENT checkSetArgumentValidValueDisplayName (#PCDATA)>
<!ELEMENT checkComparisonDefaults (checkComparisonDefaultOperator?,
checkComparisonDefaultValues)>
<!ELEMENT checkComparisonDefaultOperator (#PCDATA)>
<!ATTLIST checkComparisonDefaultOperator not CDATA #IMPLIED>
<!ATTLIST checkComparisonDefaultOperator caseInsensitive CDATA #IMPLIED>
<!ELEMENT checkComparisonDefaultValues (checkComparisonDefaultValue+)>
<!ELEMENT checkComparisonDefaultValue (checkComparisonDefaultValueItem,
checkComparisonDefaultValueDisplayName)>
<!ELEMENT checkComparisonDefaultValueItem (#PCDATA)>
<!ELEMENT checkComparisonDefaultValueDisplayName (#PCDATA)>
<!ELEMENT checkCompareValidValues (checkCompareValidValue+)>
<!ELEMENT checkCompareValidValue (checkCompareValidValueItem,
checkCompareValidValueDisplayName)>
<!ELEMENT checkCompareValidValueItem (#PCDATA)>
```

```
<!ELEMENT checkCompareValidValueDisplayName (#PCDATA)>
<!ELEMENT checkSuccessExitCodes (checkSuccessExitCode+)>
<!ELEMENT checkSuccessExitCode (checkSuccessExitCodeValue,
checkSuccessExitCodeDefaultDescription,
checkSuccessExitCodeDefaultDisplayName)>
<!ELEMENT checkSuccessExitCodeValue (#PCDATA)>
<!ELEMENT checkSuccessExitCodeDefaultDescription (#PCDATA)>
<!ELEMENT checkSuccessExitCodeDefaultDisplayName (#PCDATA)>
```

The following table describes the elements of the config.xml DTD.

DTD Elements and Attributes

Elements	Attributes
checkConfiguration version	Set to 1.0, only change if the Audit and Remediation framework requires it.
checkName	The English name that displays in the SA Client for the check/rule.
checkGUID	A standard GUID, for example, 9500A4AE-EE9E-4383-87F2-BAD7DDC26C59 can be generated using the “guidgen” Windows utility, downloaded from a web site, or by other means. The GUID MUST be unique or the pluggable check will fail on upload to core. Once a check is uploaded with its unique GUID, you MUST NOT change the GUID or it will fail on re-upload with a "Database Unique Constraint Error" until you delete the original. Checks are uniquely identified by GUID, but for upload are solely identified by their name (of the zip file).
checkDefaultDescription	Displays in the SA Client description box. Honors hard carriage returns and HTML. With HTML, the HTML tags need to be converted with < and >.
checkRemediationDefaultDescription	Displays in the SA Client under the Remediation section of the check/rule.
checkGetScriptName	The file name for the get script, for example, getUsersWithoutPasswordExpiration.sh.

DTD Elements and Attributes, continued

Elements	Attributes
checkGetScriptType	The type of code determines the interpreter to be run. Get and set scripts may be types: SH, VBS, PY, BAT.
checkSetScriptName	The file name for the remediation script.
checkSetScriptType	The type of code determines interpreter to be run. Set (remediation) scripts may be of types SH, VBS, PY, BA.
checkVersion	This is based on SA and framework build number, such as 32b.0-1.0.
checkAllowRemediationOnFailure	Some scripts may fail during the get phase, but you may be able to correct this condition via the remediation script. This allows remediation to be performed even in the event of a script failure. For example, if the non-existence of a registry key is undefined, you can create and set it in your set code.
checkReturnType	Permissible values are EXITCODE, STRING, or NUMBER: EXITCODE — Standard script return via Wscript.Quit(), exit, return, etc. NUMBER — Audit and Remediation framework will grab from stdout and interpret it as numeric type. STRING — Audit and Remediation framework will grab from stdout and interpret as a string type.
checkTestIDs	List of test IDs.
checkTestID	Used to display the CIS, MSFT, NSA or other Policy standard nomenclature, for example, CIS-RHEL 8.4. This is a free form field, and displays in the SA Client, so be consistent in naming it to correspond with the TON Content.
checkPlatformTypes	List of valid platform types for a check.
checkPlatform	WINDOWS UNIX (or both as individual elements)

DTD Elements and Attributes, continued

Elements	Attributes
checkExclusivePlatforms	<p>List of exclusive platforms. Audit and Remediation currently separates things by Windows or Unix by default, but real world standards as well as limitations and/or differences across operating systems do not make this always desirable. You can limit Audit and Remediation to any platform specified by a platform ID retrieved from the spin.</p> <p>This parameter may refer to one of the supported operating systems listed in the SA Supported Platforms documentation.</p>
checkExclusivePlatform	Individual platform ID.
checkExcludePlatforms	List of excluded platforms. If the PlatformType claims UNIX, you can supply platform IDs to exclude from the UNIX set (all Linux + all Unixes).
checkExcludePlatform	Individual platform ID
checkCategory	<p>This is the SA Client Category that a check displays in. Currently, a check can only display in a single category. If a category does not exist, it will be created upon upload. The following standard categories for existing checks should be used where possible:</p> <ul style="list-style-type: none"> Event Logging File System Operating System Operating System Domain Controller (sub-category) Operating System Network (sub-category) Registry Services Users and Groups
<pre>checkGetArgument (checkGetArgumentType, checkGetArgumentDefaultLabel, checkGetArgumentDefaultDescription, checkGetArgumentDefaultValue?, checkGetArgumentValidValues?)></pre>	Specifies parameters to the get script.
checkGetArgumentType	NUMBER STRING

DTD Elements and Attributes, continued

Elements	Attributes
checkGetArgumentDefaultLabel	SA Clienttag next to the input box or drop-down.
checkGetArgumentDefaultDescription	Hover text with further explanation.
checkGetArgumentDefaultValue	Default value for this get parameters.
checkGetArgumentValidValue (checkGetArgumentValidValueItem, checkGetArgumentValidValueDisplayName)	checkGetArgumentValidValueItem (#PCDATA)> checkGetArgumentValidValueDisplayName (#PCDATA)>
checkGetArgumentValidValues (checkGetArgumentValidValue+)	(Optional) Useful for limiting the parameters for example to 0/disable and 1/enable.
checkSetArguments (checkSetArgument+)	checkSetArgument (checkSetArgumentType, checkSetArgumentDefaultLabel, checkSetArgumentDefaultDescription, checkSetArgumentDefaultValue?, checkSetArgumentValidValues?) setArgument elements are identical to the GetArguments, but for the remediation/set script if it exists. The exception is: checkSetArgument populateFromRule — the set parameter default should or should not populate itself from the rule data, versus if any default values were supplied in config.xml. Generally, this is always set to true.
checkSetArgumentType	NUMBER STRING
checkSetArgumentDefaultLabel	SA Clienttag next to the input box or drop-down.
checkSetArgumentDefaultDescription	Hover text with further explanation.
checkSetArgumentDefaultValue	Default value for this set parameter.
checkSetArgumentValidValues (checkSetArgumentValidValue+)	
checkSetArgumentValidValue (checkSetArgumentValidValue Item,	checkSetArgumentValidValueItem (#PCDATA)>

DTD Elements and Attributes, continued

Elements	Attributes
<pre>checkSetArgumentValidValue DisplayName) ></pre>	<pre>checkSetArgumentValidValueDisplayName (#PCDATA) > checkSetArgumentValidValueItem (#PCDATA) > checkSetArgumentValidValueDisplayName (#PCDATA) ></pre>
<pre>checkSetArgumentValidValue Item</pre>	(Optional) Useful for limiting the parameters for example to 0/disable and 1/enable.
<pre>checkSetArgumentValidValueDisplayName</pre>	
<pre><!ELEMENT checkComparisonDefaults (checkComparisonDefaultOperator?, checkComparisonDefaultValues) ></pre>	<pre>checkComparisonDefaultOperator not — negation of operator specified, TRUE FALSE checkComparisonDefaultOperator caseInsensitive — only valid for STRING types.</pre>
<pre><!ELEMENT checkComparisonDefaultOperator (#PCDATA) ></pre>	List of default values for comparator. Useful for field or development outside the TON build framework.
<pre>checkComparisonDefaultValues (checkComparisonDefaultValue+)</pre>	<pre>checkComparisonDefaultValue (checkComparisonDefaultValueItem, checkComparisonDefaultValueDisplayNam e).</pre>
<pre>checkComparisonDefaultValueItem</pre>	Value for default, passed to code.
<pre>checkComparisonDefaultValueDisplayNam e</pre>	Display name for the value, seen in the SA Client.
<pre>checkCompareValidValues (checkCompareValidValue+) > checkCompareValidValue (checkCompareValidValueItem, checkCompareValidValueDisplayName) > checkCompareValidValueItem (#PCDATA) > checkCompareValidValueDisplayName (#PCDATA) ></pre>	
<pre>checkSuccessExitCodes (checkSuccessExitCode+) checkSuccessExitCode (checkSuccessExitCodeValue,</pre>	For a checkReturnType of EXITCODE, you must define the valid values for proper script operation, which generally include both the compliant and non-compliant expected

DTD Elements and Attributes, continued

Elements	Attributes
checkSuccessExitCodeDefaultDescription, checkSuccessExitCodeDefaultDisplayName)>	values. Anything returned other than a value specified here will be seen as a script failure, which is shown differently in the SA Client, as well as in reporting.
checkSuccessExitCodeValue	Value for script completion, for example, 0 (for <i>disabled</i> typically).
checkSuccessExitCodeDefaultDescription	Hover text for the DisplayName/Value.
checkSuccessExitCodeDefaultDisplayName	Value or text shown to user for this value, for example, Disabled.

Search filter syntax

Filter grammar

A search filter is a parameter for methods such as `findServerRefs`. The expression in a search filter enables you to get references to SA objects (such as servers and folders) according to the values of the object attributes. The formal syntax for a search filter follows:

```

<filter> ::= (<expression-junction>)+
<expression-junction> ::= <expression-list-open> <junction>
(<expression>)+ <expression-list-close>
<expression> ::= <expression-open> <attribute> <general-delimiter> <operator>
<general-delimiter> <value-list> <expression-close>

<attribute> ::= <resource_field>
<vo_member> ::= <text>
<resource_field> ::= <text>
<value-list> ::= (<double-quote> <text> <double-quote>)* |
(<number>)*
<text> ::= [a-z] [A-Z] [0-9]
<number> ::= [0-9] [.]

<junction> ::= <union-junction> | <intersect-junction>
<union-junction> ::= '|'
<intersect-junction> ::= '&'
<expression-list-open> ::= '('
<expression-list-close> ::= ')'

```

```
<expression-open> ::= '(' | '{'  
<expression-close> ::= '(' | '}'  
<general-delimiter> ::= <whitespace>  
<whitespace> ::= ' '  
<double-quote> ::= '\"'  
<escape-character> ::= '\\'  
<operator> ::= <equal_to> |...| <contains_or_above>
```

Valid operators for the preceding line:

```
<equal_to> ::= '=' | 'EQUAL_TO'  
<not_equal_to> ::= '!=' | '<>' | 'NOT_EQUAL_TO'  
<in> ::= '=' | 'IN'  
<not_in> ::= '!=' | '<>' | 'NOT_IN'  
<greater_than> ::= '>' | 'GREATER_THAN'  
<less_than> ::= '<' | 'LESS_THAN'  
<greater_than_or_equal> ::= '>=' | 'GREATER_THAN_OR_EQUAL'  
<less_than_or_equal> ::= '<=' | 'LESS_THAN_OR_EQUAL'  
<begins_with> ::= '=*' | 'BEGINS_WITH'  
<ends_with> ::= '*=' | 'ENDS_WITH'  
<contains> ::= '*=*' | 'CONTAINS'  
<not_contains> ::= '*<>*' | 'NOT_CONTAINS'  
<in_or_below> ::= 'IN_OR_BELOW'  
<in_or_above> ::= 'IN_OR_ABOVE'  
<between> ::= 'BETWEEN'  
<not_between> ::= 'NOT_BETWEEN'  
<not_begins_with> ::= 'NOT_BEGINS_WITH'  
<not_ends_with> ::= 'NOT_ENDS_WITH'  
<is_today> ::= 'IS_TODAY'  
<is_not_today> ::= 'IS_NOT_TODAY'  
<within_last_days> ::= 'WITHIN_LAST_DAYS'  
<within_last_months> ::= 'WITHIN_LAST_MONTHS'  
<within_next_days> ::= 'WITHIN_NEXT_DAYS'  
<within_next_months> ::= 'WITHIN_NEXT_MONTHS'  
<not_within_last_days> ::= 'NOT_WITHIN_LAST_DAYS'  
<not_within_last_months> ::= 'NOT_WITHIN_LAST_MONTHS'  
<not_within_next_days> ::= 'NOT_WITHIN_NEXT_DAYS'  
<not_within_next_months> ::= 'NOT_WITHIN_NEXT_MONTHS'  
<contains_or_below> ::= 'CONTAINS_OR_BELOW'  
<contains_or_above> ::= 'CONTAINS_OR_ABOVE'
```

Rebuilding the Apache HTTP server and PHP

This topic describes how to rebuild the Apache HTTP server and PHP and replace them in SA. SA includes an Apache HTTP server and PHP so this information is only needed if you need to use a different version of the Apache HTTP server or if you need to compile extra libraries or modules into PHP.

SA uses the Apache HTTP server and PHP for web Automation Platform Extensions (APX). For more information, see ["Automation Platform Extensions \(APX\)" on page 82](#).

Extending the APX HTTP environment

This section describes how you can extend the APX HTTP environment by rebuilding the Apache HTTP server and PHP.

Note: You must perform these tasks after all core upgrades.

If you have a Multimaster Mesh, these tasks must be performed on each slice in all cores. For more information on slice component bundles, see the Server Automation Administration Guide on the HPE SSO portal.

Rebuilding PHP

1. Perform the following tasks to rebuild PHP.

Download the PHP source from <http://www.php.net/>.

2. Put the source in a directory on the server where apxproxy is installed, typically under `/opt/opsware/apxproxy`.
3. Enter the following commands, replacing the version number if you downloaded a different version of PHP.

```
mkdir /build ; cp php-4.4.8.tar.gz /build; cd /build
gzip -dc php-4.4.8.tar.gz | tar xvf -
cd php-4.4.8
./configure --prefix=/opt/opsware/apxphp
```

```
--with-pear=/opt/opsware/apxphp/lib/pear  
--with-config-file-path=/opt/opsware/apxphp/lib  
--with-apxs2=/opt/opsware/apxhttpd/bin/apxs <any other options you>  
make clean  
make
```

4. Backup your old copy of libphp4.so:

```
cp /opt/opsware/apxhttpd/modules/libphp4.so  
/opt/opsware/apxhttpd/modules/libphp4.so.backup
```

5. Copy the new libphp4.so file to the apxhttps directory:

```
cp libs/libphp4.so /opt/opsware/apxhttpd/modules/libphp4.so
```

6. Ensure that the complete reference library exists in the tool.list:

```
ldd ./libs/libphp4.so
```

For each entry in the output ensure that the file exists in
/etc/opt/opsware/ogfs/tool.list.

If an entry does not exist, add it.

7. Backup the apxphp folder:

```
mv /opt/opsware/apxphp /opt/opsware/apxphp.orig
```

8. Install PHP:

```
make install
```

9. Reload and relink the OGFS to make sure anything you added to
/etc/opt/opsware/ogfs/tools.list shows up in the OGFS:

```
/opt/opsware/ogfs/tools/relink && /opt/opsware/ogfs/  
tools/reload
```

10. Restart apxproxy:

```
/etc/opt/opsware/startup/apxproxy restart
```

Rebuilding Apache

Perform the following tasks to rebuild the Apache HTTP server.

1. Download the source code for the Apache HTTP server from <http://httpd.apache.org/>.
2. Put the source in a directory on the server that hosts the slice component bundle. For more information on slice component bundles, see the Server Automation Administration Guide on the HPE SSO portal.
3. Enter the following commands, replacing the version number if you downloaded a different version of httpd.

```
mkdir /build; cp httpd-2.2.8.tar.gz /build; cd /build
gzip -dc httpd-2.2.8.tar.gz | tar xf -
cd httpd-2.2.8
./configure --prefix=/opt/opsware/apxhttpd <any other options you want>.
```

SA currently uses:

```
--enable-mods-shared="actions alias auth_basic auth_digest authn_file authz_
user cgi deflate dir dumpio env expires headers ident logio log_config mime
negotiation rewrite userdir vhost_alias imagemap status"
--disable-dav
--with-port=8021
--with-expat=builtin
--without-pgsql
```

(*On SunOS only*) Enter this command:

```
perl -pi -e 's/#define HAVE_GETADDRINFO 1/#undef HAVE_GETADDRINFO/g'
./srclib/apr/include/arch/unix/apr_private.h
make
```

4. Make a backup of the apxhttpd directory:

```
mv /opt/opsware/apxhttpd /opt/opsware/apxhttpd.orig
```

5. Install Apache:

```
make install
```

6. Reload and relink the new files into the OGFS:

```
/opt/opsware/ogfs/tools/relink && /opt/opsware/ogfs/tools/reload
```

7. The HTTPD and the .so files in the modules directory may reference external libraries. These libraries must be visible (or winked in) to the OGFS.

Log in to the OGFS and run LDD on `/opt/opsware/apxhttpd/bin/httpd` and any `.so` file in `/opt/opsware/apxhttpd/modules` and ensure that all the files listed there exist in the OGFS. If they do not, add the files to `/etc/opt/opsware/ogfs/tool.list` (outside the OGFS) and then re-run step 6 until all files are available to `/opt/opsware/apxhttpd/bin/httpd`.

8. You must now rebuild PHP. See ["Rebuilding PHP" on page 195](#).

Application Configuration

The section describes how to manage and create a template of a configuration file:

- ["Managing XML configuration files" below](#)
- ["CML primer" on page 210](#)
- ["CML Reference" on page 223](#)
- ["XML Tutorial 1 - Creating a non-DTD XML configuration template" on page 273](#)
- ["XML Tutorial 2 - Creating an XML-DTD configuration template" on page 280](#)
- ["CML Tutorial 1 - Creating an Application Configuration for a simple web app server" on page 289](#)
- ["CML Tutorial 2 - Creating a template of a web server configuration file" on page 300](#)

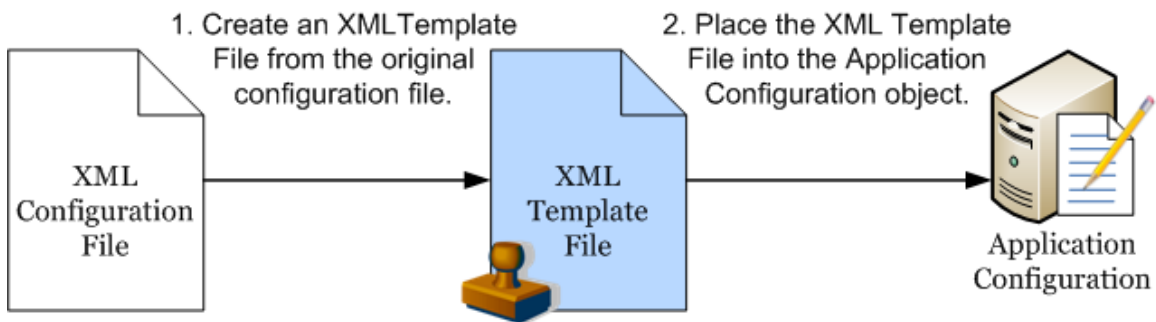
Managing XML configuration files

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 section 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.

Configuration files



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.

Also see the following examples:

- ["XML Tutorial 1 - Creating a non-DTD XML configuration template" on page 273.](#)
- ["XML Tutorial 2 - Creating an XML-DTD configuration template" on page 280.](#)

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 208](#).

For information on setting the parser syntax to XML for a configuration template, see [Creating a configuration template](#).

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 208](#).

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-PUBLIC-ID: The public ID of the XML file.

Customize 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.

```
<!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 password
ACM-DESCRIPTION = The db-password element specifies the password used to connect to
the database.
-->
```

Add 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.

Add 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/mysql12.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.
-->

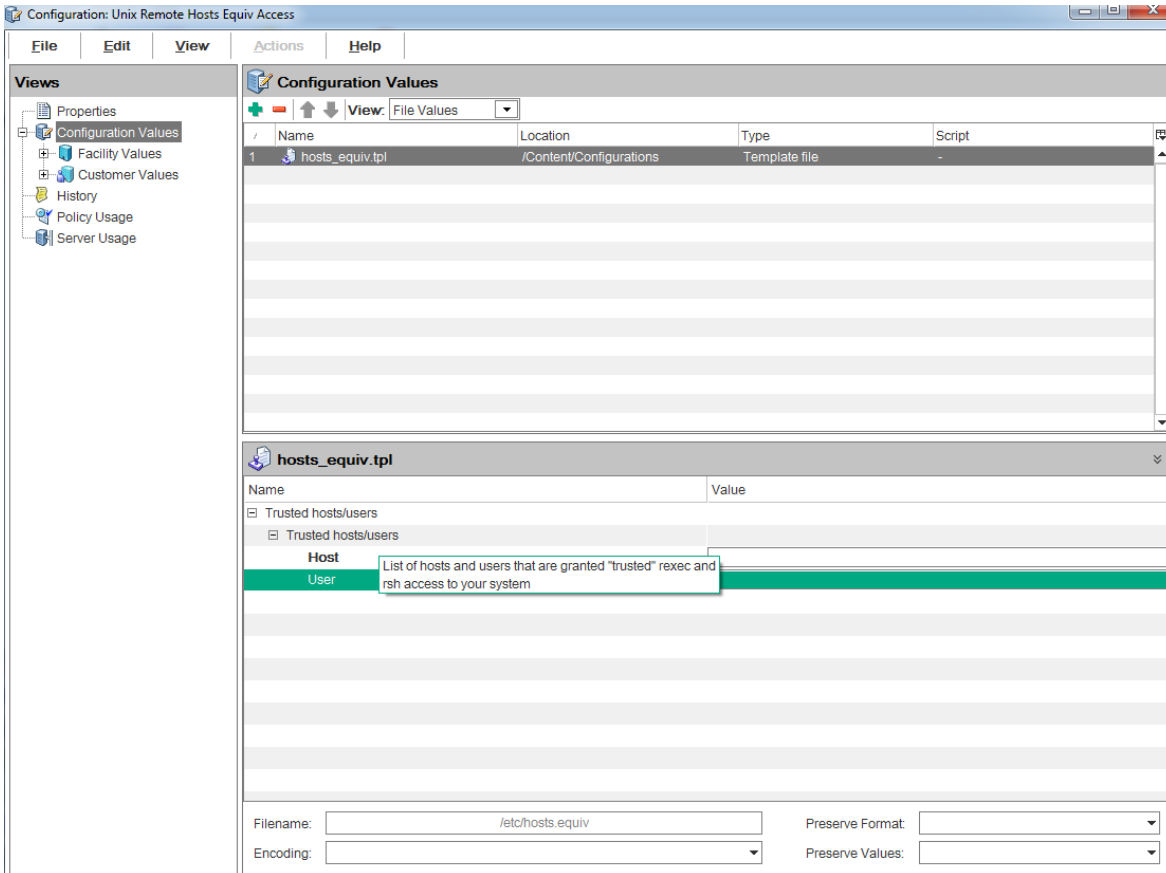
<!--
ACM-ELEMENT = db-user
ACM-PRINTABLE = database user
ACM-DESCRIPTION = The db-user element specifies the user identification used to
connect to the database.
-->

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

Customize 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 the following figure.

Custom element names and mouse-over text



XML configuration template settings

The following table 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.

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>	namespace identifies a location where XML elements are stored in the database.

XML and XML-DTD template settings, continued

Attribute	Description
Required; no default value.	
<p>ACM-TIMEOUT=<integer> Optional; (default value is 0)</p>	<p>timeout represents the number of minutes that are added to the application configuration's total timeout.</p> <p>A valid timeout is any integer from 0-999 inclusive.</p> <p>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.</p> <p>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.</p>
<p>ACM-DOCTYPE = <string> Required; no default value. XML-DTD templates only.</p>	<p>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.</p>
<p>ACM-DOCTYPE-SYSTEM-ID = <string> Required; no default value. XML-DTD templates only.</p>	<p>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.</p>
<p>ACM-DOCTYPE-PUBLIC-ID = <string> Required; no default value. XML-DTD templates only.</p>	<p>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.</p>
<p>ACM-ELEMENT=<element name> Optional XML-DTD templates only.</p>	<p>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.</p>
<p>ACM-ATTRIBUTE=<attribute name> Optional XML-DTD templates only.</p>	<p>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.</p>
<p>ACM-PRINTABLE=<printable></p>	<p>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.</p>

XML and XML-DTD template settings, continued

Attribute	Description
Optional XML-DTD templates only.	
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.

CML primer

This section introduces the **Configuration Modeling Language**, CML. For complete details on CML see ["CML Reference" on page 223](#). See also ["CML Tutorial 1 - Creating an Application Configuration for a simple web app server" on page 289](#) and ["CML Tutorial 2 - Creating a template of a web server configuration file" on page 300](#).

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 223](#).

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 ["Namespace tag" below](#).
- `filename-key` defines the key where the target configuration file name is stored in the SA database. See ["Filename-key tag" on the next page](#).
- `filename-default` specifies the default name of the target configuration file. See ["Filename-default tag" on page 213](#).

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 223](#). For details on these three required tags, see ["CML global option attributes" on page 250](#).

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 namespace value must be absolute. Subsequent values can have relative or absolute path names.

- 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"`.

- 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 `"/example/namespace/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@`

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@
```

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 241](#). 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:

Specifying integer ranges

Range condition	Symbols to use
Greater than	<code>n<</code>
Greater than or equal	<code>n<=</code>
Less than	<code><n</code>
Less than or equal	<code><=n</code>
Equal	<code>=n</code>

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@
```

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.*"@
```

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/ipaddress = 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.

<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.

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.

<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 215](#). 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@
```

[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 215](#).)

- 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@
```

[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";"@
```

[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 216](#).

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@  
@.@;
```

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.

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@  
@.@;
```

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.

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@
```

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 [Set Values in the Value Set Editor](#). See also ["The @!full-template and @!partial-template attributes"](#) on page 251.

CML Reference

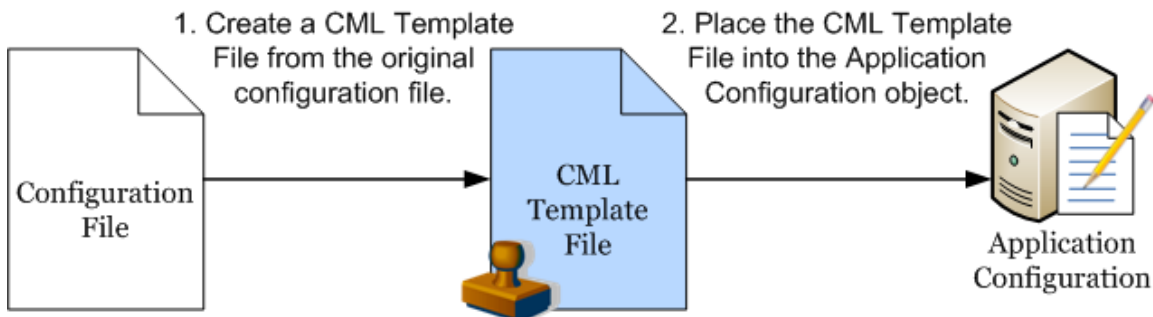
The Configuration Modeling Language (CML) is used to create a **template** of a **configuration file** so it can be managed from SA. A CML template is a separate file you create that models the format of the configuration file so the variable values in the configuration file can be set to different values 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](#).

CML Template



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 199](#).

Configuration templates

A configuration template is a “templated” 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](#).

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}@
```

Note:

Neither whitespace nor '@' can appear inside a CML tag. The '@' symbol can be escaped by prepending it with another '@'.

The following rules apply to all CML tag:

- All CML tags start and ends with the @ symbol.
- 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 228](#).
 - 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 ["Define the Namespace with the @!namespace CML Tag" on the next page](#).
- **@!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 the next page](#).
- **@!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 the next page.

Define 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 @##" on the next page
- "Replace Tag: @" on page 230
- "Instruction Tags: @!" on page 231

- "Block (or Group) Tag: @[@...@]@" on page 232
- "Loop Tag: @*" on page 235
- "Loop Target Tag: @." on page 237
- "Conditional Tag: @?" on page 238
- "DTD Tag: @~" on page 239

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 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 241](#).

- **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 247](#).
- **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 250](#) and ["CML regular option attributes" on page 252](#).

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 250](#) and ["CML regular option attributes" on page 252](#).

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 whitespace, 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 250](#) and ["CML regular option attributes" on page 252](#).

Example 1

The following instruction tag tells the CML parser that whitespace 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 (with literal strings quoted):

```
"@" [{level}] "[" [ ";" {option}[ ";" {option}]...] "@" {CML statements} "@"  
[ {level} "]"@"
```

Multiple line syntax for the group tag is as follows:

```
"@" {level} "[" [ ";" {option}[ ";" {option}]...] "@"  
{CML statements}  
"@" {level} "]"@"
```

```
@[{level}][ ;{option};{option}]...@ {set of CML tags} @[{level}]]@
```

```
@[{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.

Note:

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 250](#) and ["CML regular option attributes" on page 252](#).

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 Ur1Scan.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 237.

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 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 loop 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 241](#).

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 247](#).
 - **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 250](#) and ["CML regular option attributes" on page 252](#).

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 235.

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 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 ["Use DTD tags in CML" on page 266](#). For XML templates, see ["Customize XML DTD element display" on page 204](#).

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 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 the next page](#) 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 247](#).
- **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 250](#) and ["CML regular option attributes" on page 252](#).

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

CML supports the following two conventional forms for representing IPv6 addresses as text strings:

`x:x:x:x:x:x`, in which “x” represent one to four hexadecimal digits of the eight 16-bit pieces of the IPv6 address. For example:

```
ABCD:EF01:2345:6789:ABCD:EF01:2345:6789
```

“::” in an IPv6 address indicates one or more groups of 16 bits of zeros. The “::” can appear only once in an address. The “:” can also be used to compress leading or trailing zeros in an address. For example:

```
2001:DB8:0:0:8:800:200C:417A becomes 2001:DB8::8:800:200C:417A
```

```
0:0:0:0:0:0:1 becomes ::1
```

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}[;  
{option}]...]]@  
@[{level}]{tag-type}[[{source}]];[[{type}]];{range},{range};{option}[;  
{option}]...]]@
```

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 neither whitespace nor '@' appear inside a CML 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 , $\leq n$ 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, $0\leq\&<256$.

Whitespace is not significant when specifying ranges.

Note:

The current CML parser requires that neither whitespace nor '@' appear inside a CML tag.

" - String literal specifier

Syntax

```
@[ $\{\text{level}\}$ ]{tag-type}[[ $\{\text{source}\}$ ][[ $\{\text{type}\}$ ][["string"]]; $\{\text{option}\}$ ; $\{\text{option}\}$ ...]]@
```

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 $\backslash n$, a tab character is represented by $\backslash t$, and a literal backslash is represented by $\backslash \backslash$. 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 neither whitespace nor '@' appear inside a CML tag.

r" - Regular expression specifier

Syntax

```
@[ $\{\text{level}\}$ ]{tag-type}[[ $\{\text{source}\}$ ][[ $\{\text{type}\}$ ][["r{regular expression}"]; $\{\text{option}\}$ ; $\{\text{option}\}$ ...]]@
```

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 neither whitespace nor '@' appear inside a CML 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 can be written as a relative path and does not need to begin with a slash (/).

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.

The @!unix-newlines and @!windows-newlines attributes

Syntax

```
@!unix-newlines@
```

```
@!windows-newlines@
```

unix-newlines is the default behavior.

Description

unix-newlines is the default behavior and indicates that the configuration file generated by this template will have unix-style newlines (ASCII Line Feed character).

windows-newlines indicates that the configuration file generated by this template will have windows-style newlines (ASCII Carriage Return + Line Feed combination).

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[;  
{option}]]...]]]@
```

```
@[{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[;  
{option}]...]]@
```

```
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;ordered-elements[;  
{option}]...]]@
```

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[;  
{option}]...]]]@
```

```
@[{level}]{tag-type}[[{source}]][;{type}]][;{range}][;strict-whitespace[;  
{option}]...]]]@
```

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 whitespace 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[;  
{option}]...]]]@
```

```
@[{level}]{tag-type}[[{source}]][[{type}]][[{range}]][[optional-whitespace[;  
{option}]...]]@
```

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-error[;  
{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-keywords[;  
{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 reluctant attribute

Instruction tag syntax

```
@!reluctant@
```

Option attribute syntax

```
@[{level}]{tag-type}[{source}][;{type}][;{range}][;reluctant[;{option}...]]]@
```

Description

reluctant specifies that a specific loop or sequence will try to match as few elements as possible from the configuration file. This is not the default behavior of loops and sequences.

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-values[;  
{option}]]...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}]][;{range}]][show-lines-without-values[;  
{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-values[;  
{option}]]...]]]@
```

```
@[{level}]{tag-type}[[{source}]][;{type}]][;{range}]][;show-groups-without-values[;  
{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[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}]][;{range}]][;sequence-replace[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}]][;{range}]][;sequence-prepend[;  
{option}...]]]@
```

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[;  
{option}...]]]@  
@[{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={string}];  
{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={string}];  
{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 delimiter attribute

```
whitespace-delimited  
comma-delimited  
semicolon-delimited  
tab-delimited  
quote-delimited  
delimiter
```

Instruction tag syntax

```
@!whitespace-delimited@  
@!comma-delimited@  
@!semicolon-delimited@  
@!tab-delimited@  
@!quote-delimited@  
@!delimiter={string}@
```

whitespace-delimited is the default behavior

Option attribute syntax

```
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;whitespace-delimited[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;comma-delimited[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;semicolon-delimited[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;tab-delimited[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;quote-delimited[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;delimiter={string}][;  
{option}...]]]@
```

whitespace-delimited is the default behavior.

Description

`delimiter` sets the default delimiter character. If not explicitly specified, `sequence-delimiter` and `field-delimiter` will inherit this value.

Valid for `replace` and `sequence` 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-semicolon[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;line-comment-is-tab[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;line-comment-is-whitespace[;  
{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-is-semicolon  
sequence-delimiter-is-tab  
sequence-delimiter-is-whitespace  
sequence-delimiter-is-quote  
sequence-delimiter
```

Instruction tag syntax

```
@!sequence-delimiter-is-comma@  
@!sequence-delimiter-is-semicolon@  
@!sequence-delimiter-is-tab@  
@!sequence-delimiter-is-whitespace@  
@!sequence-delimiter-is-quote@  
@!sequence-delimiter={string}@
```

By default, `sequence-delimiter` uses the value of `delimiter`. The default for the latter is `whitespace-delimited`.

Option attribute syntax

```
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;sequence-delimiter-is-comma[;  
{option}...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;sequence-delimiter-
```

```
issemicolon[;{option}...]]]]@  
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-delimiter-is-tab[;  
{option}...]]]]@  
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-delimiter-is-  
whitespace[;{option}...]]]]@  
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-delimiter-is-quote[;  
{option}...]]]]@  
@[{level}]{tag-type}[[{source}][;[{type}][;[{range}][;sequence-delimiter={string}[;  
{option}...]]]]@
```

By default, `sequence-delimiter` uses the value of `delimiter`. The default for the latter is `whitespace-delimited`.

Description

`sequence-delimiter` sets the character that separates items within a sequence. By default, `sequence-delimiter` uses the value of `delimiter`. The default for the latter is `whitespace-delimited`.

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-is-quote  
field-delimiter
```

Instruction tag syntax

```
@!field-delimiter-is-comma@  
@!field-delimiter-is-semicolon@  
@!field-delimiter-is-tab@  
@!field-delimiter-is-whitespace@  
@!field-delimiter-is-quote@  
@!field-delimiter={string}@
```

By default, `field-delimiter` uses the value of `delimiter`. The default for the latter is `whitespace-delimited`.

Option attribute syntax

```
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;field-delimiter-is-comma[;  
{option}]...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;field-delimiter-is-semicolon  
[;{option}]...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;field-delimiter-is-tab[;  
{option}]...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;field-delimiter-is-whitespace  
[;{option}]...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;field-delimiter-is-quote[;  
{option}]...]]]@  
@[{level}]{tag-type}[[{source}]][;{type}][;{range}][;field-delimiter={string}[;  
{option}]...]]]@
```

By default, `field-delimiter` uses the value of `delimiter`. The default for the latter is `whitespace-delimited`.

Description

`field-delimiter` sets a character that will be used to terminate parsing for a replace element value. By default, `field-delimiter` uses the value of `delimiter`. The default for the latter is `whitespace-delimited`.

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={string}[;  
{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.

Use 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.

Example of DTD tags

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:

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 " below:** Sequence values from more specific scopes completely replace those from less specific scopes. This occurs for both sequences of sets and lists.
- **"Sequence append " on the next page:** 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 *prepended* (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 name spaces.

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
```

Note:

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 prepend list merge mode (CML tag “sequence-prepend”) is used for sequences, the values at more general scopes are prepended (placed before) 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

The following table describes CML grammar illustrating several types of CML tags.

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
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

CML grammar, continued

CML tag/element	Description
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
option-nv	option-nv
option-name	string
option-value	string

XML Tutorial 1 - Creating 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 200.

Sample non-DTD XML `mysql.xml` file

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. Creating 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:
 - o **Name:** TM-MySQL
 - o **Description:** This is the template for the `mysql.xml` configuration file for the Travel Manager application.
 - o **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 the “Folder Permissions” section in the SA 10.51 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.
 - Select **File > Save**.
6. Keep the Template window open for the next task.

2. Adding 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.

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

2. Copy and paste the following XML into the Content pane:
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. Creating 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 the “Folder Permissions” section in the SA 10.51 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 **Configuration Values**.
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. Attaching the Application Configuration to a managed server

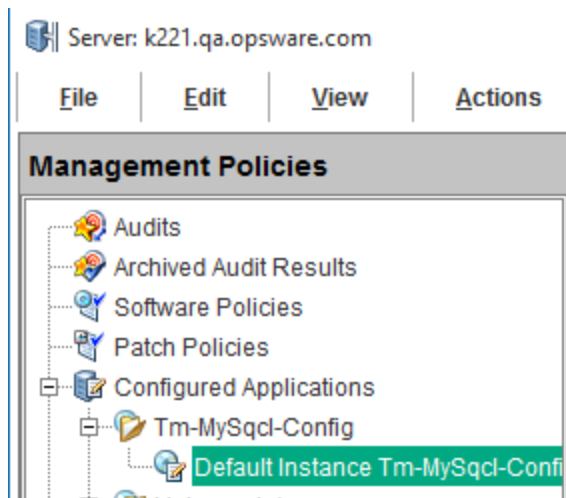
Now that you have created the configuration template and application configuration object, you need to attach the application configuration to the server where the Travel Manager application is installed and specify the path to where the `mysql.xml` file is stored on the managed server.

To attach the application configuration to a server:

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](#).
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.

The following figure 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.

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



Note:

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](#).

5. Configuring 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 mysql.xml file " on page 274](#) 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 [Set Fields in the Value Set Editor](#).
 - **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 [Set Fields in the Value Set Editor](#).
 - **Show Inherited Values:** Select this option to show the values in the value set and the inheritance level. When unchecked, only the values set at the current inheritance level are displayed. When checked, 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. Editing values and pushing 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:

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](#).
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.

XML Tutorial 2 - Creating an XML-DTD configuration template

This section 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 200](#).

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. Creating 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:

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. Adding 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 the [XML-DTD template figure](#) 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 205](#).

To add custom settings in the XML-DTD template:

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 password
ACM-DESCRIPTION = The db-password element specifies the password used to
connect to the database.
-->
```

2. Save and close the file.

3. Importing 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:

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. Creating 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. Attaching the Application Configuration to a managed server

In this task you will 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:

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.

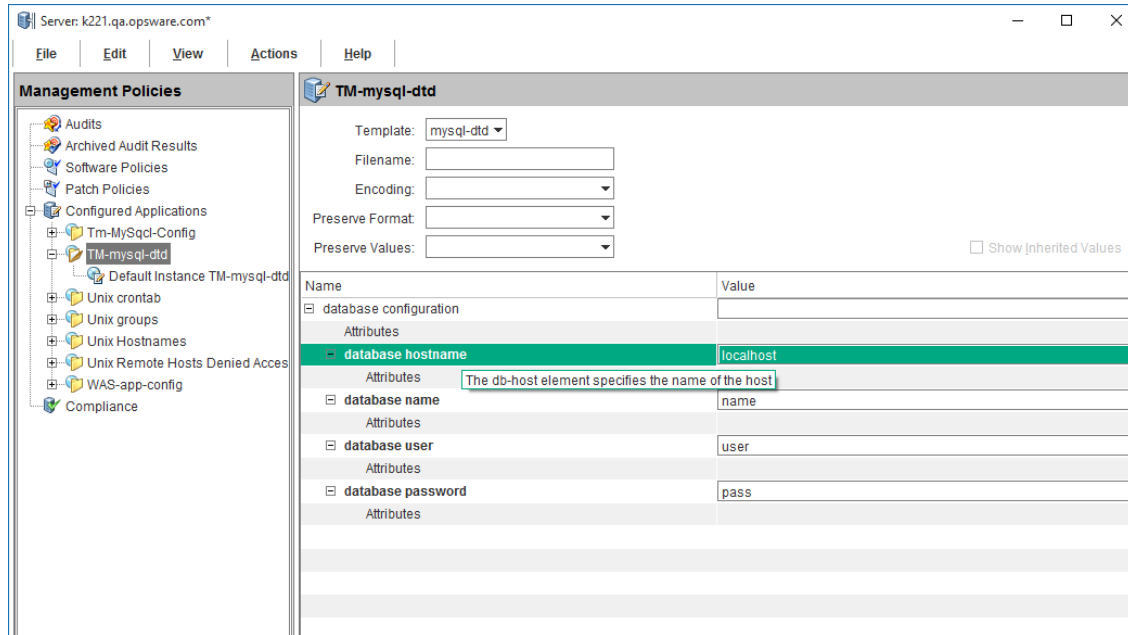
6. Importing values from the configuration file

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 280 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 281 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. The following figure shows the XML-DTD template with the server instance value set and the mouse-over text displayed from the ACM-DESCRIPTION element.

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



7. Leave the application configuration displayed for the next step.

7. Editing 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:

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.
2. Modify the password value under the Value column.
3. Select **Save Changes**.
4. Select **Preview** 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 **Push**. This displays the Push Configurations wizard.

7. Select **Start Job**. 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.

CML Tutorial 1 - Creating an Application Configuration for a simple web app server

This section 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. Determining 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. Creating 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.

Creating a template file and importing 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](#).

Creating 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. Creating 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. Adding 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. Attaching 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. Setting default values

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

Configuration value for two servers running the Web Application Server

Server: RHEL001	Server: RHEL008
size=1000	size=1000
dir=/tmp/WAS_001	dir=/tmp/WAS_008
primary=yes	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.

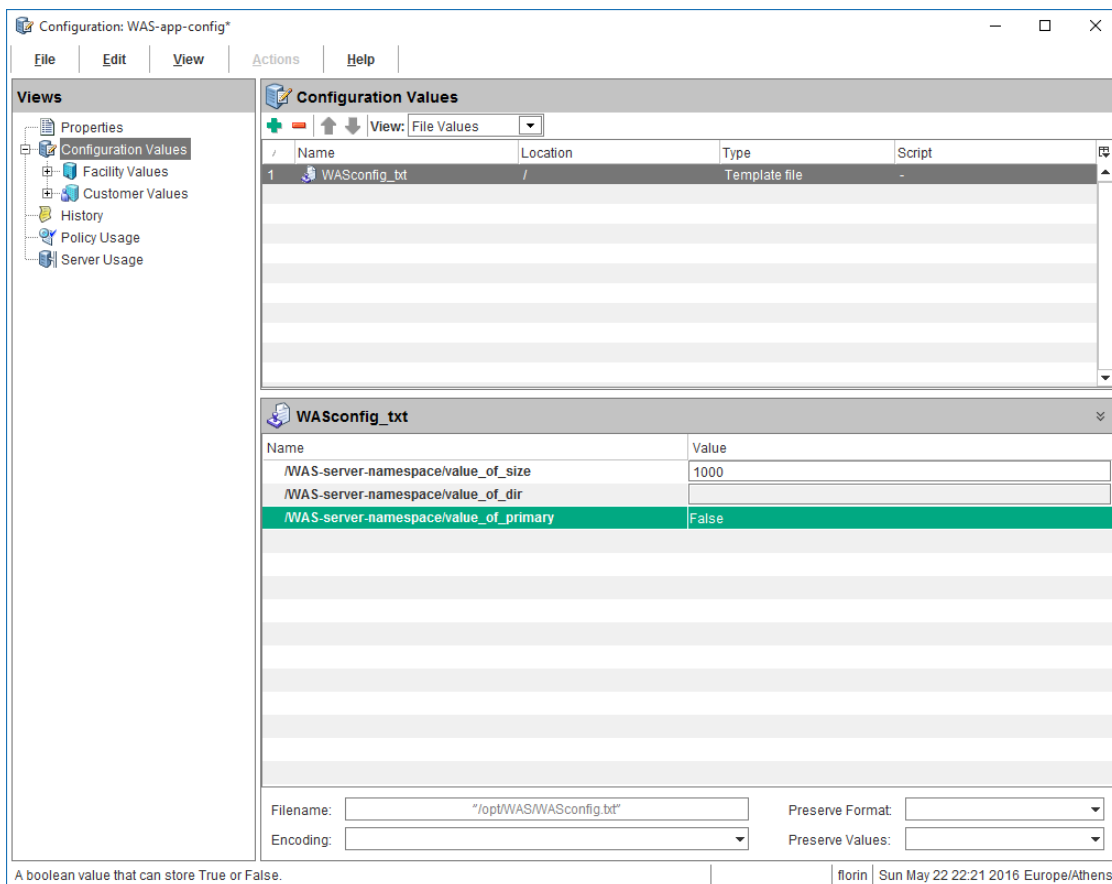
The following table shows which values will be set with default values and which will be set by individual servers:

Table 7: Application level default vValues 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.

Setting application level default values

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 “False” (case matters), as shown below. Do not set a default value for “value_of_dir” because each server will need to set this value.

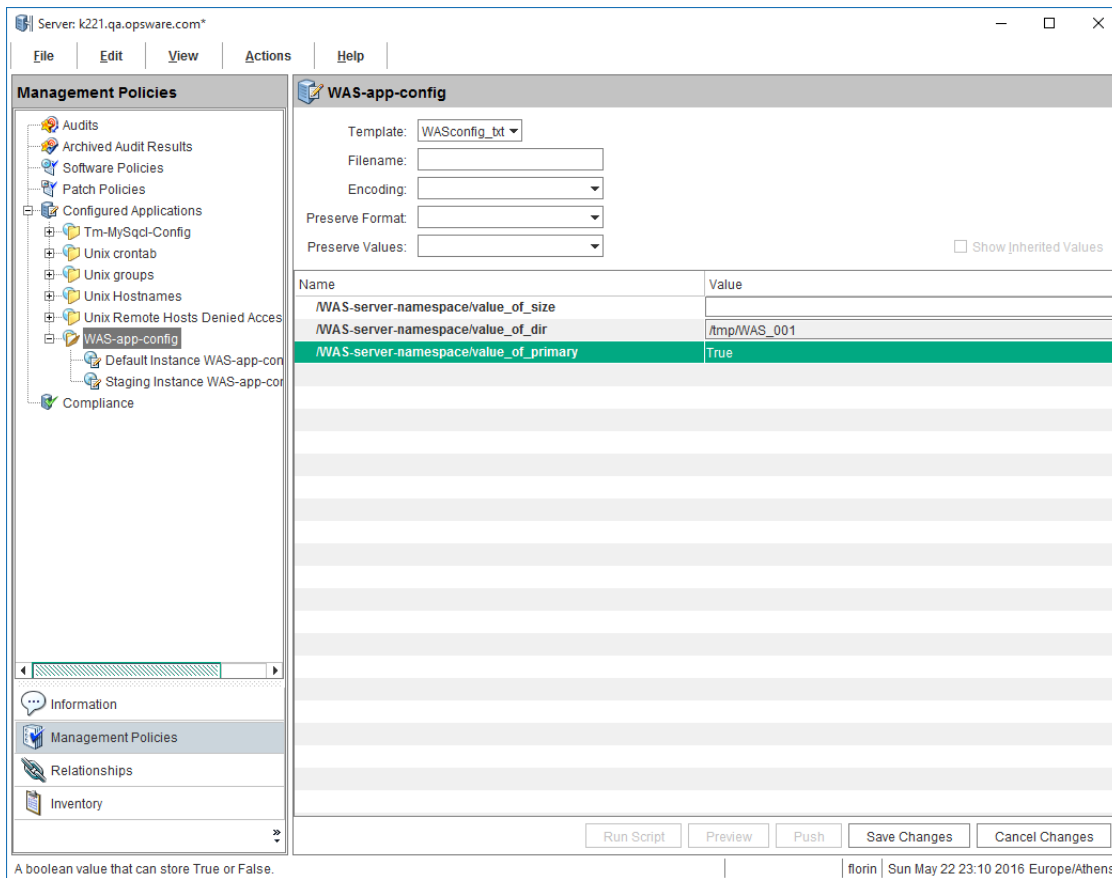


6. Select **File > Save** to save your application level default values.
7. Select **File > Close**.

Setting server level default values for RHEL001

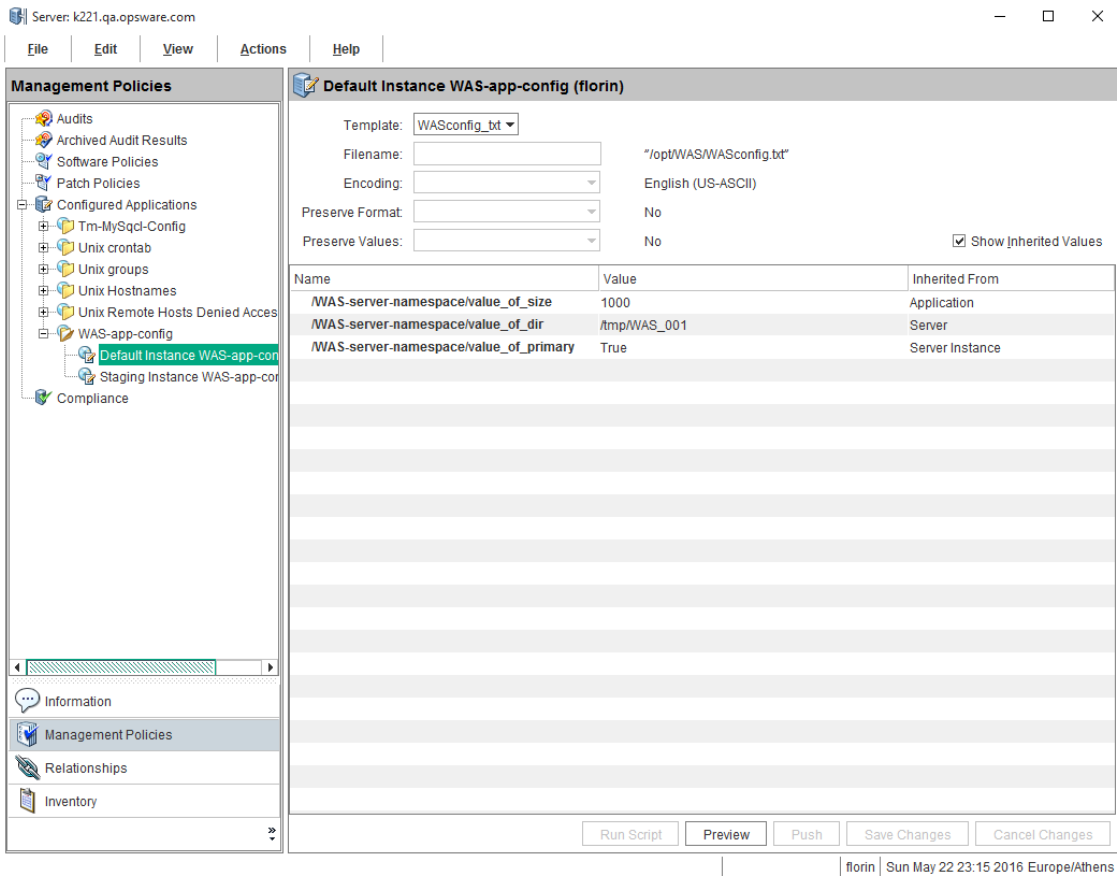
The server RHEL001 needs to set `dir=/tmp/WAS_001` and `primary=yes` at the server level. It does not need to set `size` because it can use the value set at the application level.

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.



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 “True” (case matters).
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.



Setting server level default values for RHEL008

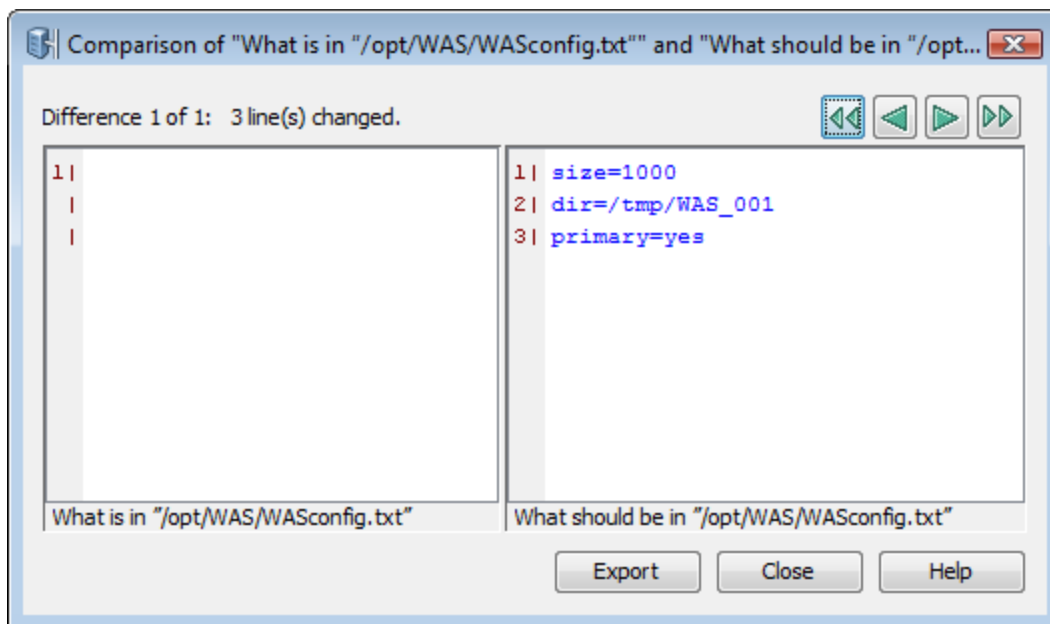
The server RHEL008 needs to set `dir=/tmp/WAS_008` at the server level. It does not need to set size or primary because it can use the values set at the application level.

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.

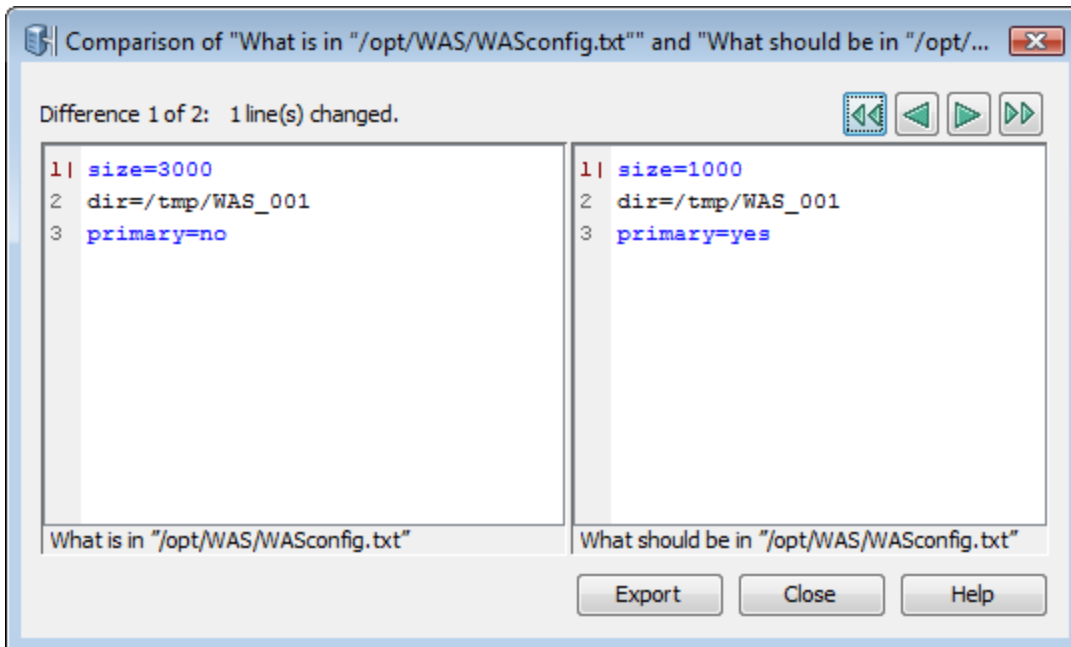
6. Set the server level default value for "value_of_dir" to "/tmp/WAS_008". Do not set a default value for "value_of_size" or "value_of_primary" because these values will be inherited from the application level.
7. Select **Save Changes** or the **File > Save** menu to save your **server level default 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. Comparing 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:



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:



8. Pushing configuration changes to the server

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 - Creating a template of a web server configuration file"](#) below.

CML Tutorial 2 - Creating 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 in ["Sample UrlScan.ini file" on page 315](#). The complete CML file is listed in ["Complete url_scan_ini.tpl CML template" on page 322](#).

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. Analyzing 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 in "[Sample UrlScan.ini file](#)" on page 315. 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. Creating 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. Creating 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 whitespace 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

The following figure explains setup instructions.

CML template setup instructions

CML Tag	Description
@!namespace=/security/@	<p>The <code>@!namespace</code> 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.</p> <p>In this example, the name space is <code>/security</code>. All value sets will be stored in this name space.</p>

CML template setup instructions, continued

CML Tag	Description
<code>@!filename-key="/files/urlscan_ini";filename-default="/c/urlscan.ini"@</code>	<p>The <code>@!filename-key</code> instruction defines the location in the name space where the file name will be 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 <code>@!namespace</code> instruction.</p> <p>In this example, the default file name will be stored in the database under the name space “/file/urlscan-ini”.</p> <p>The <code>@!filename-default</code> 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.</p> <p>In this example, when the configuration file is pushed to a managed server, it will be placed in <code>/c/urlscan.ini</code>.</p> <p>Note that the path names use only forward slashes.</p>
<code>@!optional-whitespace@</code>	<p>This instruction indicates that whitespace is optional between items in the configuration file. For example, either of the following entries would be valid if this option is set:</p> <p>Key = "value"</p> <p>Key="value"</p>
<code>@!boolean-yes-format="1";boolean-no-format="0"@</code>	<p>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.</p>
<code>@!line-comment-is-semicolon@</code>	<p>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.</p>
<code>@!unordered-lines@</code>	<p>Instructs the parser that the sections in the configuration file can be in any order. If you used <code>ordered-lines</code>, then the configuration file would have to conform to the order of the template.</p>

4. Defining 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.

The following table explains how to open two block levels for the [Options] section.

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.
[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

Marking up the start of the [Options] section, continued

CML tag	Description
	in the configuration file. That is, all the key-value pairs in the [Options] section can be in any order.

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. The following table explains the CML for each option.

Marked up key-value pairs from UrlScan.ini [Options] section

CML Tag	Description
AllowDotInPath = @allow_dot_in_path;boolean@	<p>Note: All of the key-value pairs use some variation of the following syntax (unless otherwise indicated):</p> <pre>string literal = @source;type@</pre> <p>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 set. The type is the type of data that will be stored in the value set.</p> <p>@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.</p> <p>You could also write this tag as follows: AllowDotInPath = @/security/allow_dot_in_path;boolean@ boolean</p> <p>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.</p>
AllowHighBitCharacters = @allow_high_bit_characters;boolean@	<p>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.</p> <p>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.</p>
AllowLateScanning = @allow_late_	Allows the IIS administrator to choose whether or not late scanning of a URL is acceptable. Defines a name space location to store the

Marked up key-value pairs from UrlScan.ini [Options] section, continued

CML Tag	Description
scanning;boolean@	value. <code>boolean</code> indicates this key can be 1 for true and 0 for false in the configuration file.
AlternateServerName = @alternate_servename@	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 <code>dir</code> 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@	<p>Syntax:</p> <p>string literal = @source;type;r"regular expression";option@</p> <p>reject response String literal that defines the path where the strings will be stored in the name space.</p> <p>string Indicates that the data type for the reject URL request is a string.</p> <p>r" This is a string range specifier that introduces a regular expression. In this case, a range of string literals.</p> <p>(HTTP_URLSCAN_STATUS_HEADER) (HTTP_URLSCAN_ORIGINAL_VERB) (HTTP_URLSCAN_ORIGINAL_URL)"</p> <p>The string literals (rejected URL responses) to be read by the parser: the status header, original verb, and original URL.</p> <p>optional Indicates that this value is optional. That is, the RejectResponseUrl option may be omitted from the UrlScan.ini file.</p>

Marked up key-value pairs from UrlScan.ini [Options] section, continued

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. Defining 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@
```

The following table explains how opening a new two level block closes the previous block.

Starting a New Block for the [AllowExtensions] Section

CML Tag	Description
@1 [;optional;ordered-lines@	<p>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.</p> <p>Note that you can explicitly close a block by using the close block tag. For example: @2]@</p> <p>[CML block tag that opens a new block.</p> <p>optional Indicates that this entire block is optional and not required to be in the configuration file.</p> <p>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.</p>
[AllowExtensions]	The literal string that names the section in the native configuration file.
@2[;unordered-lines@	<p>The number 2 sets the second level of the block.</p> <p>[CML block symbol that opens a new block.</p> <p>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.</p>

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@
.@.@
```

The following table explains the how the loop and loop target CML tags work:

Loop and loop target CML tags

CML tag	Description
@*allow_extension;unordered-string-set@	<p>Syntax</p> <p>@<level><tag type><name>;<data type>;<options>@</p> <p>The loop tag (*) will loop and read over the unordered string set listed in the [AllowExtensions] section.</p> <p>allow_extension String that defines the path where the strings will be stored in the name space.</p> <p>unordered-string-set Indicates that the list of strings do not have to be in any specific order.</p>
.@.@	<p>First (.)</p> <p>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.</p> <p>@.@</p> <p>Loop target tag (.) instructs the parser to read everything in this list that starts with a period character.</p>

3. Save the file.

6. Defining 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. Defining 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. Defining 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=":"@
@.@:
```

The following table describes the syntax of these two tags.

Loop and Loop Target Tags for the [DenyHeaders] Section

CML tag	Description
@*deny_header;unordered-string-set;;sequence-delimiter=":"@	<p>* Indicates a loop CML tag that will read through the list of strings.</p> <p>deny_header String literal that defines the path where the strings will be stored in name spacename space.</p> <p>unordered-string-set Indicates that the list of strings can be listed in any order.</p> <p>; The first semicolon separates the two sections of the tag.</p> <p>;</p>

Loop and Loop Target Tags for the [DenyHeaders] Section, continued

CML tag	Description
	The second semicolon allows you to enter the following colon (:) sequence delimiter without it being interpreted as a range. <code>sequence-delimiter=":"</code> 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. Defining 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 to 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@  
@.@
```


The following table describes the syntax of these tags.

TLoop and Loop Target Tags for the [DenyUrlSequence] Section

CML Tag	Description
<code>@*deny_url_sequence;unordered-string-set;;field-delimiter-is-eol@</code>	<p><code>*</code> Indicates a loop CML tag that reads through the list of strings.</p> <p><code>deny_url_sequence</code> String literal that defines the path where the string will be stored in the name space.</p> <p><code>unordered-string-set</code> Indicates that the list of strings can be listed in any order.</p> <p><code>;</code> The first semicolon separates the two sections of the tag.</p> <p><code>;</code> The second semicolon allows you to enter the following field delimiter without it being interpreted as a range.</p> <p><code>field-delimiter-is-eol</code> Instructs the parser to read the next entry up to the end of the line.</p>
<code>@.@</code>	<p>Loop target tag instructs the parser to store these values into the <code>deny_url_sequence</code> name space location. For example: <code>/security/deny_url_sequence</code>.</p>

3. Save the file.

10. Defining 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.

- Next, type the following CML replace tags to define the three key-value pairs found in the [RequestsLimits] section:

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

The following table describes the syntax of these tags.

Loop and loop target tags for the [DenyUrlSequence] section

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

Loop and loop target tags for the [DenyUrlSequence] section, continued

CML Tag	Description
@1]@	This level one block tag closes the block.

3. Save the file.

11. Placing the template in an Application Configuration

Now that you have created the CML template for UrlScan.ini and saved it as url_scan_ini.tpl, you are 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](#).
- Add the template to an Application Configuration. See [Add or Remove Templates from an Application Configuration](#).
- Attach the Application Configuration to a server. See [Attach an Application Configuration to a Server or Device Group](#).
- Test your template by making changes and pushing them to a server. See [Push Application Configurations](#).

These steps are described in "[CML Tutorial 1 - Creating an Application Configuration for a simple web app server](#)" on page 289.

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  
.png  
.png  
.png
```

[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]@
```

Send documentation feedback

If you have comments about this document, you can [contact the documentation team](#) by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Developer Guide (Server Automation 10.51)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to hpe_sa_docs@hpe.com.

We appreciate your feedback!