HP Server Automation

Ultimate Edition

Software Version: 10.23

Platform Developer Guide

Document Release Date: June 2016

Software Release Date: June 2016

Legal Notices

Warranty

The only warranties for HP 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. HP 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 HP 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 Notices

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

Intel® and Itanium® are trademarks of Intel Corporation in the U.S. and other countries.

Microsoft®, Windows®, Windows® XP are U.S. registered trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

UNIX® is a registered trademark of The Open Group.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

Support

Visit the HP Software Support Online website at:

https://softwaresupport.hp.com/

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

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support website to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

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

https://hpp12.passport.hp.com/hppcf/createuser.do

To find more information about access levels, go to:

https://softwaresupport.hp.com/web/softwaresupport/access-levels

Support Matrices

For complete support and compatibility information, see the support matrix for the relevant product release. All support matrices and product manuals are available here on the HP Software Support Online website:

https://softwaresupport.hp.com/group/softwaresupport/support-matrices

You can also download the *HP Server Automation Support and Compatibility Matrix* for this release from the HP Software Support Online Product Manuals website:

https://softwaresupport.hp.com/

This site requires that you register for an HP Passport and sign in. After signing in, click the **Search** button and begin filtering documentation and knowledge documents using the filter panel.

Documentation Updates

All the latest Server Automation product documentation for this release is available from the SA Documentation Library:

https://softwaresupport.hp.com/group/softwaresupport/search-result/-/facetsearch/document/KM00417675

Use the SA Documentation Library to access any of the guides, release notes, support matrices, and white papers relevant to this release or to download the full documentation set as a bundle. The SA Documentation Library is updated in each release and whenever the release notes are updated or a new white paper is introduced.

How to Find Information Resources

You can access the information resources for Server Automation using any of the following methods:

Method 1: Access the latest individual documents by title and version with the new SA Documentation Library

Method 2: Use the complete documentation set in a local directory with All Manuals Downloads

Method 3: Search for any HP product document in any supported release on the HP Software Documentation Portal

To access individual documents:

1. Go to the SA 10.x Documentation Library:

   https://softwaresupport.hp.com/group/softwaresupport/search-result/-/facetsearch/document/KM00417675

2. Log in using your HP Passport credentials.
3. Locate the document title and version that you want, and then click **go**.

To use the complete documentation set in a local directory:

1. To download the complete documentation set to a local directory:
   a. Go to the SA Documentation Library:

https://softwaresupport.hp.com/group/softwaresupport/search-result/-/facetsearch/document/KM00417675

b. Log in using your HP Passport credentials.

c. Locate the All Manuals Download title for the SA 10.1 version.

d. Click the **go** link to download the ZIP file to a local directory.

e. Unzip the file.

2. To locate a document in the local directory, use the Documentation Catalog (docCatalog.html), which provides an indexed portal to the downloaded documents in your local directory.

3. To search for a keyword across all documents in the documentation set:

a. Open any PDF document in the local directory.

b. Select **Edit > Advanced Search** (or Shift+Ctrl_F).

c. Select the All PDF Documents option and browse for the local directory.

d. Enter your keyword and click Search.

To find additional documents on the HP Software Documentation Search Portal:

Go to the HP Software Documentation Search Portal:

https://softwaresupport.hp.com/

This site requires that you register for an HP Passport and sign in. After signing in, click the **Search** button and begin filtering documentation and knowledge documents using the filter panel.

To register for an HP Passport ID, click the **Register** link on the HP Software Support Online login page.

You can also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details. See Documentation Change Notes for a list of any revisions.

Product Editions

There are two editions of Server Automation:

- Server Automation (SA) is the Ultimate Edition of Server Automation. For information about Server Automation, see the SA Release Notes and the SA User Guide: Server Automation.
- Server Automation Virtual Appliance (SAVA) is the Premium Edition of Server Automation. For more information about what SAVA includes, see the SAVA Release Notes and the SAVA at a Glance Guide.

Contents

1 Overview

Overview of the 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).

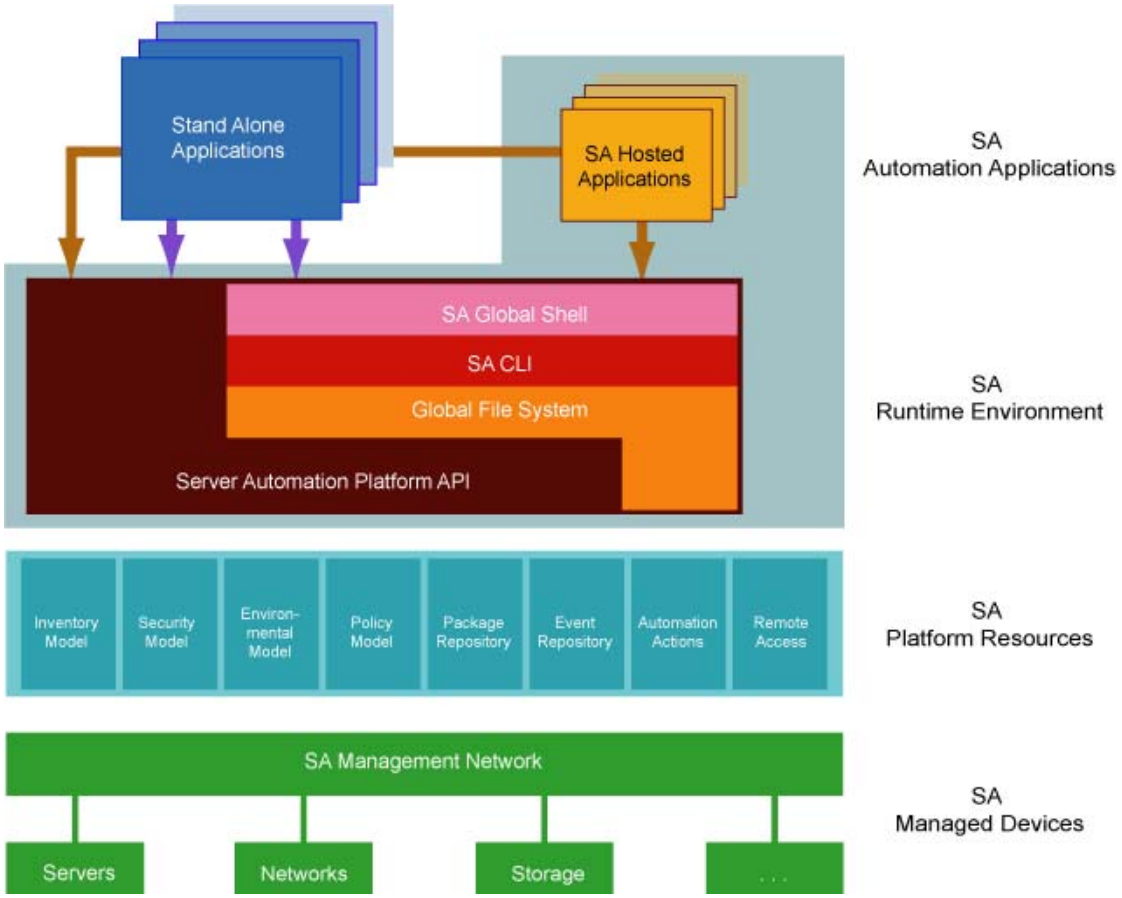
Using the Server Automation Platform, you can perform the following tasks:

- 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.

Components of the Server Automation Platform

Figure 1 shows the major elements of the Server Automation Platform.

figure 1 Server Automation Platform Components



As Figure 1 shows, the platform comprises the following five key elements. Each of these elements is discussed in more detail in subsequent sections.

- **Automation Applications:** The applications users write on top of the platform. These applications can either be SA-Hosted Applications which run in the context of the running SA or standalone applications running in the context of existing business and management systems.
- **Runtime Environment:** Provides a set of powerful, out of the box runtime services and a corresponding language independent programming model explicitly designed to be easily accessibility to a broad spectrum of programmers, from script writers to Web developers to experienced enterprise Java programmers.
- **Platform Resources:** Provide developers easy access to the platform's rich data objects, automation actions (such as patching, provisioning, and auditing), and capabilities (such as remote access to each managed server's runtime environment).
- **SA Management Network:** A powerful set of connectivity, security, and caching technologies which enable the platform to reach any device regardless of its location, IP address space, bandwidth availability, and so on.
- **SA Managed Devices:** The managed servers and network devices connected to the platform by the SA Management Network.

Automation Applications

As Figure 1 shows, 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: the Global Shell and the Global File System. Together, these two components 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

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 con.dence 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

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. Table 1 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 23.

table 1 Partial List of Services of the SA API

Service Name	Some of the Operations Provided by This Service
`AuditTaskService`	Create, get, and run audit tasks.
`ConfigurationService`	Create application configurations, get the software policies using an 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 page 22.
`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.

table 1 Partial List of Services of the SA API (cont'd)

Service Name	Some of the Operations Provided by This Service
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. TheSA 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:

- `OpswareException` - 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 `OpswareException` are `NotFoundException`, `NotInFolderException`, and `JobNotScheduledException`.
- `OpswareSystemException` - 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 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 *SA Administration Guide*.

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 163.)

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 chapter of the *SA Administration Guide*. For information about security-related exceptions, see Exceptions on page 21.

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.

2 SA CLI Methods

Overview of 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 chapter, you should be familiar with the OGSH and the OGFS. For more information, see the OGSH in the *SA User Guide: Server Automation*.

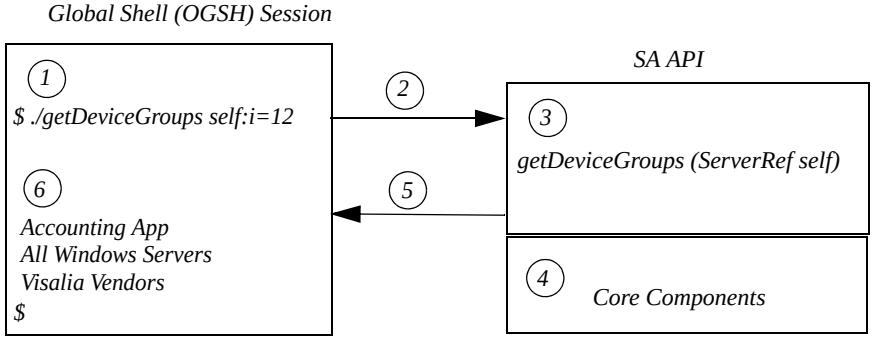
➤ For information on the `ouploa`d and `odownload` commands, see the OCLI 1.0 in the *SA User Guide: Server Automation*.

Method Invocation

As shown in Figure 2, 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.

figure 2 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 OGSH 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 *SA Administration Guide*.

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 chapter.) 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 40.

Differences Between SA CLI Methods and Unix Commands

Although you can run both Unix commands and SA CLI methods in the OGSH, 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 tutorial 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 *SA Administration Guide*.
- Your SA user has all OGSH permissions on the same managed server. For information on these permissions, see the "aaa Utility" section in the *SA User Guide: Server Automation*.
- You are familiar with the OGSH and the OGFS. If these features are new to you, before proceeding with this tutorial, see the Global Shell in the *SA User Guide: Server Automation*.

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 OGSH 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" in the *SA User Guide: Server Automation*.

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 -1
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.

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 40.)

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 Table 2.

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.

table 2 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 (`java.util.Calendar`) 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.

ID Format Specifier Examples

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 36.

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 40.

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 }
```

Structure Format Specifier Examples

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"
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
```

Directory Format Specifier Examples

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 OGSH), 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
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 "Managing Custom Attributes" in *SA User Guide: Server Automation*.

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 30.)

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

Table 3 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.

table 3 Conversion Between Primitive Types and SA CLI Methods

Primitive Type	Java Equivalent	Output from SA CLI Method	Input to SA CLI Methods
String	`java.lang.String`	Character string, presented in the encoding of the current session.	Character string, converted to Unicode from the current session encoding.
Number	`byte, short, int, long, float, double`; 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` 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"
```

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 the Structure Format Specifier Syntax on page 32.

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`.

table 4 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 (--option[=value]) and parameter values (param=value).
2	Parameter Parse Error	The parameter values could not be parsed into the object types required by the API.
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
cmnd_exit_status=$?

if [ $cmnd_exit_status -eq 0 ]
then
  echo "The command was successful."
else
  echo "The command failed."
  echo "Exit status = "  $cmnd_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:

`find`*`object`*`Refs 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 163)

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 Example Scripts on page 44.

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 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
```

Example 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 step 10 on page 29.

The script remediate_policy.sh on page 47 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 *SA User Guide: Server Automation*.

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 43.

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 if 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:{ ServerVO.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 43.

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 the 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 OGSH 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`:

1. In the upper left pane, select `com.opsware.server`.
2. In the lower left pane, select `ServerService`.
3. 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 39.)

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:

1. In the API documentation, select the `ServerVO` link. You can find the 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.
2. 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 44

3 Python API Access with Pytwist

Overview of 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 HP 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 chapter is intended for developers and consultants who are already familiar with the SA data model, custom extensions, Agents, and the Python programming language.

Setup for Pytwist

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

Supported Platforms for Pytwist

Pytwist is supported on managed servers and core servers. For a list of operating systems supported for these servers, see the *SA Release Notes*.

Pytwist relies on Python version 2.7.3, 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 65 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()

# 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 = ' + 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'
doublequote = '\"'
# 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 = ' + doublequote + platformname +
doublequote
platformrefs = platformsservice.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 = ' + 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 section provides examples of creating and deploying virtual machines (VMs) using SA API. For more examples about Virtualization, see the *HP Server Automation User Guide: Virtualization Management*.

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
# the 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 it 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 the set the  properties for your use case.


# This method constructs the deploy specification to deploy the a 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 resource pool on the hypervisor/cluster that the VM belongs to
    # It can be retrieved it 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 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 section describes the behavior and syntax that is specific to Pytwist.

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 54.) All of the arguments of `TwistServer` are optional. Table 5 lists the default values for the arguments.

table 5 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 on page 65.)

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 TwistServerobject.
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. Table 6 shows how to map the Java data types to Python for the API method invocations in Pytwist.

table 6 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).

4 Creating Automation Platform Extensions (APX)

This chapter 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 69.
- **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 69.

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:

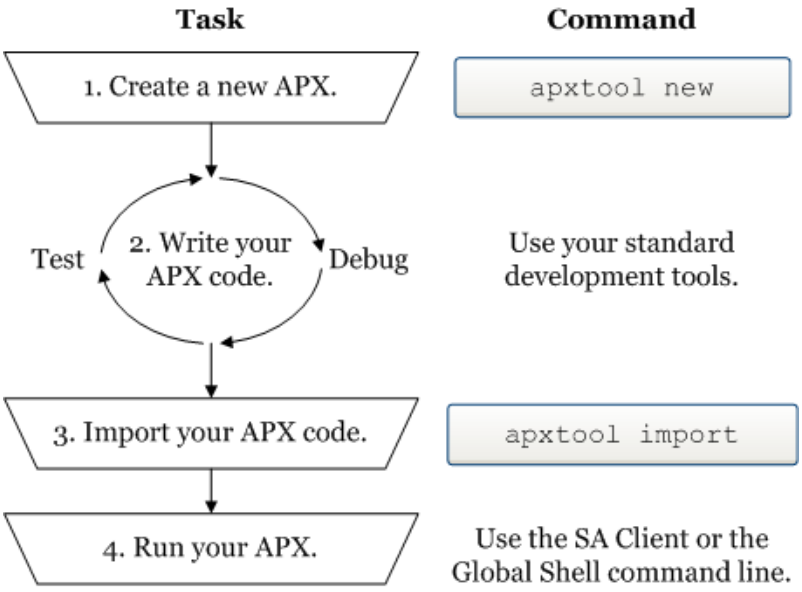
- Listed in the SA Library and can be used from the SA Client.
- Uniquely identified and managed through versioning.
- Secure because they take full advantage of SA's security model. When needed, APXs can securely and temporarily escalate a user's permissions beyond the normal defaults during the APX 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 "Running Extensions to SA" in the *SA User Guide: Server Automation*. See also the "SA Global Shell" in the *SA User Guide: Server Automation* 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 88. For a tutorial on how to create a program APX, see Tutorial: Creating a Program APX on page 94.

figure 3 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 The apxtool Command on page 76.

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" in the *SA User Guide: Server Automation* and the "SA Global Shell" in the *SA User Guide: Server Automation*.

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 "Virtual Server Management" in the *SA User Guide: Server Automation*.

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 OGSH session. They may access SA facilities such as `rosh`, the SA API, CLI, or any commands allowable from within an OGSH 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 SAS Web 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 Table 7:

table 7 APX User Roles

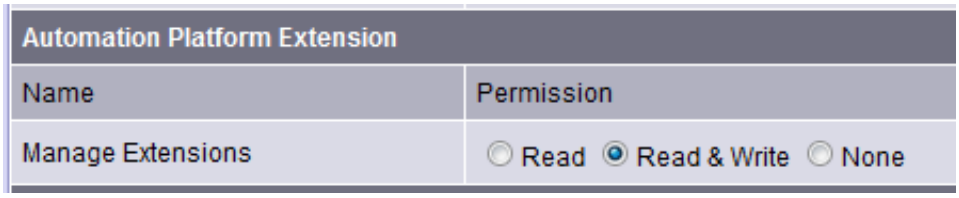
User Role	Description
End User	Runs APXs. This user typically does not have permission to modify an APX or see its content.
APX Developer	Creates and publishes APXs. This class of users can import and export APXs, and can modify APX content.
APX Administrator	Determines APXs users are permitted to run. 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

figure 4 APX Feature Permissions



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.

table 8 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.

Table 9 shows a matrix of how permissions are determined based on the combination of the Manage Extensions feature permissions and folder permissions.

table 9 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 *SA Administration Guide*.

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 The APX Permissions Escalation Configuration File - apx.perm on page 85.

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.hp.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 80 and Setting the Current Version of an APX - apxtool setcurrent on page 82 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 84.

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, OGSH. These APXs are exposed as an executable command in the OGSH. 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 a`px_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.

An Example 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 The RightClickToRun Interface on page 74.

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 77.
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.hp.sa.MyNewInterface":

   ```
   [com.hp.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 80.

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 77.
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.hp.sa.MyNewInterface", the `apx.cfg` file would contain the following lines:

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

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

   ```
   [Implementing]
   interfaces=com.hp.sa.MyNewInterface:com.hp.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 80.

➤ 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 82.

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

The RightClickToRun Interface

SA provides an interface you can use with your APXs named `com.hp.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 *SA Administration Guide*.

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.hp.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 74. For details on using an APX that implements this interface, see "Running SA Extensions" in the *SA User Guide: Server Automation*.

The CoreAffinity Interface

SA provides an interface that you can use with your APXs named 'com.hp.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 InterfaceImplementing an Interface.

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. See API Documentation and the Twister on page 23 in Chapter 1 for more information on using the SA API.

The 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 88.

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 OGSH command line with no arguments.

The APX Tool supports the following major functions:

table 10 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 77.
`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 Importing an APX into SA - apxtool import on page 80.
`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 82.
`query`	Displays information about an APX. See Querying APX Information - apxtool query on page 81.
`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 84.

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.

Table 11 lists the options for creating a new APX:

table 11 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: `script` or `webapp`. For example, -ts for script APX, -tw 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: [`a-zA-Z0-9_.`]. **Example**: `com.hp.sa.security.scan_ports`
`-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 com.hp.sa.MyWebExt, the default display name would be MyWebExt.

table 11 Options for apxtool new (cont'd)

Option	Usage
`-d <description>` `--description=<description>`	(**Required**) A brief description of an APX. If the description is a filename with the extension `.txt`, the file is assumed to be a text file and its content is used as the APX description.
`-r` `--register`	(**Optional**) Registers the name of the APX into the system. If you specify this option, you must also specify `-f` or `--folder`. If you do not specify -r and -f with `apxtool new`, you must use -f with `apxtool import`.
`-f <path>` `--folder=<path>`	(**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 `-r` or `--register` is used. If you do not specify -r and -f with `apxtool new`, you must use -f with `apxtool import`.
`-Q, --quiet`	(**Optional**) Suppresses output messages.
`-F, --force`	(**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]
```

Table 12 lists the options for deleting an APX:

table 12 Options for apxtool delete

Option	Usage
`-h` `--help`	Show this help message and exit.
`-t <type>` `--type=<type>`	(**Required**) APX type. Valid values are: `script` or `webapp`. For example `-ts` for script.
`--id=<APX id>`	(**Optional**) The object identifier of the desired APX.

table 12 Options for apxtool delete (cont'd)

Option	Usage
`-u <unique_name>` `--uniquename=<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: [`a-zA-Z0-9_.`]. **Example**: `com.hp.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.

Table 13 lists the options for exporting an APX.

table 13 Options for apxtool export

Option	Usage
`-h, --help`	Show this help message and exit.
`-t <type>, --type=<type>`	**(Required)** APX type. Valid values are: `script` or `webapp`. For example, `-ts` for script.
`--id=<APX id>`	**(Optional)** The object identifier of the desired APX.
`-u <unique_name>,` `--uniquename=<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: [`a-zA-Z0-9_.`]. **Example**: `com.hp.sa.security.scan_ports`

table 13 Options for apxtool export (cont'd)

Option	Usage
`-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.
`-v v<ersion_string>, --version=<version_string>`	**(Optional)** This option specifies which APX version to download. If omitted, the current version is downloaded.
`-a, --archive`	If specified, export the APX source in its original source archive as a ZIP or JAR file.
`-Q, --quiet`	**(Optional)** Suppresses output messages.
`-F, --force`	**(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 was 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 82.

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.

Table 14 lists the options that are available when importing an APX:

table 14 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.

table 14 Options for apxtool import (cont'd)

Option	Usage
`-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]
```

Table 15 lists the options that are available when querying APX information:

table 15 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 `list` (default), `details`, and `versions`. `-v list` is a single line representation of APX basic information presented in tabular format. `-v details` is a multiple line representation of APX information. `-v versions` lists all APX versions. You would only need to specify enough characters for the view type; for example, `-vd`, is the same as `-v details`. If the `versions` layout is selected, the query must result in a single APX object.

table 15 Options for apxtool query (cont'd)

Option	Usage
`-t <type>, --type=<type>`	**(Optional)** Specifies the type of APX to display. Valid values are: `script` or `webapp` or `interface`. The default is to display all types. `-t script` displays all script APXs. `-t webapp` displays all web APXs. `-t interface` displays all APXs that define one or more interfaces. For example, `apxtool query -ts` displays all the script APXs.
`--id=<APX id>`	**(Optional)** The object identifier of the desired APX.
`-u <unique_name> --uniquename=<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: [`a-zA-Z0-9_.`]. **Example**: `com.hp.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.
`--current`	**(Optional)** if specified, only query APX objects that have a current version set.
`--format=<format_string>`	**(Optional)** This advanced option allows you to specify custom display formatting for an APX listing. `format_string` is a string containing embedded tag names that are substituted with values at display time. Tag names must have a format of `%(tag_name)`. Use the format string "`__show_tags__`" to display a list of all the supported tag names.
`--csv`	**(Optional)** Displays the output in comma-separated values format. Ignored if the `--format` option is specified.
`-Q, --quiet`	**(Optional)** Suppresses extraneous output messages.

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 80.

Usage

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

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

Table 16 lists the options that are available when setting an APX version:

table 16 Options for apxtool setcurrent

Option	Usage
`-h, --help`	Show this help message and exit.
`-t <type>, --type=<type>`	**(Required)** APX type. Valid values are: `script, webapp`. For example, `-ts` for script.
`--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 It must have at least one dot. [`a-zA-Z0-9_.`]. **Example**: `com.hp.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.

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 Table 17:

table 17 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.

table 18 APX Files

File Name	Description
`apx.cfg`	APX configuration file, contains metadata that fully describes the APX. See The APX Configuration File - apx.cfg on page 85.
`apx.perm`	APX permissions file, specifies permission escalation rules. See The APX Permissions Escalation Configuration File - apx.perm on page 85.
`description.txt`	Text description of the APX. Specified with the `apxtool new -d` option. See Creating a New APX - apxtool new on page 77.
`interfaces`	APX interface definition file. Specifies the interfaces the APX defines or implements. See APX Interfaces - Defining Categories of APX Extensions on page 73.
`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 94.
`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 88.

The 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, "`\`".

Table 19 APX Configuration File Attributes 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.

table 19 APX Configuration File Attributes

Attribute	Modifiable?	Description
`type`	No	The type of the APX, which must be either `webapp` or `script`. (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, 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 `description.txt` instead of this attribute.
`usage`	Yes	A text description describing how to use the APX. You can alternatively use the file `usage.txt` 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.

The 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 page 86.
- All Permissions on page 86.
- With Escalation on page 86.

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.

The 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}...
```

table 20 Options to the apxprogress Command

Option	Description
`-i <total number of steps>`	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.
`-c <current step>`	Specifies the current step number. Call `apxprogress` with this option after each step in the APX code has completed.
`-m <message>`	Specifies a text message describing the status of the APX.
`-a <data>`	Specifies additional information the APX can make available about itself.
`-d`	Indicates debug mode. Displays the output of the command to stdout for debugging purposes.
`-h`	Displays help information about the apxprogress 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 7. View the APX Progress in the Twister Interface on page 98, which is part of the Tutorial: Creating a Program APX on page 94.

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.

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 `jdoe`.
- An understanding of how to set client feature permissions in the SA Client.

  For more information about permissions, see the User and Group Setup chapter of the *SA Administration Guide*.
- An understanding of how to create folders in the SA Client

  For details on folders, see the *SA User Guide: Server Automation*.
- An understanding of how to open a Global Shell session.

  For instructions, see the Global Shell chapter of the *SA User Guide: Server Automation*.
- An understanding of basic Unix commands such as `ls` and `cd`.
- Experience developing web applications that run on HTTP servers.

1. Set Permissions and Create 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.

2. Create 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.hp.sa.jdoe.mywebapp
   Create source directory /home/jdoe/mywebapp/com.hp.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.hp.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 84.

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 The APX Configuration File - apx.cfg on page 85.

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

3. Import 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.hp.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.hp.sa.jdoe.mywebapp? Y/N y
APX with unique name 'com.hp.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'.
```

4. Run the New Web Application

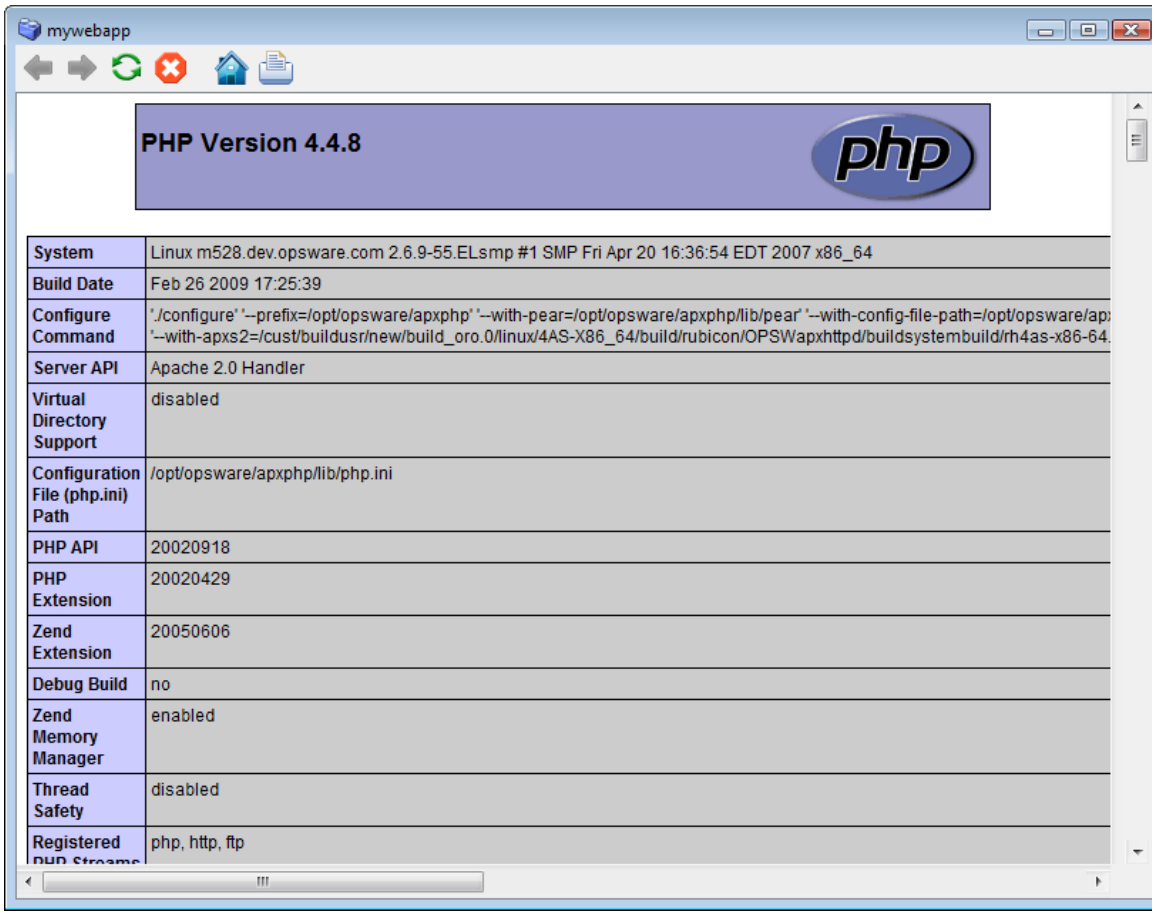
Now that you have published the web application, you are ready to run it from the SA Client, just as an end-user would.

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 1. Set Permissions and Create the Tutorial Folder on page 89.
4. To run the web application, select `mywebapp`. and select the **Actions > Run** menu.

The window shown in Figure 5 appears. The web application displays the information generated by the `phpinfo` statement of the `index.php` file.

figure 5 Web Application Version 1



5. Modify 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.hp.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.hp.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'.
```

6. Run 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 Figure 5 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.

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 `jdoe`.
- An understanding of how to set client feature permissions in the SA Client.

  For more information about permissions, see the User and Group Setup chapter of the *SA Administration Guide*.
- An understanding of how to create folders in the SA Client

  For details on folders, see the *SA User Guide: Server Automation*.
- An understanding of how to open a Global Shell (OGSH) session and use the Global Shell.

  For instructions, see the Global Shell chapter of the *SA User Guide: Server Automation*.
- An understanding of basic Unix commands such as `ls` and `cd`.

1. Set Permissions and Create 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 program APX 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 APX.
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.

2. Create 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.hp.sa.jdoe.myshellapp

   Create source directory under '/home/jdoe/myshellapp/
   com.hp.sa.jdoe.myshellapp' for APX 'myshellapp'? Y/N y
   Info: Successfully created source directory '/home/jdoe/myshellapp/
   com.hp.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.hp.sa.jdoe.myshellapp
   $ cd com.hp.sa.jdoe.myshellapp
   $ pwd
   /home/jdoe/myshellapp/com.hp.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 84.

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 The APX Configuration File - apx.cfg on page 85.

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

3. Import 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.hp.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.hp.sa.jdoe.myshellapp? Y/N y
APX with unique name 'com.hp.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.

4. Run 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 1. Set Permissions and Create the Tutorial Folder on page 94.
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.

5. Modify 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.hp.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 86.

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'.
```

6. Run 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 the Next button.

5 Select the Start Job button.

7. View 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 23.

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.

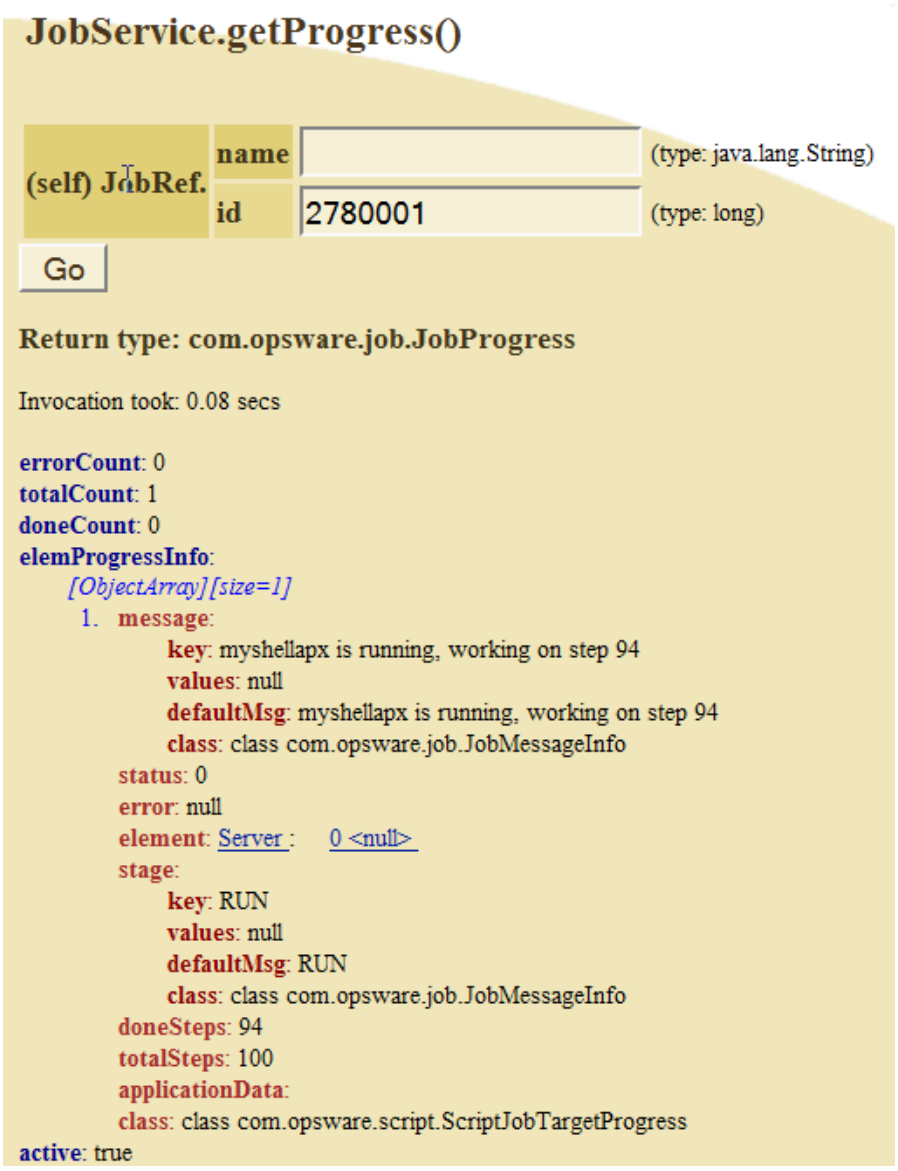
8 Select the Try It button just above the `getProgress()` method.

9 Enter your SA credentials.

10 Select the Login button.

11 In the "id" field, enter the job number of your running APX, from step 3 above.

12 Select the Go button. 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.



5 Agent Tools

Introduction to 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.

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 *SA Simple/Advanced Installation 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 HP 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:

```
/Opsware/Tools/Python Opsware 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 Opsware 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 the *SA Administration Guide*.

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
```

table 21 Agent Tool Scripts

Script	Function
get_all_cust_attr	Retrieves all custom attributes for a server record. Usage: `get_all_cust_attr.py [--localonly] [--mode=python|shell|pretty]` The mode determines the format for the output (such as Python dictionary, shell statements, etc.). `Pretty` is the default. **Note**: Shell mode does not work when there are multi-line custom attributes.
get_cust_attr	Retrieves the value of a single custom attribute. **Usage**: `get_cust_attr.py [--localonly] <custom attribute name>`
set_cust_attr	Sets the value of a single custom attribute on the server. **Usage:** `set_cust_attr.py <custom attribute name> <custom attribute value>|--valuefile <path to file with value in it>`
del_cust_attr	Deletes a custom attribute from the server's record in the database. **Usage:** `del_cust_attr.py <custom attribute name>`
get_cust_field	Retrieves the value of a single custom 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`

table 21 Agent Tool Scripts (cont'd)

Script	Function
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"
```

Do not hard code the path to Agent Tools Instead you must do the following:

1. Source the PyTwist configuration file:
UNIX:
`./etc/opt/opsware/pytwist/pytwist.conf`
Windows:
call
`C:\Program Files\Common Files\Opsware\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.

6 Microsoft Windows PowerShell/SA Integration

Introduction to Microsoft Windows PowerShell

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*.

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).

Integrated PowerShell/SA Cmdlets

Table 22 lists and describes the integrated PowerShell/SA cmdlets included with SA.

table 22 PowerShell Cmdlets

Cmdlet	Description	Arguments

`Get-SASServer`	Retrieves server data from specified server(s)	`-Credential <PSCredential>`
		`-Core <Hostname|IPAddress>`
		`-Name < ListOfHostnameFragments> |`
		`-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 <ServerVO>`

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:

1. Locate the Microsoft Windows PowerShell/SA Connector MSI package in the OCC **Library > Software Policies**.
2. 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 PowerShell Integration with SA Features

Microsoft PowerShell is available as an option in the following areas:

- Remote access to Managed Servers
- Audit and Snapshots Rules
- DSE Script Integration

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 the *SA User Guide: Audit and Compliance*.

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 53.

Sample Sessions

This section provides four scenarios that demonstrate using Windows PowerShell/ SA integration.

- Scenario 1 demonstrates extracting managed server data from an SA Core, modifying it, and writing it back to the core.
- Scenario 2 demonstrates exporting SA managed server data to an Excel spreadsheet using Windows PowerShell/SA integration.
- Scenario 3 demonstrates mounting the SA core as a Windows PowerShell PSdrive and navigating around the virtual file system.
- Scenario 4 demonstrates listing all the types of SA objects available to a Windows PowerShell environment.

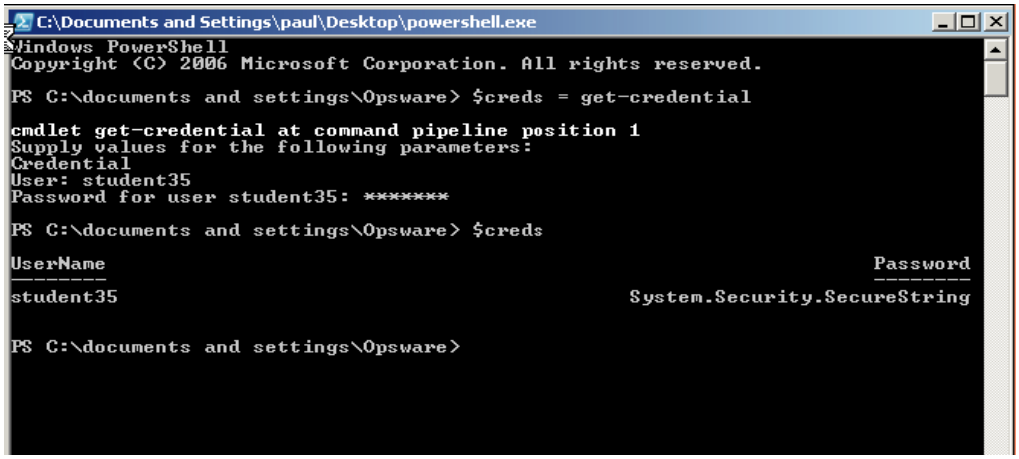
Scenario 1

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.

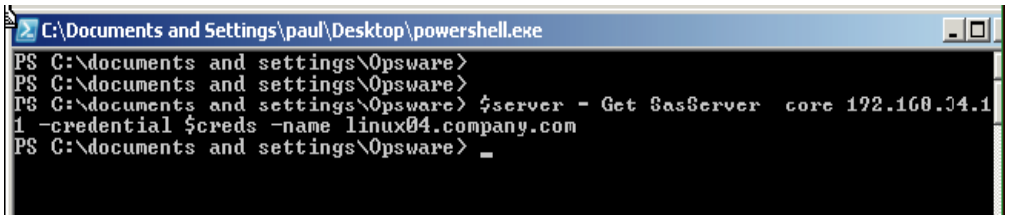
2 Store the SA Core credentials securely in a PowerShell shell variable. See Figure 6.

figure 6 Storing the SA Credentials in a PowerShell Variable



3 Using the `Get-SasServer` cmdlet, you can retrieve the SA record representing a server as shown in Figure 7.

figure 7 Using the Get-SasServer cmdlet



The returned object is stored in a shell variable.

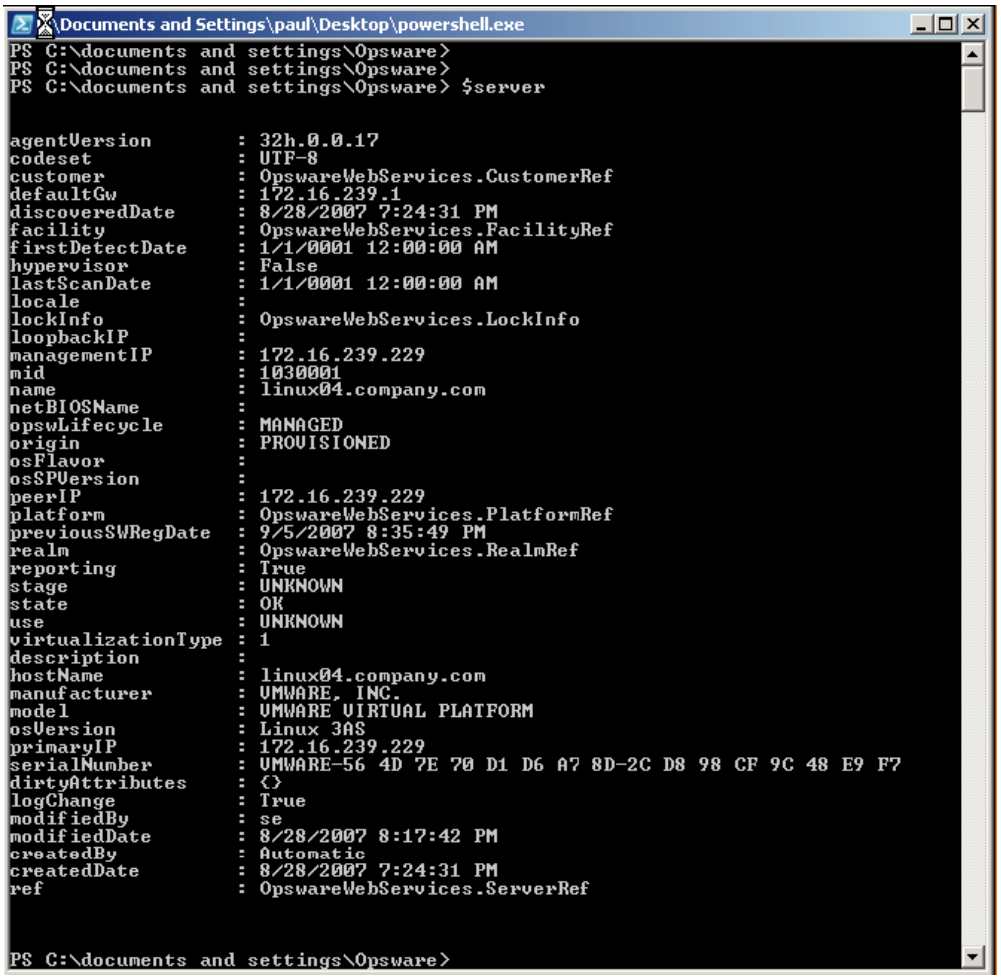
The `Get-SasServer` cmdlet takes a parameter to identify the SA Core from which the server data is to be retrieved, a parameter to supply credentials to the SA core for the operation, identifying and authenticating the SA user account in whose identity the operation is to be attempted, and a parameter 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
```

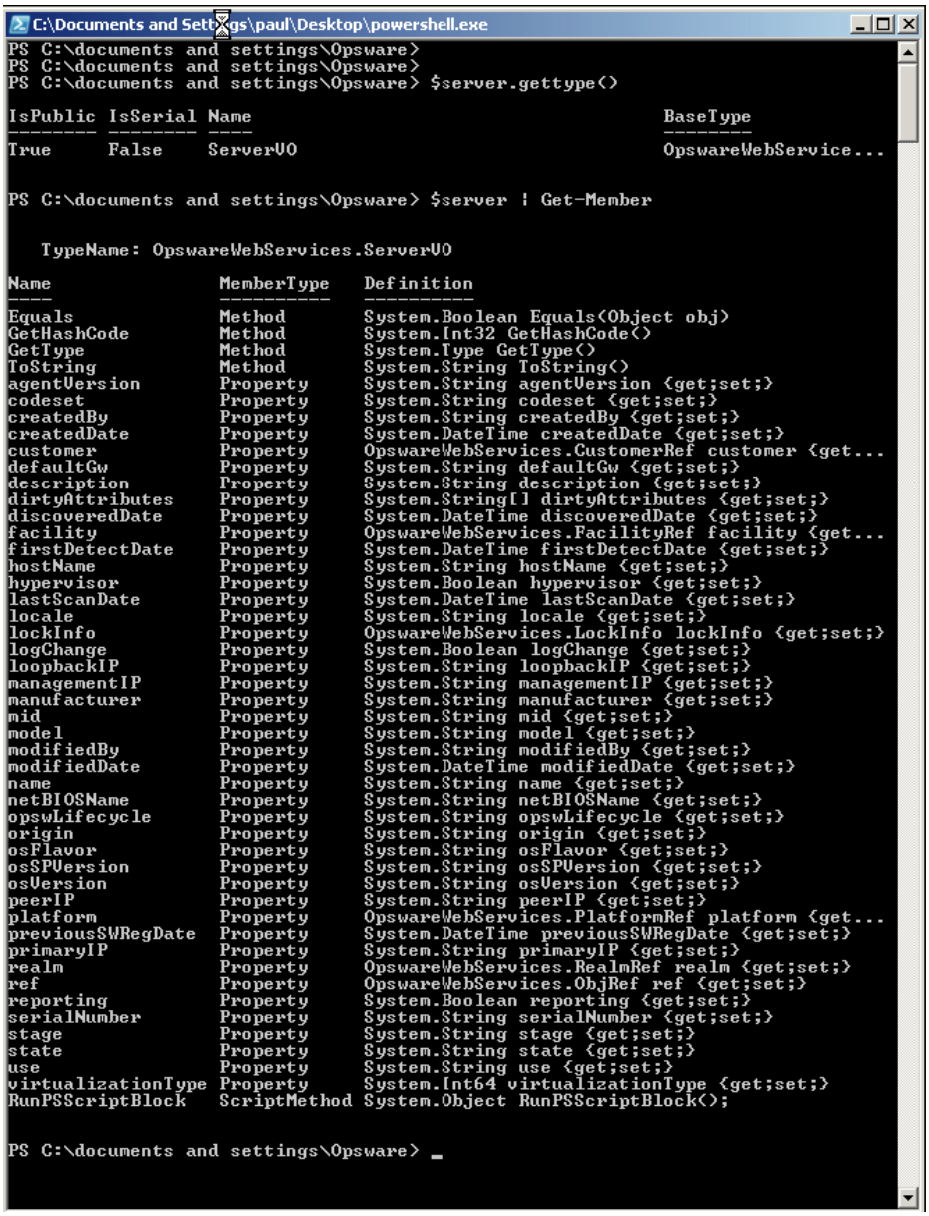
4 You can now examine the properties of the returned object by entering the name of the shell variable. See Figure 8.

figure 8 Examining SA Server Properties



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 inFigure 9.

figure 9 Listing an Object's Properties



6 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 Figure 10.

figure 10 Modifying an Object's Description

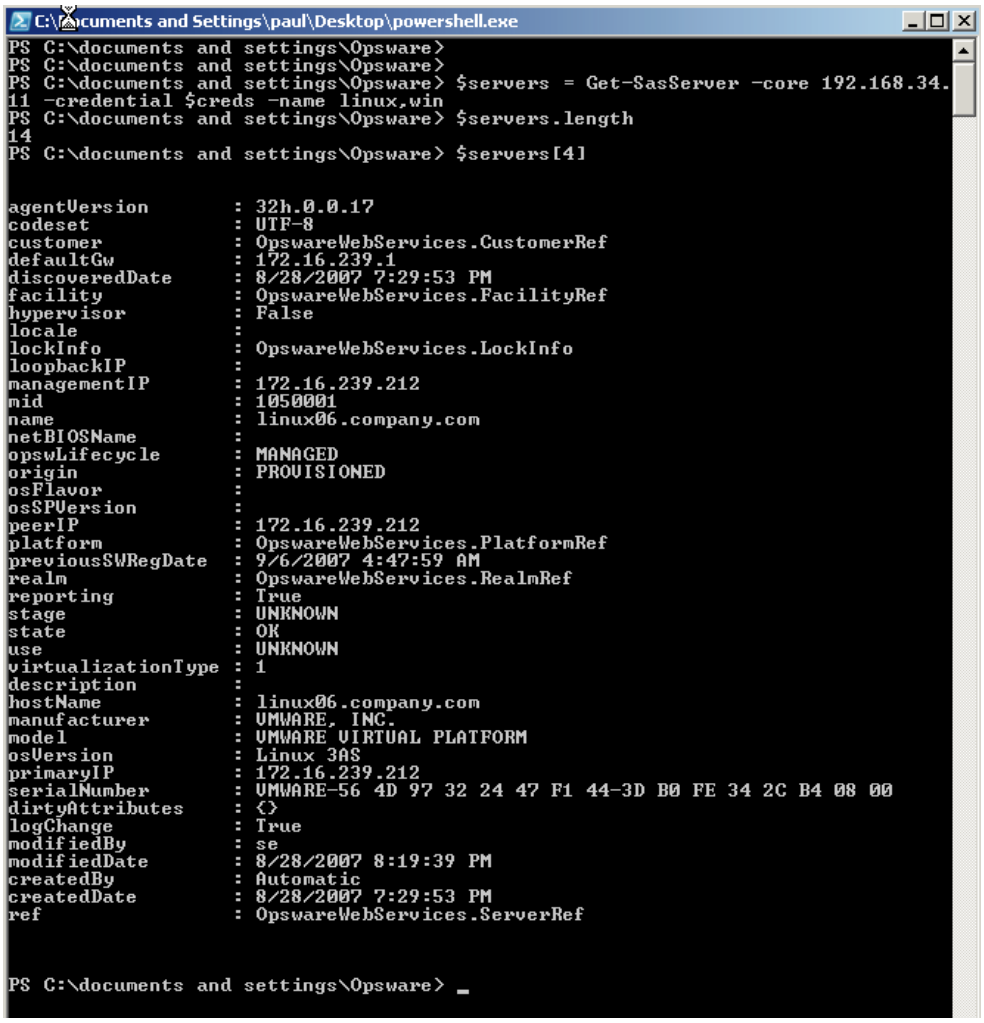
Scenario 2

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 `ServerVO`s 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.

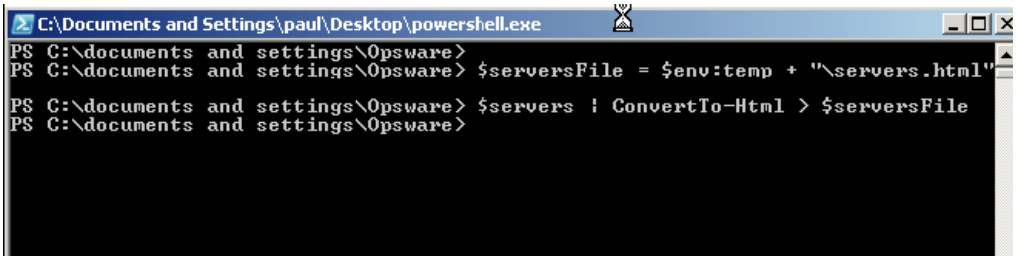
The `Get-SasServer` cmdlet returns an array of `ServerVO`s that is, in this example, 14 items in length. You can index into this array to examine any one of the `ServerVO` objects. See Figure 11.

figure 11 Using the Get-SasServer cmdlet with a Name Filter



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 Figure 12.

figure 12 Converting ServerVO Data to HTML and Saving to a Temporary File



3 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 SA Core. See Figure 13.

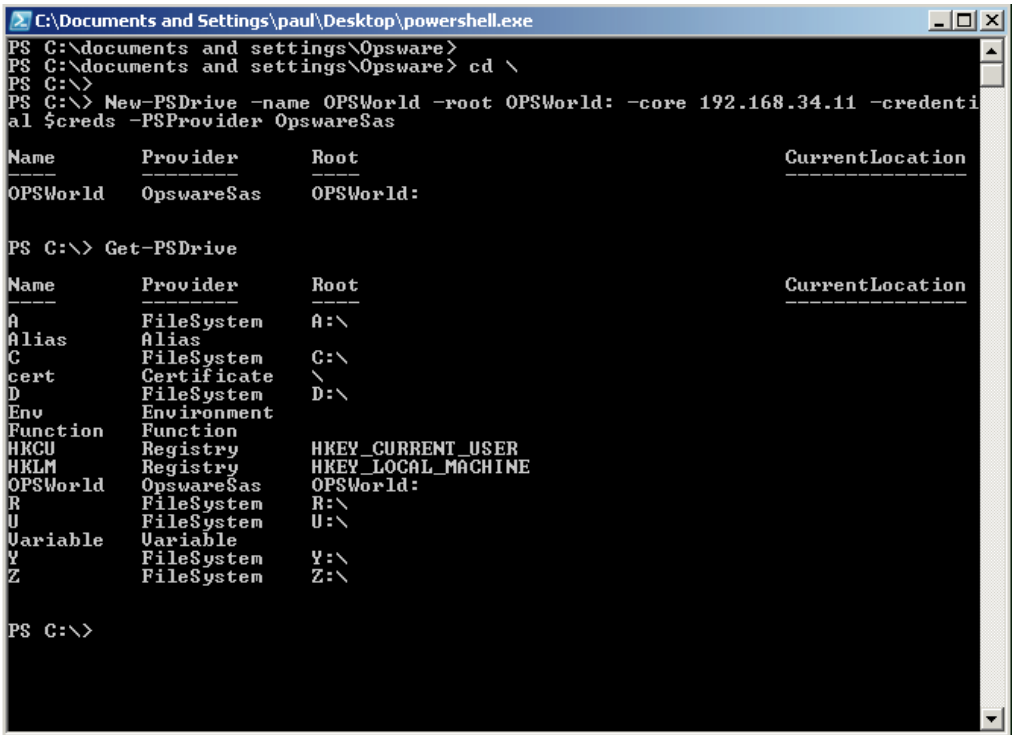
figure 13 Using the New-Object cmdlet to Launch Microsoft Excel

Scenario 3

This scenario demonstrates mounting the SA Core as a Windows PowerShell PSDrive, navigating to the SA **Jobs** folder and retrieving its contents.

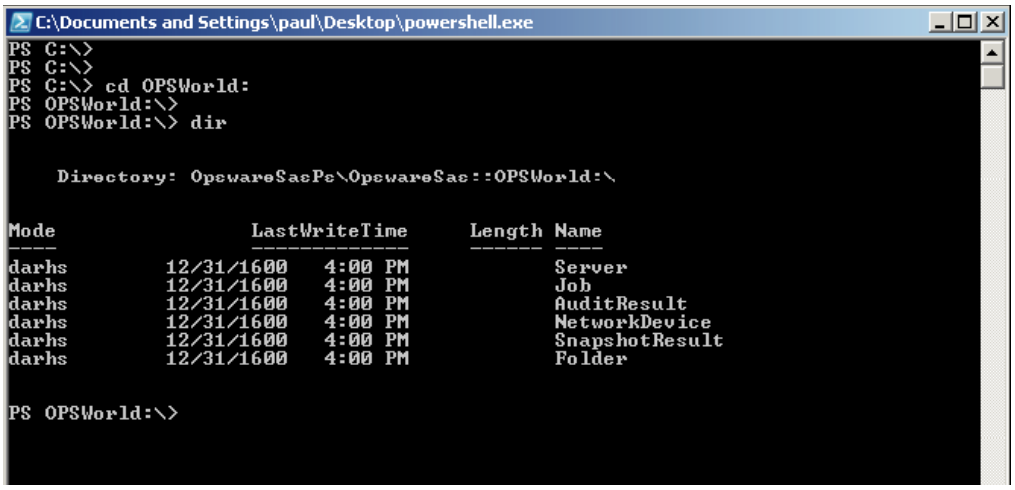
1 Mount the SA Core as a Windows PowerShell PSDrive. PowerShell allows different data stores or repositories to be navigated as if they were a file system. In this scenario, you *mount* the SA Core, specifically the managed environment data store, as if it were a drive named `OPSWorld`. The windows PowerShell 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 Figure 14.

figure 14 Mounting the SA Core as a Windows PowerShell PSDrive



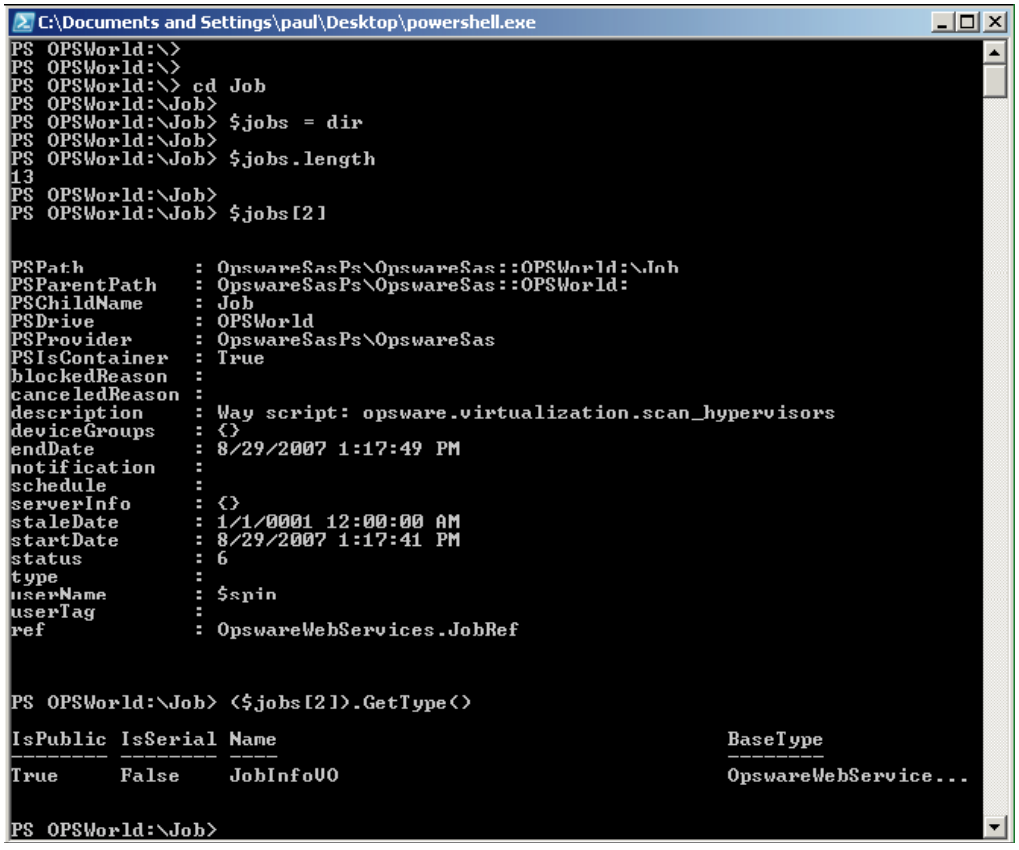
2 Change directory to the newly mounted drive and obtain a directory listing. `dir` is a PowerShell alias for the `Get-ChildItem` cmdlet. See Figure 15.

figure 15 DIR as an Alias for the Get-Child cmdlet



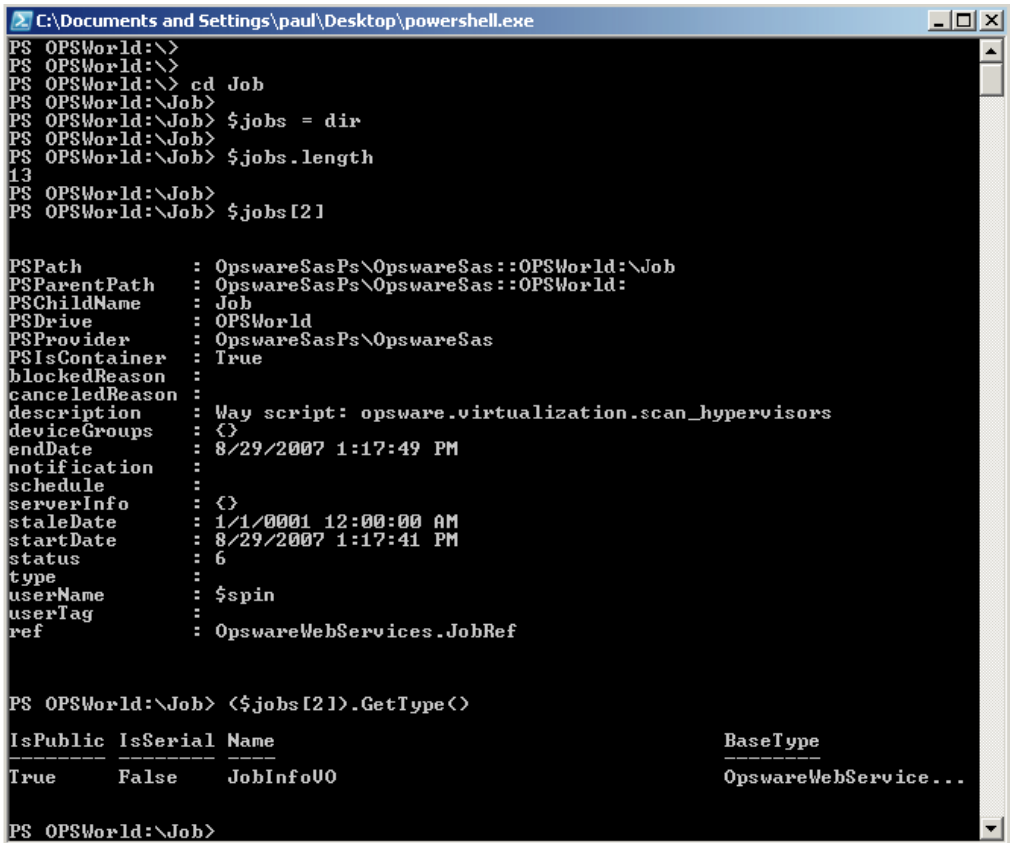
3 Change directory to the **Jobs** folder, get a directory listing, and save the directory listing as a shell variable. This shell variable will contain an array of `JobInfoVO` objects from the SA Core into which you can index. See Figure 16.

figure 16 Save a Directory Listing as a PowerShell Variable



4 Change directory to the C: drive and remove the `OPSWorld` PSDrive. See Figure 17.

figure 17 Removing the OPSWorld PSDrive

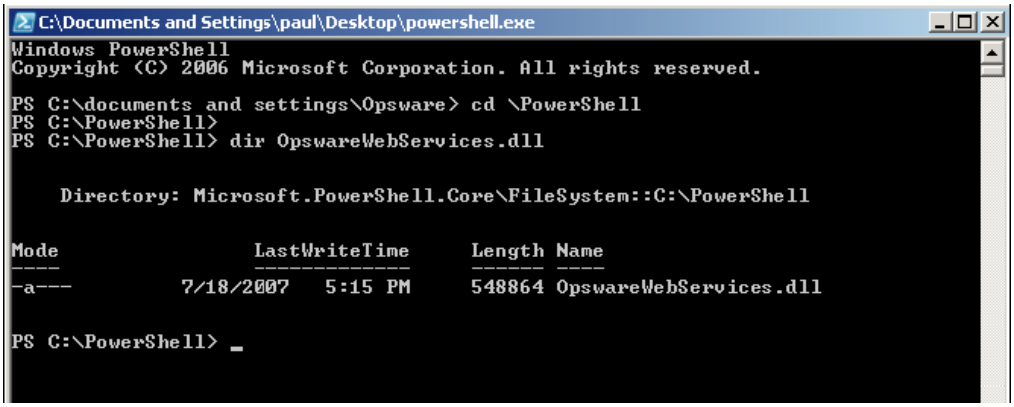


Scenario 4

This scenario describes examining all the types of SA objects available inside the Windows PowerShell environment.

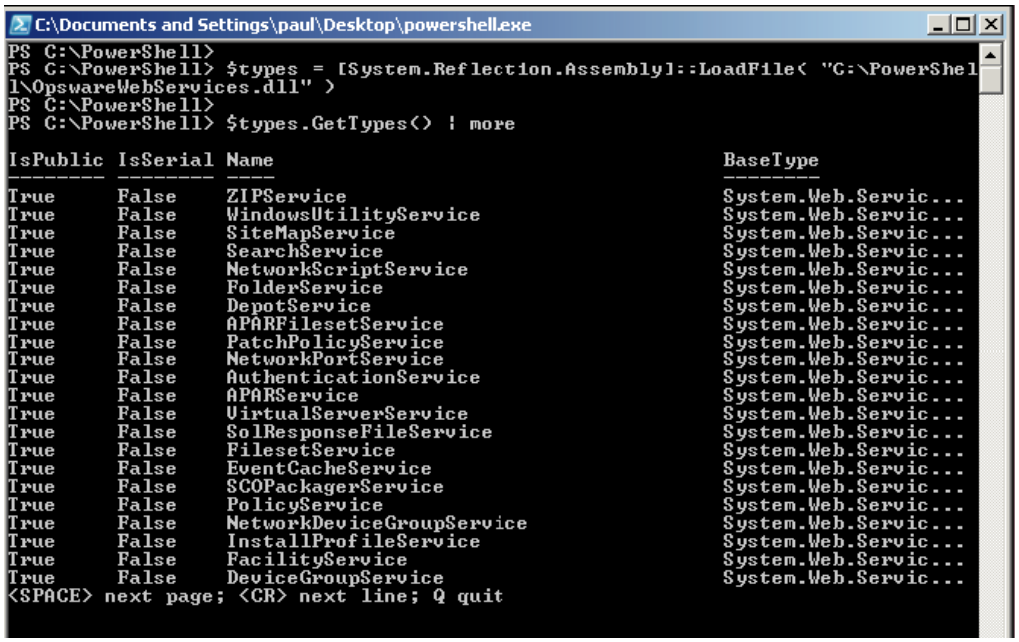
1 Locate the .NET assembly containing the PowerShell SAS Provider and cmdlets. See Figure 18.

figure 18 Locating the .NET Assembly Containing the PowerShell SAS Provider and cmdlets



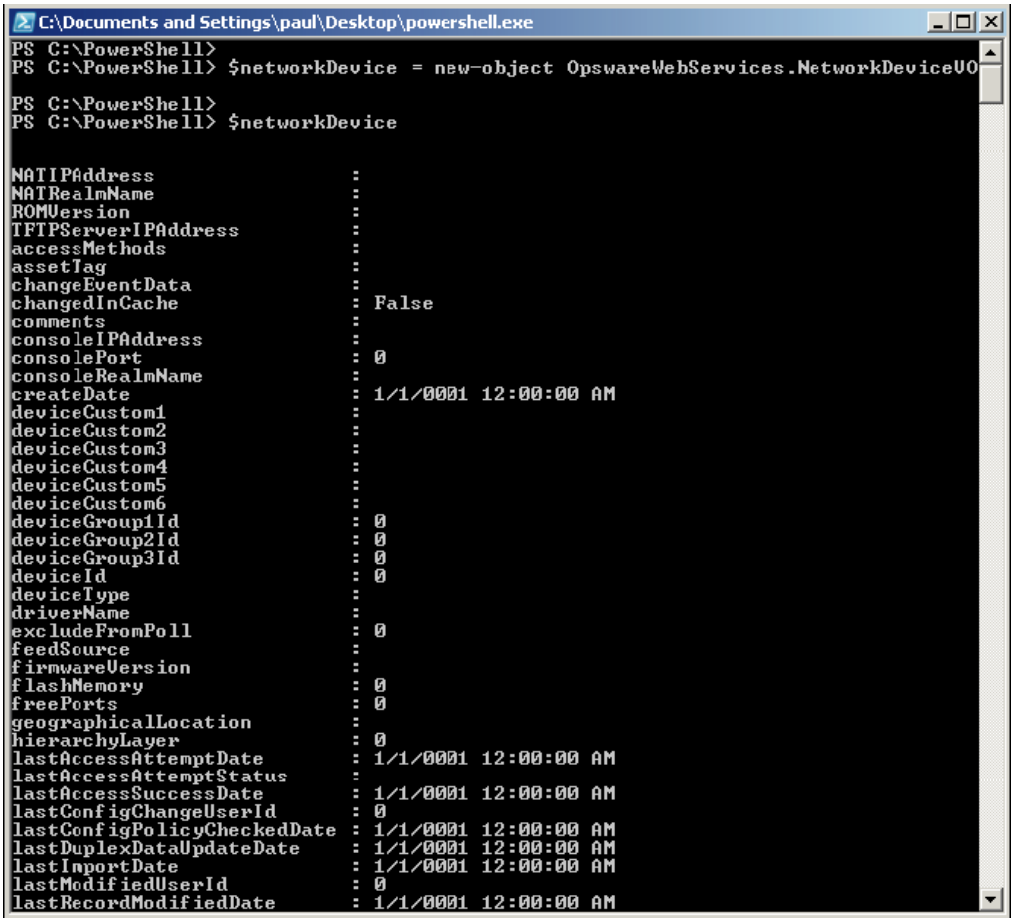
2 Using .NET Reflection, load the .NET assembly and examine the loaded types. This displays all the SA types that are available for use in the Windows PowerShell environment. See Figure 19

figure 19 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 Figure 20.

figure 20 Creating an Instance of a NetworkDeviceVO



7 Java RMI Clients

Overview of 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 chapter 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.

   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.

Java RMI Example

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:

```
OpswareClient.connect("https", host, (short)port,
userPasswd[0], userPasswd[1], true);
```

2. Gets a reference to the ServerService interface:

```
serverSvc = (ServerService)OpswareClient.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 page 121.
2. Download the ZIP file that contains the demo program `GetServerInfo.java` file.
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.

8 Web Services Clients

Overview of 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 chapter) or with Web Services-enabled development environments such as Microsoft Visual Studio .NET and BEA WebLogic Workshop.

This chapter is intended for software developers who are familiar with SA fundamentals and Web Services development.

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 chapters.

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 135.

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.CMLSnapshot.getValueSet
com.opsware.compliance.sco.CMLSnapshot.setValueSet
com.opsware.compliance.sco.SnapshotResultService.remediateCMLSnapshot
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 setDirtyAtrributes 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 the 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

Your development environment must have the following Perl modules:

- Crypt-SSLeay-0.51
- IO-Socket-SSL-0.95
- Net_SSLeay.pm-1.25
- HTML-Parser-3.35
- MIME-Base64-3.01
- URI-1.30
- libwww-perl-5.76
- SOAP-Lite-0.65_6

Depending on your Perl version, newer versions of these modules could be required.

Running the Perl Demo Program

To run the demo program, perform the following steps:

1. Obtain the ZIP file that contains the demo program `uapisample.pl` file.
2. Edit the `uapisample.pl` file, changing the hardcoded values for `host`, `username`, `password`, and object IDs such as `serverID`.
3. Run `uapisample.pl`.
4.

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
```

Perl Example 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 that represents the
# with the name self.
#
$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 23.)

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/XM
   LSchema">
   ```

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

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: `OpswareWebServices`. 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 23.

Building the C# Demo Program

To build the demo program, perform the following steps:

1. Obtain the ZIP file that contains the following demo program files:
   - `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 page 133.

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.

Running the C# Demo Program

To run the demo program, perform the following steps:

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
   ```

C# Example 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.
```

9 Pluggable Checks

Overview of 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 chapter is intended for software developers who are familiar with XML and with the Audit and Remediation feature of SA.

Setup for Pluggable Checks

Before developing pluggable checks, perform the following steps:

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 the *SA User Guide: Server Automation*.

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, perform the following steps:

1. Follow the instructions in Setup for Pluggable Checks on page 139. 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 on page 145), 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 on page 145. 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 *SA User Guide: Server Automation*.

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 `oupload` 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**.

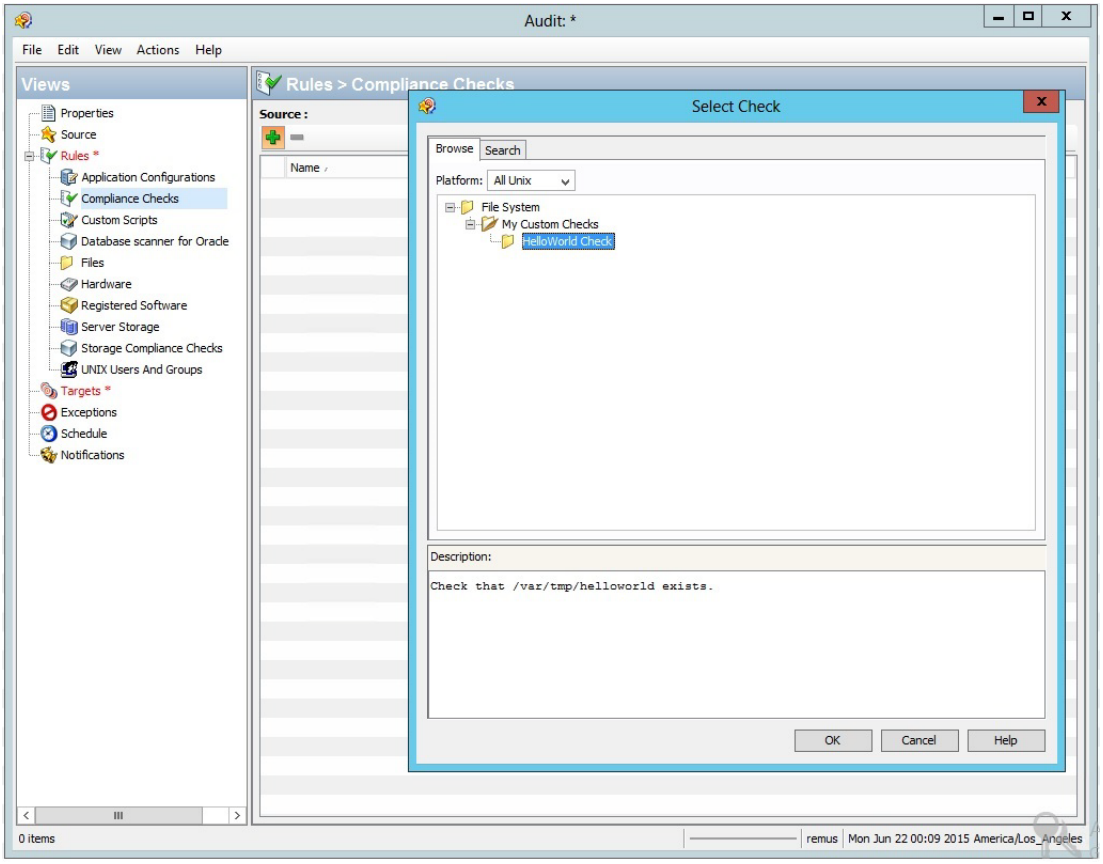
18 In the Audit Window, in the Name field of the Properties pane, enter HelloWorld Audit.

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 Figure 21.

figure 21 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 in Figure 21:

```
<!-- The check name is the rule name shown in the SA Client. -->
<checkName>HelloWorld Check</checkName>
```

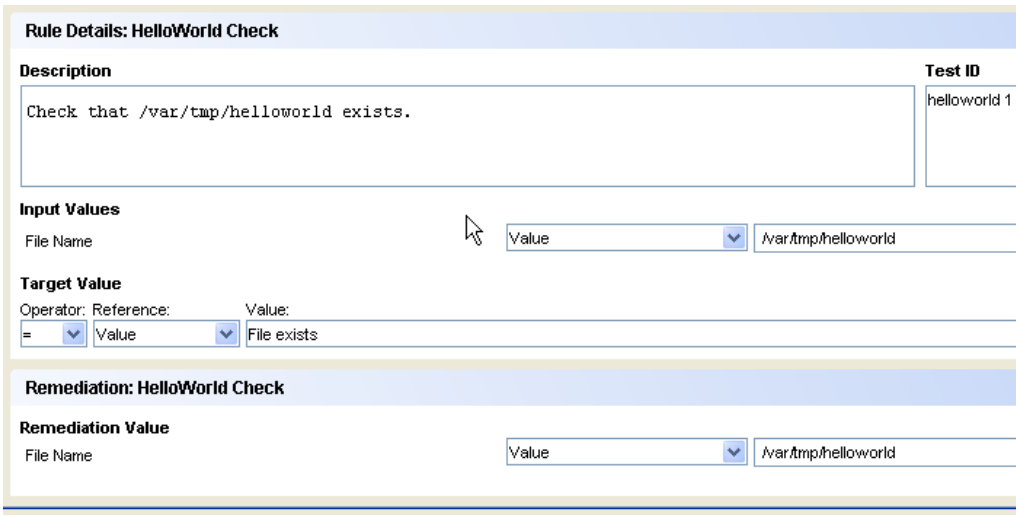
```
<!-- The category corresponds to the rule hierarchy displayed by the SA
Client. -->
```

```
<checkCategory>File System|My Custom Checks</checkCategory>
```

22 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 Figure 22.

figure 22 HelloWorld Check Rule Details



23 In the `config.xml` file, examine the following elements, which are related to the information displayed under Rule Details in Figure 22:

```
<!-- 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>
```

24 In the Views pane of the SA Client, select Targets.

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**.

27 In the Audit window, select **File** > **Save**.

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.

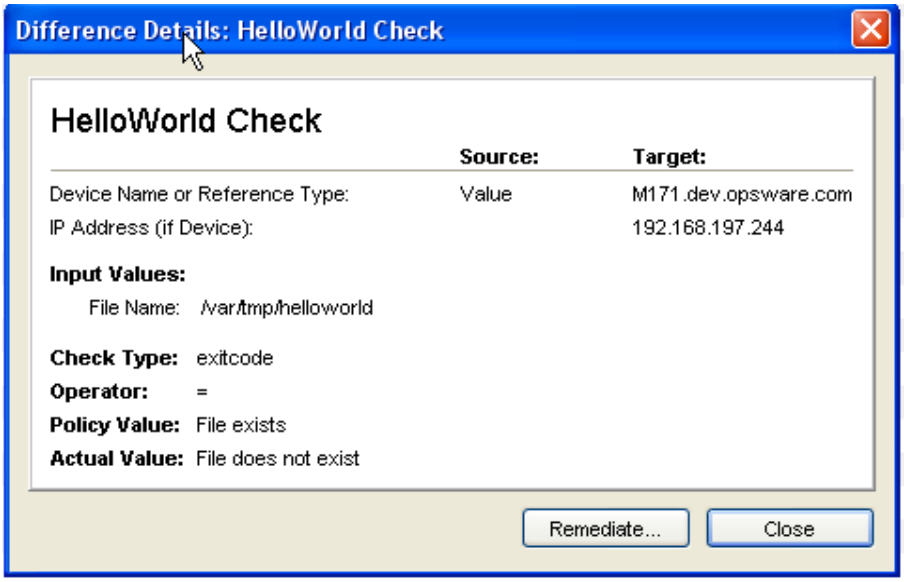
31 After the job has completed, click **View Results**.

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 Figure 23.

figure 23 HelloWorld Check Difference Details



34 In the `config.xml` file, note the following elements, which are related to the information displayed in the Difference Details window of Figure 23:

```
<!-- 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 *SA User Guide: Audit and Compliance*.

Overview of 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.

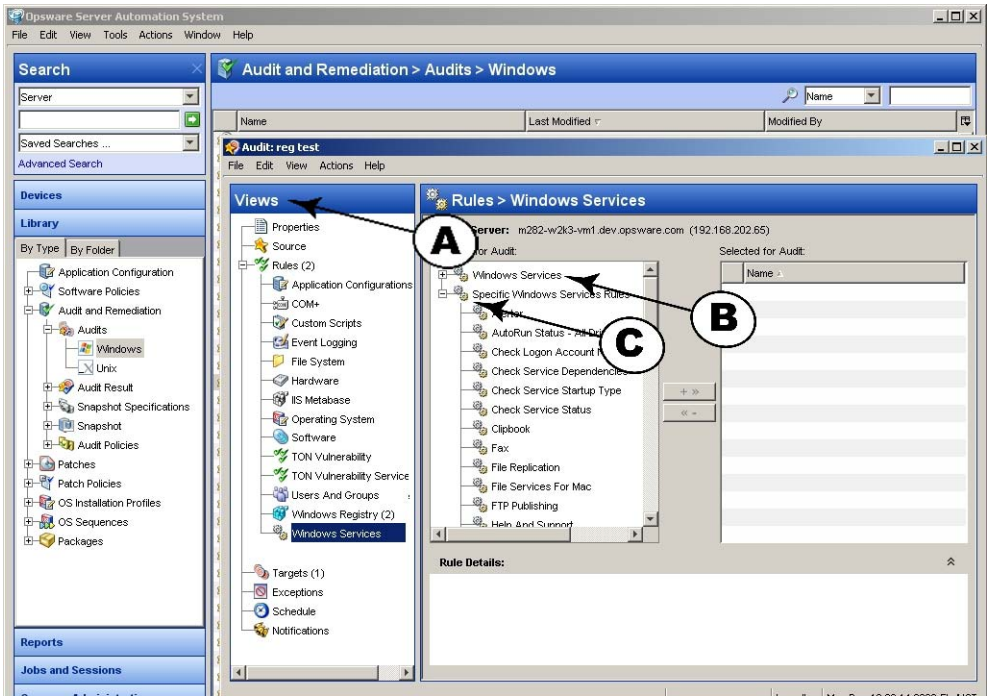
You can create an audit that defines these configurations, then audit the servers after installing the application. The audit results will confirm whether or not the application is installed and has been configured successfully according to your criteria. If the configuration is non-compliant, you can 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, Figure 24 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.

figure 24 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 HP 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 *SA User Guide: 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.

Pluggable Check Creation

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 Table 23.

table 23 Pluggable Check Contents

File Name	Description
`config.xml`	(Required) The XML configuration file defining how this pluggable check executes, returns, and ultimately reports compliance or non-compliance.
`get`*`Name`*`. {py | sh | BAT | vbs}`	(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 config.xml definitions.
`set`*`Name`*`. {py | sh | BAT | vbs}`	(Optional) The remediation script, written in Python, VBS, BAT, or shell, that remediates the condition checked by the audit script.
`Additional Code, Scripts, or Libraries`	(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.

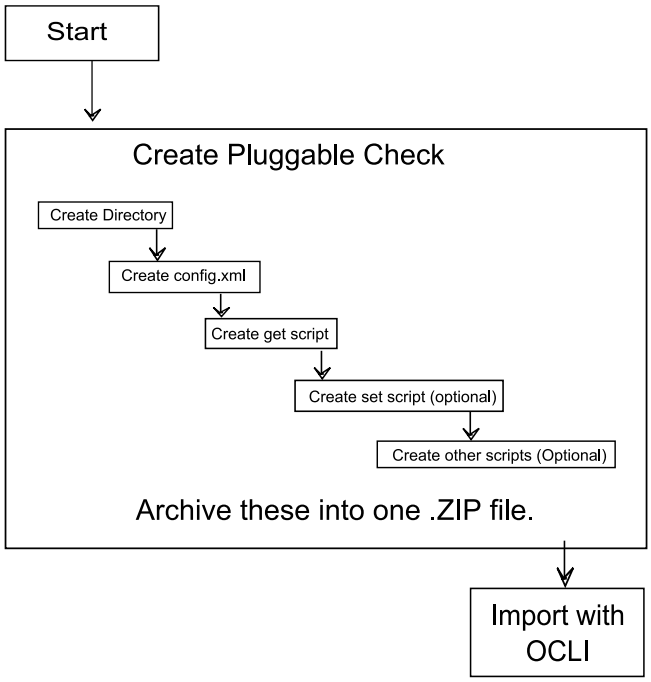
The following example shows a directory structure for a pluggable check:

```
./check_name/
./check_name/config.xml
./check_name/getcheckname.py
./check_name/setcheckname.py
```

Development Process for Pluggable Checks

Figure 25 shows an overview for the development process, which takes place in a command-line environment.

figure 25 Development Process



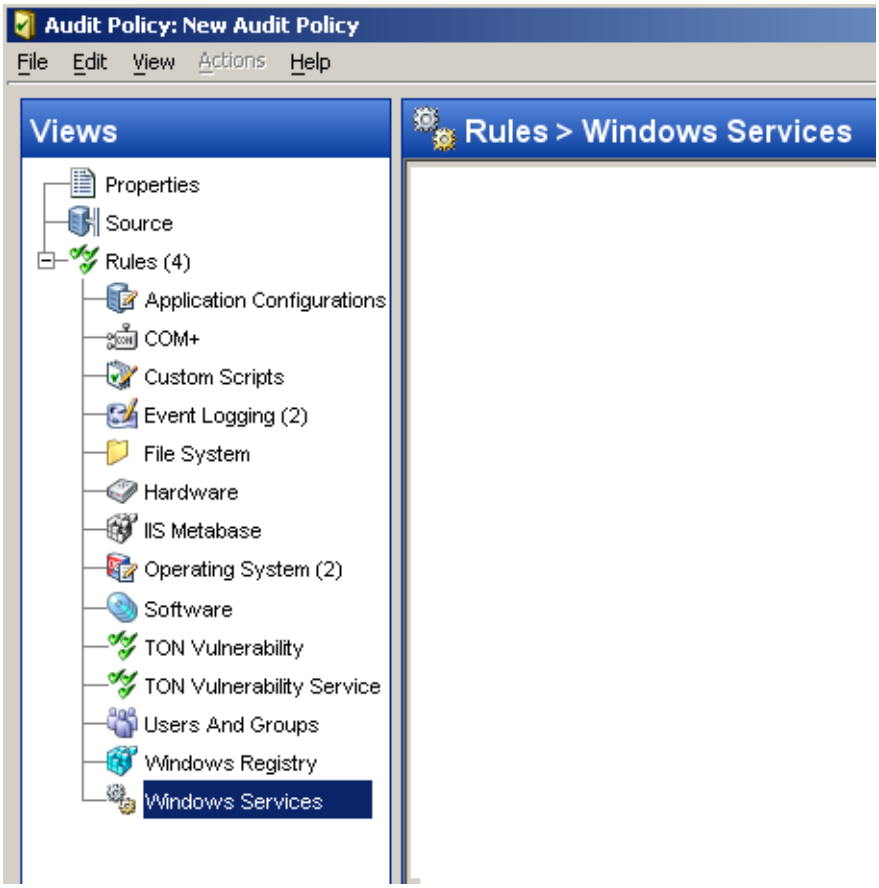
Pluggable Check Configuration (config.xml)

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 by Figure 26.

figure 26 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>$SETSCRIPTNAME</checkSetScriptName><!-- Optional -->
<checkSetScriptType>PY</checkSetScriptType><!-- Optional -->
<checkVersion>32b.0-1.0</checkVersion>
<checkReturnType>$RETURNTYPE</checkReturnType>   <!-- 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 156.

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:

```
import sys
import os
import string

if __name__ == "__main__":
```

```
  # If there are get arguments they will be loaded into sys.argv

  # Enter the desired check code here
  # Example:
  #   Looking for file "/usr/bin/ssh"

  if os.path.isfile("/usr/bin/ssh"):
     result = 1
  else:
     result = 0

  # Case A:
  #  If number/string check, the results are grabbed from    #  stdout.
  #   All debugging statements must be sent to stderr so as not
  #   to be picked up.

  sys.stderr.write("Debugging:  Found result %s\n" % result)

  sys.stdout.write(result)

  # Case B:
  # If exitcode check, the results are returned by the argument
  #   passed to sys.exit().  The exitcodes must match the
  #   ExitCodeValues defined in the config.xml file.

sys.exit(result)
```

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 Guide*. 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
```

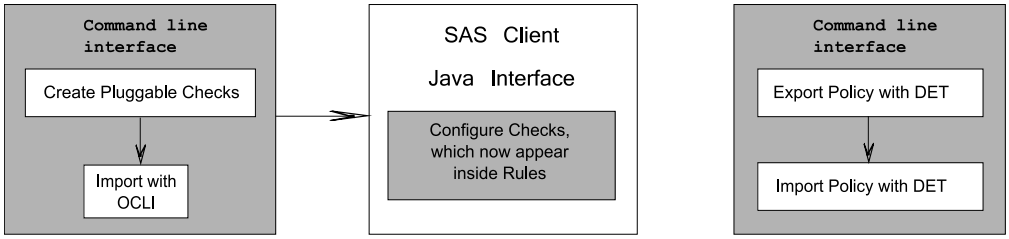
```
# ouploaD -C"Customer Independent" -t"Server Configuration Check" --
forceoverwrite --old -O"SunOS 5.8" your_Pluggable_check.zip
```

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."

Audit Policy Creation

The audit policy creation procedure is illustrated in Figure 27 below:

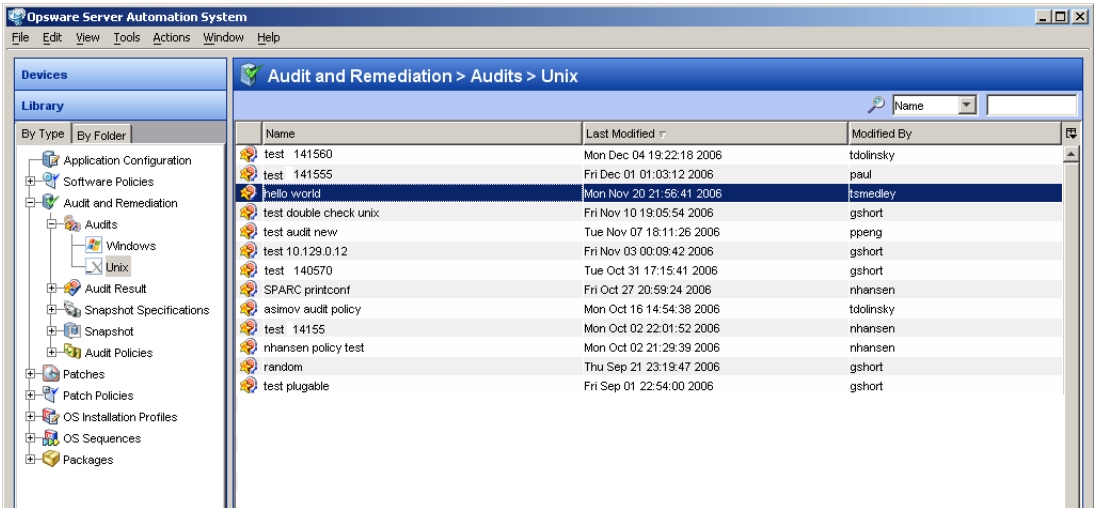
figure 27 Audit Policy Creation Procedure



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. Figure 28 shows a list of the audit rules available on a model system.

figure 28 List of Audit Rules



For detailed information on creating an audit policy, see the *SA User Guide: Audit and Compliance*.

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 *SA Content Utilities Guide*.

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.

table 24 DTD Elements and Attributes

Elements	Attributes
`checkConfiguration`	`version` Set to 1.0, only change if the Audit and Remediation framework requires it.

table 24 DTD Elements and Attributes (cont'd)

Elements	Attributes
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.
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.

table 24 DTD Elements and Attributes (cont'd)

Elements	Attributes
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)
checkExclusivePlatforms	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. This parameter may refer to one of the supported operating systems listed in the *SA Supported Platforms* documentation.
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	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: Event Logging File System Operating System Operating System|Domain Controller (sub-category) Operating System|Network (sub-category) Registry Services Users and Groups
checkGetArgument (checkGetArgumentType, checkGetArgumentDefaultLabel, checkGetArgumentDefaultDescription, checkGetArgumentDefaultValue?, checkGetArgumentValidValues?)>	Specifies parameters to the get script.
checkGetArgumentType	NUMBER | STRING
checkGetArgumentDefaultLabel	SA Client tag next to the input box or drop-down.

table 24 DTD Elements and Attributes (cont'd)

Elements	Attributes
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 Client tag 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, checkSetArgumentValidValue DisplayName)>	checkSetArgumentValidValueItem (#PCDATA)> checkSetArgumentValidValueDisplayName (#PCDATA)> checkSetArgumentValidValueItem (#PCDATA)> checkSetArgumentValidValueDisplayName (#PCDATA)>
checkSetArgumentValidValue Item	(Optional) Useful for limiting the parameters for example to 0/disable and 1/enable.
checkSetArgumentValidValueDisplayName	

table 24 DTD Elements and Attributes (cont'd)

Elements	Attributes
<!ELEMENT checkComparisonDefaults (checkComparisonDefaultOperator?, checkComparisonDefaultValues)>	checkComparisonDefaultOperator not — negation of operator specified, TRUE | FALSE checkComparisonDefaultOperator caseInsensitive — only valid for STRING types.
<!ELEMENT checkComparisonDefaultOperator (#PCDATA)>	List of default values for comparator. Useful for field or development outside the TON build framework.
checkComparisonDefaultValues (checkComparisonDefaultValue+)	checkComparisonDefaultValue (checkComparisonDefaultValueItem, checkComparisonDefaultValueDisplayName).
checkComparisonDefaultValueIte	Value for default, passed to code.
checkComparisonDefaultValueDisplayName	Display name for the value, seen in the SA Client.
checkCompareValidValues (checkCompareValidValue+)> checkCompareValidValue (checkCompareValidValueItem, checkCompareValidValueDisplayName)> checkCompareValidValueItem (#PCDATA)> checkCompareValidValueDisplayName (#PCDATA)>	
checkSuccessExitCodes (checkSuccessExitCode+) checkSuccessExitCode (checkSuccessExitCodeValue, checkSuccessExitCodeDefaultDescription, checkSuccessExitCodeDefaultDisplayName)>	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 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 *disabled* typically).
checkSuccessExitCodeDefaultDescription	Hover text for the DisplayName/Value.
checkSuccessExitCodeDefaultDisplayName	Value or text shown to user for this value, for example, Disabled.

A 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'
```

Usage Notes

The same junction type must be used within each expression junction:

- valid: `((x = y) & (a = y) & (x = a))`
- invalid: `((x = y) & (a = y) | (x = a))`

A text value needs to have double-quotes surrounding it but a number does not. Any double-quote that is part of the value must be escaped with a backslash:

- valid number: `123.456`
- valid text: `"abc"`
- invalid text: `abc`
- valid text: `"ab\"c"`
- invalid text: `"ab"c"`
- invalid text: `ab"c`

Parentheses must surround groups of expressions which will junction with another group of expressions:

- valid grouping: `((x = y) & (a = b)) | (n = r)`
- invalid grouping: `(x = y) & (a = b) | (n = r)`

B Rebuilding the Apache HTTP Server and PHP

This appendix 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 appendix 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 Creating Automation Platform Extensions (APX) on page 67.

Extending the APX HTTP Environment

This section describes how you can extend the APX HTTP environment by rebuilding the Apache HTTP server and PHP.

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 *SA Administration Guide*.

Rebuilding PHP

Perform the following tasks to rebuild PHP.

1. 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 `apxhhtps` 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/rewink && /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 *SA Administration Guide*.

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 `apxhttp` 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/rewink && /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 165.

Index