



Opsware[®] Automation Platform Developer's Guide

Corporate Headquarters

599 North Mathilda Avenue Sunnyvale, California 94085 U.S.A.
T + 1 408.744.7300 F +1 408.744.7383 www.opsware.com

Opware SAS Version 6.5

Copyright © 2000-2007 Opware Inc. All Rights Reserved.

Opware Inc. Unpublished Confidential Information. NOT for Redistribution. All Rights Reserved.

Opware is protected by U.S. Patent Nos. 6,658,426, 6,751,702, 6,816,897, 6,763,361 and patents pending.

Opware, OCC, Model Repository, Data Access Engine, Web Services Data Access Engine, Software Repository, Command Engine, Opware Agent, Model Repository Multimaster Component, and Code Deployment & Rollback are trademarks and service marks of Opware Inc. All other marks mentioned in this document are the property of their respective owners.

Additional proprietary information about third party and open source materials can be found at <http://www.opware.com/support/sas65tpos.pdf>.

Table of Contents

Preface	11
<hr/>	
About this Guide	11
Contents of this Guide	11
Chapter 1: Overview	13
<hr/>	
Overview of the Opsware Automation Platform	13
Components of the Opsware Automation Platform	14
Opsware Automation Applications	15
Opsware Runtime Environment	15
Opsware Platform Resources	17
Opsware Management Network	19
Opsware Managed Devices	20
Benefits of the Opsware Automation Platform	20
Powerful Security	20
Comprehensive Reach	21
Rich Services	21
Easily Accessible to a Broad Spectrum of Programmers	22
Opsware Automation Platform API Design	22
Services	22
Objects in the API	24
Exceptions	25
Event Cache	25

Searches	26
Security	26
API Documentation and the Twister	27
Constant Field Values	28
Importing and Exporting Packages With PUT and GET	28
Supported Clients	28
Obtaining the Code Examples	29
Chapter 2: Opsware CLI Methods	31
Overview of Opsware CLI Methods	31
Method Invocation	32
Security	32
Mapping Between API and OCLI Methods	33
Differences Between OCLI Methods and Unix Commands	33
OCLI Method Tutorial	34
Format Specifiers	39
Position of Format Specifiers	40
Default Format Specifiers	40
ID Format Specifier Examples	41
Structure Format Specifier Syntax	41
Structure Format Specifier Examples	42
Directory Format Specifier Examples	44
Value Representation	44
Opsware Objects in the OGFS	45
Primitive Values	47
Arrays	48
OCLI Method Parameters and Return Values	49

Method Context and the self Parameter	49
Passing Arguments on the Command-Line	50
Specifying the Type of a Parameter	51
Complex Objects and Arrays As Parameters	51
Overloaded Methods	51
Return Values	52
Exit Status	52
Search Filters and OCLI Methods	53
Search Syntax	53
Search Examples	54
Searchable Attributes and Valid Operators	56
Example Scripts	57
create_custom_field.sh	57
create_device_group.sh	58
create_folder.sh	59
detect_hba_version.sh	60
remediate_policy.sh	61
remove_custom_field.sh	63
schedule_audit_task.sh	63
Getting Usage Information on OCLI Methods	64
Listing the Services	64
Finding a Service in the API Documentation	65
Listing the Methods of a Service	65
Listing the Parameters of a Method	65
Getting Information About a Value Object	65
Determining If an Attribute Can Be Modified	66
Determining If an Attribute Can Be Used in a Filter Query	66

Chapter 3: Python Access to the API with Pytwist	67
Overview of Pytwist	67
Setup for Pytwist	67
Supported Platforms for Pytwist	68
Access Requirements for Pytwist	68
Installing Pytwist on Managed Servers	68
Pytwist Examples	69
get_server_info.py	70
create_folder.py	71
remediate_policy.py	72
Pytwist Details	74
Authentication Modes	75
TwistServer Method Syntax	75
Error Handling	76
Mapping Java Package Names and Data Types to Pytwist	76
Chapter 4: Java RMI Clients	79
Overview of Java RMI Clients	79
Setup for Java RMI Clients	79
Java RMI Example	80
Compiling and Running the GetServerInfo Example	80
Chapter 5: Web Services Clients	83
Overview of Web Services Clients	83
Programming Language Bindings Provided in This Release	83
URLs for Service Locations and WSDLs	83
Security for Web Services Clients	84

Overloaded Operations.....	84
Java Interface Support	84
Unsupported Data Types	84
Invoke setDirtyAttributes When Creating or Updating VOs.....	86
Compatibility With Opsware Web Services API 2.2.....	86
Perl Web Services Clients	86
Running the Perl Demo Program.....	87
Perl Example Code.....	87
Construction of Perl Objects for Web Services	91
C# Web Services Clients	95
Required Software for C# Clients.....	95
Obtaining the C# Client Stubs.....	95
Accessing the C# Stub Documentation.....	95
Building the C# Demo Program.....	96
Running the C# Demo Program	97
C# Example Code	97
Chapter 6: Pluggable Checks	101
Overview of Pluggable Checks	101
Setup for Pluggable Checks	101
Pluggable Check Tutorial.....	102
Overview of Audit and Remediation	110
Pluggable Check Creation	112
Guidelines for Pluggable Checks	113
Development Process for Pluggable Checks	115
Pluggable Check Configuration (config.xml).....	115
Audit (get) Scripts	118

Remediation (set) Scripts	119
Other Code for Pluggable Checks	120
Zippping Up Pluggable Checks	120
Importing Pluggable Checks	121
Audit Policy Creation	122
Creating an Audit Policy	122
Exporting the Audit Policy.	123
Document Type Definition (DTD) for config.xml File.	123
Chapter 7: Job Approval Integration	133
<hr/>	
Overview of Job Approval Integration	133
Scenario for Job Approvals	134
Behind the Scenes.	134
The Opsware PAS Connector	135
Prerequisites for the PAS Connector	135
Configuring Opsware SAS for Job Approval Integration.	136
PAS Connector Configuration File	136
Securing the PAS Password.	138
Enabling Job Approval Integration for Opsware SAS	138
Troubleshooting the PAS Connector	139
Managing Blocked Jobs With the Opsware API	139
Approving Blocked Jobs	139
Updating Blocked Jobs	140
Canceling Blocked Jobs	140
Searching for Blocked Jobs	140
Appendix A: Search Filter Syntax	143

Filter Grammar	143
Usage Notes	144

Preface

Document Date: May 16, 2007

Welcome to the Opsware Server Automation System (SAS) – an enterprise-class software solution that enables customers to get all the benefits of Opsware Inc.'s data center automation platform and support services. Opsware SAS provides a core foundation for automating formerly manual tasks associated with the deployment, support, and growth of server and server application infrastructure.

About this Guide

Intended for advanced system administrators and software developers, this guide explains how to create client applications for the Opsware Automation Platform.

Contents of this Guide

This guide contains the following chapters:

Chapter 1: Overview - Summarizes the Opsware Automation Platform, the Opsware API, and the supported client technologies.

Chapter 2: Opsware OCLI Methods - Explains the concepts and syntax of the Opsware CLI methods. Provides scripting examples for the methods.

Chapter 3: Python Access to the API With Pytwist - Describes how to invoke the Opsware API in Python scripts that run on managed servers or from within custom extensions.

Chapter 3: Python Access to the API With Pytwist - Describes how to invoke the Opsware API in Python scripts on managed servers or within custom extensions. Includes several examples.

Chapter 4: Java RMI Clients - Shows how to set up and create Java RMI clients that access the Opsware API. Provides a simple example.

Chapter 5: Web Services Clients - Shows how to set up and create Perl and C# clients that access the Opsware API through Web Services. Includes simple examples.

Chapter 6: Pluggable Checks - Includes a tutorial and reference information for developing and uploading customized audit rules, which are also known as pluggable checks.

Appendix A: Search Filter Syntax - Contains formal syntax for search filters as well as usage notes.

Chapter 1: Overview

IN THIS CHAPTER

This chapter discusses the following topics:

- Overview of the Opsware Automation Platform
- Components of the Opsware Automation Platform
- Benefits of the Opsware Automation Platform
- Opsware Automation Platform API Design
- Supported Clients

Overview of the Opsware Automation Platform

The Opsware Automation Platform (OAP) is a set of APIs and a runtime environment that facilitate the integration and extension of Opsware SAS. The Opsware 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 Opsware Global File System (OGFS).

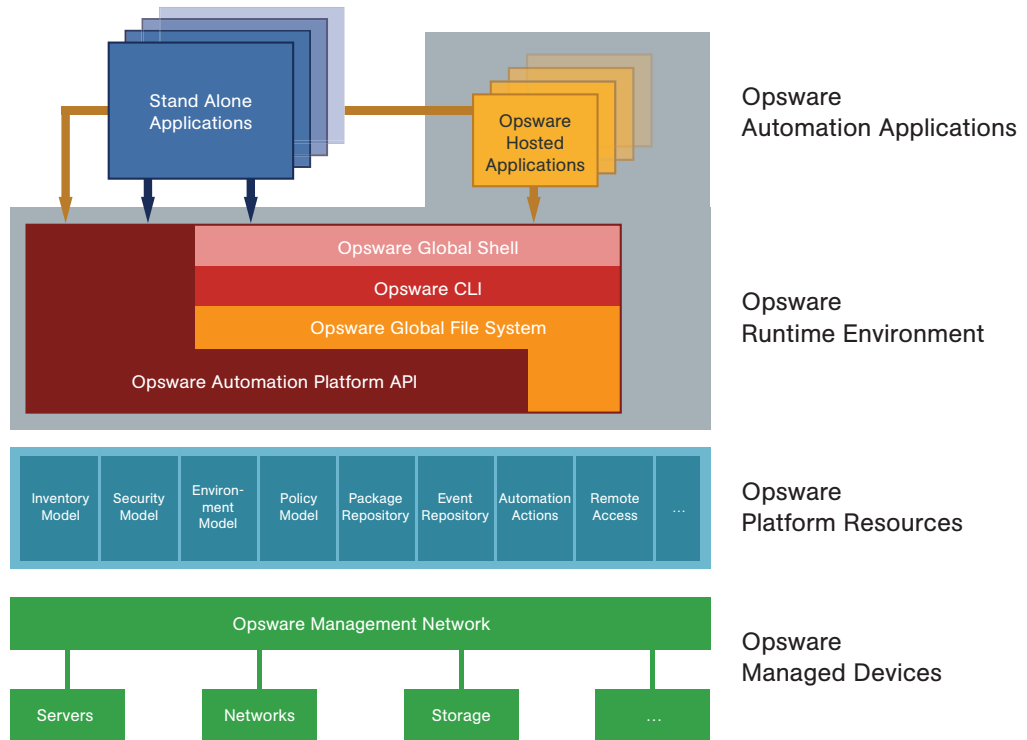
With the Opsware Automation Platform, you can perform the following tasks:

- Build new automation applications and extend Opsware SAS 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 Opsware 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 without having to wait for Opsware, Inc. to deliver out-of-the-box support for a particular technology.
- Incorporate existing Unix and Windows scripts with Opsware SAS, enabling the scripts to run in a secure, audited environment.

Components of the Opsware Automation Platform

Figure 1-1 is a layer cake diagram showing the major elements of the Opsware Automation Platform (OAP).

Figure 1-1: OAP Components



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

- **Opsware Automation Applications:** The applications users write on top of the OAP. These applications can either be Opsware Hosted Applications which run in the context of the running Opsware SAS or stand alone applications running in the context of existing business and management systems.
- **Opsware 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 scripters to Web developers to experienced enterprise Java programmers.

- **Opsware Platform Resources:** Provide developers easy access to the OAP'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).
- **Opsware Management Network:** A powerful set of connectivity, security, and caching technologies which enable the OAP to reach any device regardless of its location, IP address space, bandwidth availability, and so on.
- **Opsware Managed Devices:** The managed servers and network devices connected to the platform by the Opsware Management Network.

Opsware Automation Applications

As Figure 1-1 shows, the Opsware Automation Applications are at the top of the stack. These are the applications users write on top of the OAP.

Automation applications can either be Opsware Hosted Applications, which run in the Opsware Runtime Environment, or as stand alone applications that run in a completely independent context. Stand alone applications access the OAP 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 OAP is designed as a language-independent system easily adopted by a wide variety of developers.

Opsware Runtime Environment

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

The core of the runtime environment consists of two components: the Opsware Global Shell and the Opsware 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.

Opsware Global Shell

The Global Shell is a command-line interface to the Opsware 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 Opsware data model and the contents of managed servers—including files—into a single, virtual file system.

Opsware Global File System

The OGFS represents objects the OAP data model (such as facilities, customers, and device groups) and information available on OAP 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 Opsware 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 Opsware 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 Opsware Global File System, a virtual file system that represents all devices managed by Opsware SAS and NAS. 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.

Opsware Command Line Interface

The Opsware Command Line Interface (OCLI) provides system administrators and OAP 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 OCLI invocations.

The OCLI itself is actually programmatically generated on top of the OAP API, discussed in the next section. The advantage of this is that as soon as Opsware Inc. adds a new API to the OAP API, a corresponding OCLI 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 OCLI methods in the platform.

Opware Automation Platform API

The Opware Automation Platform API is the Win32 API of Opware: It defines a set of application programming interfaces to get and set values as well as perform actions. The Opware user interfaces, including the SAS Client and the Opware Command Line Interfaces (OCLI), are all built on top of the Opware Automation Platform API. The Opware Automation Platform 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.

Opware Platform Resources

Opware Platform Resources sit beneath the Opware 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 Opware SAS about each managed devices such as make, manufacturer, CPU, operating system, installed software, and so on. Inventory information is made available through the Opware Automation Platform API and also appears as files (in the `attr` subdirectories) in the Opware 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 OAP 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 authentication and authorization security systems of the Opware SAS.

All clients of the OAP—management applications, scripts, as well as the end-user interfaces provided by Opware Inc.—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 Opsware 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 OAP 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 that policy setters have defined in the Opsware SAS. 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 the Server Automation System. These include operating system builds, operating system patches, middleware, agents, and any other pieces of software that users have uploaded into the Opsware SAS.

Event Repository

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

Automation Actions

Automation Actions allow developers to programmatically launch any of the actions the Opsware SAS can perform on managed devices, ranging from running an audit to provisioning software to applying the latest OS patch. The OAP provides access to the same features available to end-users in the Opsware SAS Client. These features include tasks such as installing patches, provisioning operating systems, and installing and removing software policies. In fact, the Opsware SAS Client calls the same APIs that are exposed programmatically through the Opsware 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.

Opsware Management Network

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

- **Connectivity:** Allows the OAP (and hence automation applications) to reach any managed device.
- **Security:** Includes SSL/TLS-based encryption, authentication, and message integrity.
- **Address space virtualization:** Enables the OAP 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 Opsware SAS and any Opsware Automation Applications may 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.

Opsware Managed Devices

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

Opsware Inc. has a dedicated device engineering team whose entire job is to grow the list of supported devices. OAP developers and scripters benefit directly from this growing device list since their automation applications can consistently reach an ever growing list of managed devices in the same, familiar OAP programming environment.

Benefits of the Opsware Automation Platform

The Opsware Automation Platform (OAP) has the following key benefits.

Powerful Security

The OAP 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 OAP respects the role-based access controls built into the Opsware SAS so developers can easily share their applications with the confidence that they will run just on those devices that an administrator has been granted access to.
- **Digitally signed audit trail:** After an automation application runs, the OAP 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 OAP 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 office.
- **In any IP address space:** The devices can belong to any IP address space, as the OAP 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 OAP 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 OAP itself (such as device inventory data and facility information) and in part by users interacting with the OAP (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 OAP objects to include their own data. Device inventory models can be extended to include attributes the OAP 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 OAP exposes nearly all the capabilities of the underlying automation systems to developers: patching, provisioning, auditing, and others. This enables developers writing complex workflows 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 OAP is explicitly designed to appeal to a broad range of developers ranging from Unix shell and Visual Basic scripters to Perl and Python programmers to enterprise .NET or Java programmers. The OAP's Runtime Services layer makes most OAP objects available in a file-and-directory paradigm and most OAP services available from a command-line interface (the OCLI). This allows system administrators used to writing shell scripts to instantly use the OAP 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.

Opsware Automation Platform API Design

The Opsware Automation 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 Opsware Automation 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 Opsware Automation Platform API contains over 70 services – too many to describe here. Table 1-1 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 27.

Table 1-1: Partial List of Services of the Opsware 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 25.
FolderService	Create folders, get children of folders, set customers of folders, move folders.
InstallProfileService	Create, get, and update OS installation profiles.
JobService	Get progress and results of jobs, cancel jobs, update job schedules.
NasConnectionService	Get host names of NAS servers, run commands on NAS 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).

Table 1-1: Partial List of Services of the Opsware API (continued)

SERVICE NAME	SOME OF THE OPERATIONS PROVIDED BY THIS SERVICE
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 Opsware Automation Platform API is function-oriented, its design enables clients to create object-oriented libraries. The Opsware SAS data model includes objects such as servers, folders, and customers. These are persistent objects; that is, they are stored in the Opsware 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 Opsware 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 Opware SAS are derived from one of the following exceptions:

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

The following exceptions are related to security:

- `AuthenticationException` - Thrown when an invalid Opware 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 *Opware[®] SAS Administration Guide*.

Event Cache

Some client applications need to keep local copies of Opware SAS objects. Accessed by clients through the `EventCacheService`, the cache contains events that describe the most recent change made to Opware SAS 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 2 hours are maintained. To change the sliding window size, edit the Web Services Data Access Engine configuration file, as described in the *Opware[®] SAS Administration Guide*.

Searches

The search mechanism of the Opsware Automation 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 143.)

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

```
ServerVO.hostName = "d04.opsware.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 Opsware Automation Platform must be authenticated and authorized to invoke methods on the Opsware Automation Platform API. To connect to Opsware SAS, a client supplies an Opsware user name and password (authentication). To invoke methods, the Opsware user must belong to a user group with the necessary permissions (authorization). These permissions restrict not only the types of Opsware SAS 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 Opsware Administrator must specify the required users and permissions with the Opsware Command Center. For instructions, see the User Group and Setup chapter of the *Opsware® SAS Administration Guide*. For information about security-related exceptions, see “Exceptions” on page 25.

Communication between clients and Opsware SAS 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

The Opsware SAS 6.0 core ships with API documentation (javadocs) that describe the Opsware Automation Platform API. To access the API documentation, specify the following URL in your browser:

```
https://occ_host:1032/twister/docs/index.html
```

Or:

```
https://occ_host:443/twister/docs/index.html
```

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

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

```
https://occ_host:443
```

Also included in the core, 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 Opsware SAS 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://occ\_host:1032/twister/docs/constant-values.html
```

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

Importing and Exporting Packages With PUT and GET

The following wiki page is available only to Opsware, Inc. employees:

```
http://wiki.corp.opsware.com/owiki/  
OpswareReleases\_2fEinstein\_2fPatchManagement\_2fFileTransferApi
```

Supported Clients

The Opsware Automation 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 Opsware Automation Platform. A developer can create the following types of clients that call methods in the Opsware Automation Platform API:

- **Opsware Command-line Interface (OCLI):** Launched from Global Shell sessions, shell scripts can access the Opsware Automation Platform API by invoking the OCLI methods, which are executable programs in the OGFS. Each OCLI method corresponds to a method in the Opsware Automation Platform API.
- **Web Services:** Using SOAP over HTTPS, these clients send requests to Opsware SAS and get responses back. The Web Services operations (defined in WSDLs) correspond to the methods in the Opsware Automation Platform 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 Opsware SAS managed or core servers.

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

Obtaining the Code Examples

To obtain the code examples discussed in this guide, perform the following steps:

- 1** In a browser, go to the Opsware Technical Support Downloads page:
<https://download.opsware.com/download/>
- 2** Download the ZIP file labelled Opsware SAS API Code Examples.

Chapter 2: Opsware CLI Methods

IN THIS CHAPTER

This chapter contains the following topics:

- Overview of Opsware CLI Methods
- OCLI Method Tutorial
- Format Specifiers
- Value Representation
- OCLI Method Parameters and Return Values
- Search Filters and OCLI Methods
- Example Scripts
- Getting Usage Information on OCLI Methods

Overview of Opsware CLI Methods

End-users access Opsware SAS through the GUI utilities, that is, the SAS Client and the SAS Web Client. At times, advanced users need to access Opsware SAS in a command-line environment to perform bulk operations or repetitive tasks on multiple servers. In Opsware SAS, the command-line environment consists of the Global Shell, OGFS, and Opsware Command-line Interface (OCLI) methods.

To perform Opsware SAS operations from the command-line, you invoke OCLI methods from within a Global Shell session. An OCLI method is an executable in the OGFS that corresponds to a method in the Opsware API. When you run an OCLI method, the underlying API method is invoked.

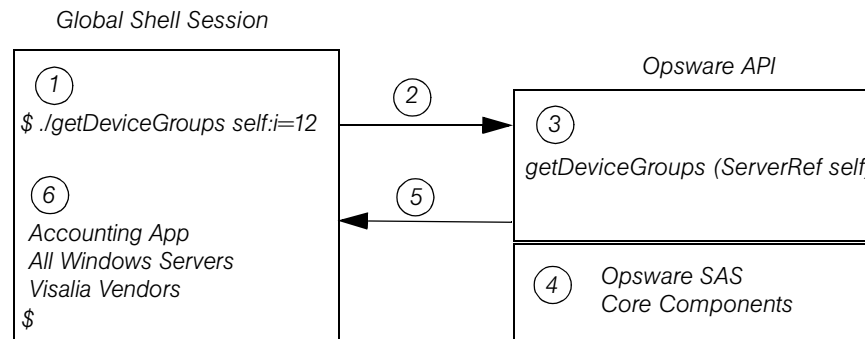
In order to understand this chapter, you should already be familiar with the Opsware Global Shell and the OGFS. For a quick introduction to these features, see the “Global Shell Tutorial” in the *Opsware® SAS User’s Guide: Server Automation*.

Method Invocation

As shown by Figure 2-1 when an OCLI method is invoked, the following operations occur:

- 1** In a Global Shell session, the user enters an OCLI method with parameters.
- 2** The command-line entered in the previous step is parsed to determine the API method and parameters.
- 3** The underlying API method is invoked.
- 4** An authorization check verifies that the user has permission to perform this operation. Then, Opware SAS performs the operation.
- 5** The API method passes the results back to the OCLI method.
- 6** The OCLI method writes the return value to the `stdout` of the Global Shell session. If an exception was thrown, the OCLI method returns a non-zero status.

Figure 2-1: Overview of an OCLI Method Invocation



Security

OCLI methods use the same authentication and authorization mechanisms as the SAS Client and the SAS Web Client. When you start a Global Shell session, Opware SAS authenticates your Opware user. When you run an OCLI method, authorization is performed. To run an OCLI method successfully, your Opware user must belong to a group that has the required permissions. For more information on security, see the *Opware® SAS Administration Guide*.

Mapping Between API and OCLI Methods

The OGFS represents Opware SAS objects as directory structures, object attributes as text files, and API methods as executables. These executables are the OCLI methods. Every OCLI 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 OCLI 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 OCLI method does not write the result to the `stdout`. For information on how OCLI methods return strings that represent objects, see “Return Values” on page 52.

Differences Between OCLI Methods and Unix Commands

Although you can run both Unix commands and OCLI methods in the Global Shell, OCLI methods differ in several ways:

- Unlike many Unix commands, OCLI methods do not read data from `stdin`. Therefore, you cannot insert an OCLI method within a group of commands connected by pipes (`|`). (However, OCLI methods do write to `stdout`.)
- Most Unix commands accept parameters as flags and values (for example, `ls -l /usr`). With OCLI 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, OCLI methods can accept and return complex objects.
- With OCLI methods, you can specify the format of the parameter and return values. Unix commands do not have an equivalent feature.

OCLI Method Tutorial

This tutorial introduces you to OCLI methods with a few examples for you to try out in your environment. After completing this tutorial, you should be able to run OCLI methods, examine the `self` file of an Opware object, and create a script that invokes OCLI methods on multiple servers.

Before starting the tutorial, you need the following capabilities:

- You can log on to the SAS Client.
- Your Opware user has Read & Write permissions on at least one managed server. Typically assigned by a security administrator, permissions are discussed in the *Opware® SAS Administration Guide*.
- Your Opware user has all Global Shell permissions on the same managed server. For information on these permissions, see the “aaa Utility” section in the *Opware® SAS User's Guide: Server Automation*.
- You are familiar with the Global Shell and the OGFS. If these features are new to you, before proceeding with this tutorial you should step through the “Global Shell Tutorial” in the *Opware® SAS User's Guide: Server Automation*.

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

1 Open a Global Shell session.

You can open a Global Shell session from within the SAS Client. From the **Actions** menu, select **Global Shell**. You can also open a Global Shell session from a terminal client running on your desktop. For instructions, see “Opening a Global Shell Session” in the *Opware® SAS User's Guide: Server Automation*.

2 List the OCLI 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 OCLI methods for the `abc.opware.com` server:

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

```

checkDuplex
clearCustAttrs
. . .

```

These methods have instance context – they act on a specific server instance (in this case, `abc.opsware.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 OCLI method without parameters.

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

```

$ cd /opsw/Server/@/abc.opsware.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.opsware.com` server from Opware to C39. The ID of customer C39 is `20039`.

```

$ cd /opsw/Server/@/abc.opsware.com
$ cat attr/customer ; echo
Opware
$ 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.opsware.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 Opware SAS, 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 Opware SAS 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.opsware.com
$ cat self ; echo
abc.opsware.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.opsware.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.opsware.com
$ cat .self:i ; echo
com.opsware.server.ServerRef:530039
```

This output shows that the ID of `abc.opsware.com` is 530039. The `com.opsware.server.ServerRef` is the class name of a server reference, the corresponding object in the Opsware 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 51.)

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

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

10 Create a script that invokes an OCLI 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` OCLI 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.opsware.com
2006/06/20 16:46:56.000
. . .
```

Format Specifiers

Format specifiers indicate how values are displayed or interpreted in the OCLI 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 2-1.



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 2-1: 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.	Opsware objects	Yes. If the name is ambiguous, an error occurs.
:i	ID: A format that uniquely identifies the object type and its Opsware ID. Also known as an object reference.	Opsware objects; Dates (<code>java.util.Calendar</code>) objects	Yes. If the type is clear from the context, the type may be omitted.
:s	Structure: A compact representation intended for specifying complex values on the command-line. Attributes are enclosed in curly braces.	Any complex object	Yes

Table 2-1: Summary of Format Specifiers (continued)

FORMAT SPECIFIER	DESCRIPTION	VALID OBJECT TYPES	ALLOWED AS METHOD PARAMETER?
: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.opsware.com/attr
cat osVersion
cat .osVersion:n
```

The name format specifier is the default for Opsware 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.opsware.com` server belongs to:

```
cd /opsw/Server/@/d04.opsware.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.opsware.com` belongs to:

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

For more ID format examples, see “The self File” on page 46.

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 51.

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.opsware.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.opsware.com/method
./getDeviceGroups:s
```

This call to `getDeviceGroups` displays the following output. Because `d04.opsware.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.opsware.com"
devices="d04.opsware.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.opsware.com"
devices="glengarriff.snv1.dev.opsware.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.opsware.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 Opsware 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.opsware.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.opsware.com`:

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

The next command lists the name of this customer:

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

However, the next command is legal:

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

Value Representation

Because they run in a shell environment (Global Shell), OCLI 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 OCLI methods represent non-string data types.

Opsware Objects in the OGFS

The Opsware 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 Opsware SAS 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.opsware.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.opsware.com` server, enter the following commands:

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

Custom Attributes

Custom attributes are name-value pairs that you can assign to Opsware objects such as servers. In the OGFS, custom attributes are represented as text files in the `CustAttr` subdirectory. You can create custom attributes in a Global Shell 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.opsware.com` server:

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

For more examples, see “Managing Custom Attributes” in *Opsware® SAS User's Guide: Server Automation*.

The self File

The `self` file resides in the directory of an Opsware SAS 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 39.)

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

```
cd /opsw/Server/@/buzz.opsware.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 2-2 indicates how primitive values are converted between the API and their string representations in OCLI methods. Except for Dates, primitive values do not support format specifiers. Dates support ID format specifiers.

Table 2-2: Conversion Between Primitive Types and OCLI Methods

PRIMITIVE TYPE	JAVA EQUIVALENT	OUTPUT FROM OCLI METHOD	INPUT TO CLI METHODS
String	<code>java.lang.String</code>	Character string, presented in the encoding of the current session.	Character string, converted to Unicode from the current session encoding.
Number	<code>byte</code> , <code>short</code> , <code>int</code> , <code>long</code> , <code>float</code> , <code>double</code> ; and their object equivalents	Decimal format, not localized. Scientific notation for very large or small values.	Examples - Decimal: 101, 512.34, -104 Hex: 0x1F32, 0x2e40 Octal: 0543 Scientific: 4.3E4, 6.532e-9, 1.945e+02
Boolean	<code>boolean</code> , <code>Boolean</code>	<code>true</code> or <code>false</code>	The string "true" and all mixed-case variants evaluate to <code>true</code> . All other values evaluate to <code>false</code> .
Binary data	<code>byte[]</code> , <code>Byte[]</code>	Binary string. No conversion from session encoding.	Binary string. No conversion to session encoding.

Table 2-2: Conversion Between Primitive Types and OCLI Methods

PRIMITIVE TYPE	JAVA EQUIVALENT	OUTPUT FROM OCLI METHOD	INPUT TO CLI METHODS
Date	java.util.Calendar	Date value. By default, presented in this format: YYYY/MM/DD HH:MM:SS 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 to the underlying type, separated by new-line characters. Within an array element, a new-line character is escaped by `\n` and a backslash 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
```

OCLI 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 Opsware SAS 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.opsware.com` server:

```
cd /opsw/Server/~/d04.opsware.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 Opware API.

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

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

The following OCLI 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 OCLI 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 41.

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 OCLI 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 OCLI method returns a value, then the OCLI 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, OCLI 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. OCLI methods output error messages to `stderr`.

Table 2-3: Exit Status Codes for OCLI Methods

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

Table 2-3: Exit Status Codes for OCLI Methods (continued)

EXIT STATUS	CATEGORY	DESCRIPTION
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.opsware.com/method
./getDeviceGroups
cmd_exit_status=$?

if [ $cmd_exit_status -eq 0 ]
then
    echo "The command was successful."
else
    echo "The command failed."
    echo "Exit status = " $cmd_exit_status
fi
```

An OCLI method invokes an underlying API method. If the API method throws an exception, the OCLI 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 OCLI Methods

Many methods in the Opsware 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 `opsware.com` in the `hostname` attribute.

Search Syntax

Methods such as `findServerRefs` have the following syntax:

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

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

```
value-object.attribute operator value
```

(This syntax is simplified. For the full definition, see “Filter Grammar” on page 143)

Search Examples

Most of the SAS 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 OCLI methods, see “Example Scripts” on page 57.

Finding Servers

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

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

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 Table 7-6 on page 141.)

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 Opsware user `jdoue`:

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

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 OCLI methods. (To download the scripts, see “Obtaining the Code Examples” on page 29.) 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 hardcoded object names, such as the `d04.opsware.com` server. For tutorial instructions on creating and running scripts within the OGFS, see step 10 on page 37.

Of the following scripts, the most interesting is `remediate_policy.sh` on page 61. It 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 SAS Web Client. You can create custom fields for servers, device groups, or software policies. To create a custom field, your Opsware user must belong to a user group with the Manage Virtual Columns permission (new in 6.0.1).

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 *Opsware® SAS User’s 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.opsware.com
devices_id='./findServerRefs:i \
filter=\
'device:{ ServerVO.hostname CONTAINS "d04.opsware.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.opsware.com
devices_id='./findServerRefs:i \
filter=\
'device:{ ServerVO.hostname CONTAINS "d04.opsware.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 opsware.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 opsware.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 SAS 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
```

detect_hba_version.sh

This script detects the HBA firmware level of all Unix servers and for each server assigns the level to a custom field. (The HBA is the Host Bus Adaptor, an interface card that connects a host to a storage device.) Before running this script, create a server custom field named `hba_firmware_version` and then create a dynamic device group with a rule that specifies the value of this custom field. After the script runs, the device group is automatically populated with servers that have the specified HBA firmware level.

A future version of Opsware SAS might include the HBA firmware level in the server properties gathered by the Opsware Agent. Until then, you can run this script to fetch the firmware level and store it in a custom field.

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

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

# Native Emulex command that fetches the HBA firmware level:
NATIVE_CMND="/opt/EMLXemlxu/bin/get_fw_rev"

cd "/opsw/Group/Public/All Unix Servers/@/Server"

# Iterate through all Unix servers.
# Run the native command on each server
```

```
# Assign the results of the command to the server's custom
field.
for SERVER in *; do
    FIRMWARE_VER=$(cd $SERVER; rosh -l root "$NATIVE_CMND")
    ./$SERVER/method/setCustomField \
    fieldName=hba_firmware_version strValue="$FIRMWARE_VER"
    echo SERVER = $SERVER FIRMWARE_VER = $FIRMWARE_VER
done
```

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 SAS Client. For examples that search for jobs with OCLI methods, see “Finding Jobs” on page 55.

In this script, in the `create` method of the `SoftwarePolicyService`, the value of the `platforms` parameter is hardcoded. In most of these example scripts, hardcoding is avoided by searching for an object by name. In the case of platforms, searching by the `name` attribute is difficult because it differs from the `displayName` attribute, which is exposed in the SAS 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 hardcodes 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 SAS Client, search for Software by fields such as File Name and Operating System, open the package located by the search, and note the Opware 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 2003 (ID 10007)
./create softwarePolicyVO="{ platforms:i=10007 \
name="TestPolicyA" \
folder:i=$folder_id }"

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 is 4230039.
#
# NOTE: The following command has a bad line break.
# The vo parameter should be on a single line.
#
./update self:i=$policy_id force=true\
# The next 2 lines should be on a single line.
vo='{
softwarePolicyItems:i=com.opsware.pkg.windows.MSISRef:4230039 }'

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 OCLI 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 OCLI methods, see “Finding Jobs” on page 55.

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. 17, 2008.
# In the startDate parameter, note that the last delimiter for
# the time is a period, not a colon.
job_id='./startAudit self:i=140039 \
schedule:s='{ startDate="2008/10/17 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 OCLI Methods

In a future release, the OCLI 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 Opsware API methods are organized into services. To find out what services are available for OCLI methods, enter the following commands in a Global Shell 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 Opsware 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 50.)

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 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 Opware SAS 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 a Global Shell session, to find out if an attribute can be searched on, follow the techniques described in “Searchable Attributes and Valid Operators” on page 56

Chapter 3: Python Access to the API with Pytwist

IN THIS CHAPTER

This chapter contains the following topics:

- Overview of Pytwist
- Setup for Pytwist
- Pytwist Examples
- Pytwist Details

Overview of Pytwist

Pytwist is a set of Python libraries that provide access to the Opware 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 Opware APIs through Pytwist so that end users can invoke the scripts as DSEs or ISM controls. Created by Opware Inc. Professional Services, custom extensions are Python scripts that run in the Command Engine (way). Pytwist enables custom extensions to access recent additions to the Opware SAS 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 Opware SAS data model, custom extensions, Opware 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 *Opsware® SAS Release Notes*.

Pytwist relies on Python version 1.5.2, the version used by Opsware Agents and custom extensions.

Unlike Web Services and Java RMI clients, a Pytwist client relies on internal Opsware SAS libraries. If your client program needs to access the Opsware 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 on Managed Servers

During an Opsware SAS installation or upgrade, the Pytwist libraries are placed on the core server with the Command Engine component. Therefore, you do not need to install Pytwist to use it with custom extensions.

However, Pytwist is not included with the Agent installation. You install Pytwist on a managed server by remediating a policy that contains a Pytwist ZIP file. In the Opsware SAS Client, the Pytwist ZIP files are located in the following folder:

```
/Opsware/Tools/Python Opsware API Access
```

This folder also includes pre-built software policies containing the Pytwist ZIP files for each platform. For example, the policy named Windows Python Opsware API Access contains ZIP files for Windows XP, 2000, 2003, and so forth. When you remediate this policy, only the ZIP file that matches platform version is installed. For example, if you remediate the policy on a Windows 2003 server, only the ZIP file for Windows 2003 is installed.

To install Pytwist on a managed server, perform the following steps:

- 1** In the Opsware SAS Client, under Devices, locate the managed server.
- 2** In the Content pane, open the managed server.
- 3** In the Managed Server window, from the **Actions** menu select **Install Software**.
- 4** In the Install Software window, select the software policy, for example, Windows Python Opsware API Access.

- 5** Click **Install**.
- 6** Step through the Remediate wizard until you get to the Summary Review window.
- 7** Click **Start Job**.

Pytwist Examples

The Python code examples in this section show how to get information from managed servers, create folders, and remediate software policies. To download the examples, see “Obtaining the Code Examples” on page 29.

Each Pytwist example performs the following operations:

- 1** Import the packages.

When importing objects of the Opsware API namespace, 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 75 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 Opsware 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/bin/python
# get_server_info.py

# Search for servers by partial hostname.

import sys
sys.path.append("/opt/opsware/pylibs")
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/bin/python
# create_folder.py

# Create a folder in Opsware SAS.

import sys
sys.path.append("/opt/opsware/pylibs")
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 hardcoded 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 hardcoded 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/bin/python
# remediate_policy.py

# Create, attach, and remediate a software policy.

import sys
sys.path.append("/opt/opsware/pylibs")
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.opsware.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 = platformservice.findPlatformRefs(platformfilter)

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

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

# Create a value object for the new software policy.
vo = SoftwarePolicyVO()
vo.name = policyname
vo.folder = parent

```

```
vo.platforms = platformrefs
vo.softwarePolicyItems = unitrefs

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

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

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 Opware SAS 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 Opware user specified by the `username` parameter, much like an end user who logs onto the Opware SAS 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 Opware 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 Opware Inc. Support.

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 69.) All of the arguments of `TwistServer` are optional. Table 3-1 lists the default values for the arguments.

Table 3-1: Arguments of the `TwistServer` Method

ARGUMENT	DESCRIPTION	DEFAULT
<code>host</code>	The hostname to connect to.	<code>twist</code>
<code>port</code>	The port number to connect to.	1032
<code>secure</code>	Whether to use https for the connection. Allowed values: 1 (true) or 0 (false).	1
<code>ctx</code>	The SSL context for the connection.	None. (See also “Authentication Modes” on page 75.)

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 Opware API method.

Error Handling

If the `TwistServer.authenticate` method or an Opsware 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 Opsware 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 Opsware 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 Opsware 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 Opsware API documentation specifies method parameters and return values as Java data types. Table 3-2 shows how to map the Java data types to Python for the API method invocations in Pytwist.

Table 3-2: Mapping Data Types from Java to Python

JAVA DATA TYPE IN OPSWARE 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 arrays. As output from API method calls, object arrays are returned as Python tuples.
Map	Dictionary
List	Array

Table 3-2: Mapping Data Types from Java to Python

JAVA DATA TYPE IN OPSWARE API	PYTHON DATA TYPE IN PYTWIST
Date	A long data type representing the number of milliseconds since epoch (midnight on January 1, 1970).

Chapter 4: Java RMI Clients

IN THIS CHAPTER

This chapter contains the following topics:

- Overview of Java RMI Clients
- Setup for Java RMI Clients
- Java RMI Example

Overview of Java RMI Clients

A Java Remote Invocation (RMI) client can call the methods of the Opware API from a server that has network access to the Opware core. The server running the client does not have to be an Opware core or managed server. When it connects to the core, the client specifies an Opware user name and password, much like an end user logging on with the Opware SAS Client. The group that the user belongs to determines which Opware resources and tasks are available to the client.

This chapter is intended for software developers who are familiar with Opware SAS fundamentals and the Java programming language.

Setup for Java RMI Clients

Before developing Java RMI clients for the Opware API, perform the following steps:

- 1** Install an Opware SAS 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 J2SE v 1.4.2 SDK.
- 4** Verify that the development server has a network connection to the Opware SAS core server that runs the OCC component.

- 5 Download the `opswclient.jar` file from the Opsware SAS core server to your development server. The `opswclient.jar` file contains the Java RMI stubs for the Opsware API. You include the `opswclient.jar` in the `classpath` option when compiling and running Java RMI clients.

To download `opswclient.jar` specify the following URL, where `occ_host` is the core server running the OCC component:

```
https://occ_host:/twister/opswclient.jar
```

Java RMI Example

This section describes a simple Java RMI client named `GetServerInfo`. To download the source code, see “Obtaining the Code Examples” on page 29.

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 Opsware SAS:

```
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 `opswclient.jar` file, as described in “Setup for Java RMI Clients” on page 79.
- 2 Download the ZIP file that contains the demo program `GetServerInfo.java` file.
- 3 To compile the client, specify the `opswclient.jar` file for the `classpath` option:


```
javac -classpath path/opswclient.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 Opsware SAS:

```
java -classpath .:path/opswclient.jar \  
GetServerInfo [options] target
```

In the following example, `GetServerInfo` connects to Opsware SAS 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 \  
GetServerInfo --host c44.dev.opsware.com --port 443 opsw
```

- 5** Respond to the prompts for the Opsware user name and password. The Opsware user must have read permissions for the servers that match the *target* specified on the command line.

Chapter 5: Web Services Clients

IN THIS CHAPTER

This chapter contains the following topics:

- Overview of Web Services Clients
- Perl Web Services Clients
- C# Web Services Clients

Overview of Web Services Clients

The Opsware 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 Opsware SAS fundamentals and Web Services development.

Programming Language Bindings Provided in This Release

This release of Opsware SAS 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 Opsware 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.opsware.com:443/osapi/com/opsware/folder/  
FolderService
```

```
https://occ.c39.opsware.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 Opware API, Web Services clients must be authenticated and authorized to perform operations in Opware SAS. Communication between clients and the Web Services component in the Opware 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 Opware API has overloaded operations, but the WSDL 2.0 specifications do not support overloading. An overloaded operation in the Opware API is exposed by the Web Service as a single operation.

Java Interface Support

The Opware 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 98.

Unsupported Data Types

The following data types are used by the Opware 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 Opsware 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, Opsware 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.HypervisorInventoryNode.getProperties
com.opsware.virtualization.HypervisorInventoryNode.setProperties
com.opsware.virtualization.VirtualConfigNode.getProperties
com.opsware.virtualization.VirtualConfigNode.setProperties
com.opsware.virtualization.VirtualServerConfig.getProperties
```

```
com.opsware.virtualization.VirtualServerConfig.setProperties
```

Invoke `setDirtyAttributes` When Creating or Updating VOs

Web Services clients must invoke `setDirtyAttributes` before invoking a `create` or `update` method on a service. The `setDirtyAttributes` method explicitly marks the attributes (fields) of a VO that need to be set by the `create` or `update` invocation. The attribute names specified by `setDirtyAttributes` are case sensitive.

For example, to modify the `description` attribute of a `FolderVO` object, the following code invokes `setDirtyAttributes` before it invokes `update`:

```
// fs is FolderService
FolderVO folderVO = fs.getFolderVO(folderRef);
folderVO.setDescription("credit card processing");
folderVO.setDirtyAttributes(new String[]{"description"});
fs.update(folderRef, folderVO, true, true);
```

Invoking `setDirtyAttributes` is required for Web Services clients because of the way Axis deserializes XML objects from XML. If `setDirtyAttributes` is not invoked, Axis calls setters on all attributes of the VO, including read-only attributes, resulting in a `ReadOnlyException`.

Compatibility With Opware Web Services API 2.2

The Opware Web Services API 2.2 is not compatible with the the Opware 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 Opware API, not the Opware Web Services API 2.2.



The Opware Web Services API 2.2 is still supported for Opware SAS 6.x. Clients created for the Opware Web Services API 2.2 will run with Opware SAS 6.x and do not require any modification.

Perl Web Services Clients

This section contains step-by-step instructions and sample code for creating Perl Web Services clients that access the Opware 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

If you are running a recent version of ActiveState Perl on Windows, the only module you need to install is SSL. To install SSL with PPM, perform the following steps:

- 1** Start PPM, either from the Windows Start menu or by entering `ppm.bat` at the command prompt.
- 2** Enter the following command:
`install http://theoryx5.uwinnipeg.ca/ppms/Crypt-SSLeay.ppd`
- 3** Respond to the prompts. The default values should work.

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. To download the file, see “Obtaining the Code Examples” on page 29.
- 2** Edit the `uapisample.pl` file, changing the hardcoded values for `host`, `username`, `password`, and object IDs such as `serverID`.
- 3** Run `uapisample.pl`.

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.opsware.com";
my $port = "443";
my $contextUri = "osapi/com/opsware/";
my $folderServiceName = "folder/FolderService";
my $folderUri = "http://www.opsware.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.opsware.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 $folder = SOAP::Data->name('self') ->attr({ 'xmlns:ns_fs' =>
'http://folder.opsware.com'}) ->type('ns_fs:FolderRef')-
>value(\SOAP::Data->name('id')->type('long') ->value($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.opsware.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.opsware.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.opsware.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 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.opsware.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 Opsware 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 Opsware Command Center component. (For instructions on invoking methods with the Twister, see “API Documentation and the Twister” on page 27.)

- 2 Examine the API documentation to determine the input parameters and return value of the method.

The `getServerVO` method is defined in the interface `com.opsware.server.ServerService`. In the following method signature, note that `getServerVO` accepts a `ServerRef` as a parameter and returns a `ServerVO`:

```
public ServerVO getServerVO(ServerRef self)
    throws java.rmi.RemoteException,
           NotFoundException,
           AuthorizationException
```

- 3 In a browser, specify the following URL to open the WSDL file for the `ServerService`:

`https://occ_host/osapi/com/opsware/server/ServerService?wsdl`

- 4 In the WSDL file, locate the namespace for the `ServerService`:

```
<schema targetNamespace="http://server.opsware.com"
xmlns="http://www.w3.org/2001/XMLSchema" >
```

The following Perl statement (from the code listed previously) specifies the namespace:

```
$self->attr({'xmlns:ns_ss =>
'http://server.opsware.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 Opsware 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 Opsware API

Obtaining the C# Client Stubs

Opsware, Inc. provides a stub file for each service, for example, `FolderService.cs`. All stubs have the same namespace: `OpswareWebServices`. In addition to the stubs, Opsware, Inc. 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 28.

Accessing the C# Stub Documentation

Reference documentation generated by Ndoc is available as a compiled Windows help file that is contained in the same ZIP file as the C# stubs. (Ndoc generates code documentation from the from .NET assemblies and the XML documentation files output by the C# compiler.) This reference documentation contains syntax (but not descriptive) information about the class hierarchy and member method signatures. For descriptions, see the corresponding javadocs as explained in “API Documentation and the Twister” on page 27.

To access the C# stub documentation, perform the following steps:

- 1** Download the `opswcsharpclient.zip` file from the URL shown in the previous section.
- 2** Unzip `opswcsharpclient.zip`.

- 3** In Windows, open the `wscsharpclient.chm` file.

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

To download the ZIP file, see “Obtaining the Code Examples” on page 29.

- 2** Create the following directory:

`C:\wsapi`

- 3** From the Visual Studio.NET 2003 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 file `Class1.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 95.
- 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.NET, from the File menu, select Add Existing Item.
 - Browse to the directory `C:\wsapi\WSAPIDemo`, and select each file, one at a time.
- 9** Add a reference to `System.Web.Services.dll`:

- In Visual Studio.NET, 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.NET, 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 .NET 2003 command prompt:

```
Start ► All Programs ► Microsoft Visual Studio .NET 2003 ►
Visual Studio .NET Tools ► Visual Studio .NET 2003 Command Prompt Change
```

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 Opsware, Inc. 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);
...
}
```


Chapter 6: Pluggable Checks

IN THIS CHAPTER

This chapter contains the following topics:

- Overview of Pluggable Checks
- Setup for Pluggable Checks
- Pluggable Check Tutorial
- Pluggable Check Creation
- Audit Policy Creation
- Document Type Definition (DTD) for config.xml File

Overview of Pluggable Checks

The Opsware SAS Audit and Remediation feature enables you to define and monitor the compliance information for Opsware SAS managed servers. Because compliance standards are continuously evolving, Opsware SAS lets you create specialized custom checks and policies, and extend those provided by Opsware, Inc. 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 SAS Client.

This chapter is intended for software developers who are familiar with XML and with the Audit and Remediation feature of Opsware SAS.

Setup for Pluggable Checks

Before developing pluggable checks, perform the following steps:

- 1** Install an Opsware SAS core in a development environment. Do not use a production core.

- 2 On a server that has an Opware Agent, install OCLI 1.0. For step-by-step instructions, see “Installing OCLI 1.0” in the *Opware® SAS Content Utilities Guide*.

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 101. 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. See “Obtaining the Code Examples” on page 29.

- 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 Opware SAS.

- 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 SAS 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 29 on page 108), when you run the audit in the SAS 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 34 on page 109. 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 Opsware SAS 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 "Installing OCLI 1.0" in the *Opsware® SAS Content Utilities Guide*.

- 14** Open the SAS 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**.

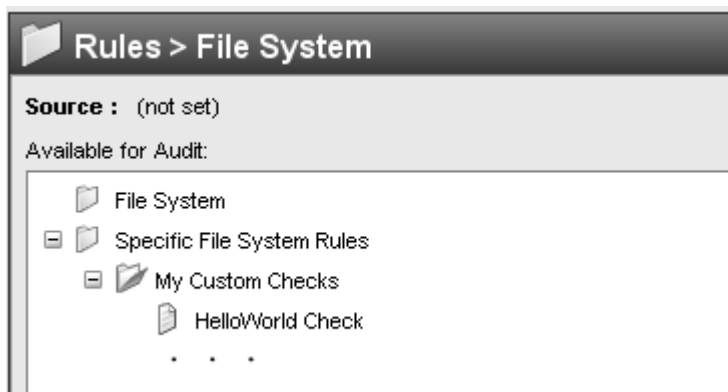
- 17** From the **Actions** menu, select **New**.

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

- 19** In the Views pane, select **Rules > File System**.

The Content pane should list the HelloWorld Check under Available for Audit, as shown in Figure 6-2.

Figure 6-2: HelloWorld Check in the Rules for a File System



- 20** In the `config.xml` file, note the following elements, which are related to the information displayed in Figure 6-2:

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

```
<!-- The category corresponds to the rule hierarchy displayed
by the SAS Client. -->
<checkCategory>File System|My Custom Checks</checkCategory>
```

- 21** In the Audit Window of the SAS 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 6-3.

Figure 6-3: HelloWorld Check Rule Details

The screenshot shows the 'Rule Details: HelloWorld Check' window. It is divided into several sections:

- Description:** A text area containing 'Check that /var/tmp/helloworld exists.' To the right, a 'Test ID' field contains 'helloworld 1'.
- Input Values:** A section with a 'File Name' label and a dropdown menu set to 'Value', with the text '/var/tmp/helloworld' entered in the adjacent field.
- Target Value:** A section with 'Operator: Reference:' and 'Value:' labels. The operator is '=', the reference is 'Value', and the value is 'File exists'.
- Remediation: HelloWorld Check:** A section with a 'Remediation Value' label and a 'File Name' label. The dropdown is set to 'Value' and the text '/var/tmp/helloworld' is entered in the field.

- 22** In the `config.xml` file, examine the following elements, which are related to the information displayed under Rule Details in Figure 6-3:

```
<!-- The following value appears under Description in the
Rule Details of the SAS Client. -->
<checkDefaultDescription>
Check that /var/tmp/helloworld exists.
</checkDefaultDescription>
```

```
<!-- The following element correponds to the Test ID in the
SAS Client. -->
<checkTestID>helloworld 1</checkTestID>

<!-- This label is under Input Values in the SAS Client. -->
<checkGetArgumentDefaultLabel>File Name
</checkGetArgumentDefaultLabel>

<!-- The default argument to the gethelloworld.py script also
appears under Input Values in the SAS Client. -->
<checkGetArgumentDefaultValue>/var/tmp/helloworld
</checkGetArgumentDefaultValue>
```

**23** In the Views pane of the SAS 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.

**24** In the Contents pane, click **Add**.

**25** In the Select Server window, drill down to a server and click **OK**.

**26** 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.

**27** In the Audit window, from the **Actions** menu, select **Run Audit**.

**28** Step through the windows of the Run Audit task.

**29** In the Run Audit window, click **Start Job**.

This action launches the job that runs the `gethelloworld.py` script on the target server.

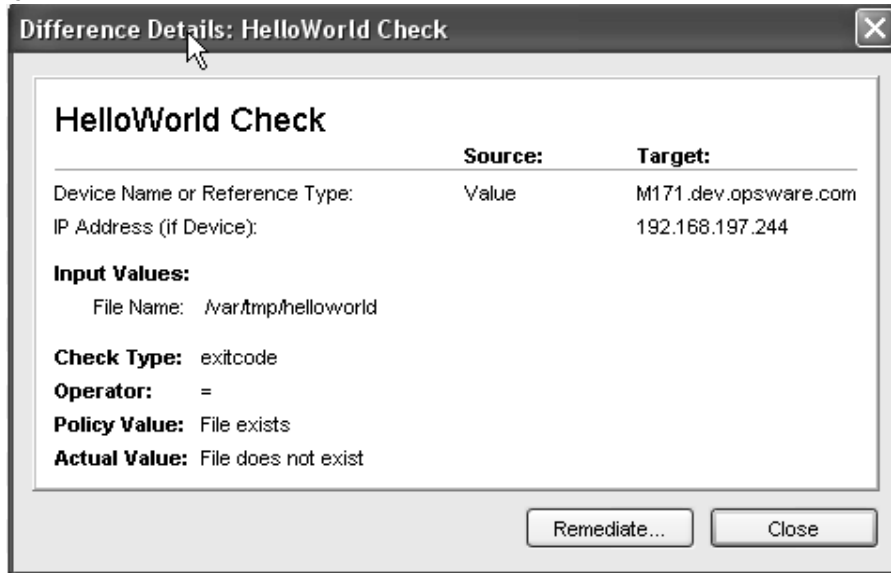
**30** After the job has completed, click **View Results**.

**31** In the Views pane of the Audit Result window, select Policy Rules (1).

**32** In the Content pane of the Audit Result window, open HelloWorld Check.

The Difference Details window should appear, as shown in Figure 6-4.

Figure 6-4: HelloWorld Check Difference Details



- 33** In the `config.xml` file, note the following elements, which are related to the information displayed in the Difference Details window of Figure 6-4:

```
<!-- 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>
```

- 34** 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 “Audit and Remediation” in the online help or the *Opsware® SAS User’s Guide: Application Automation*.

## 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 Opware SAS 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.

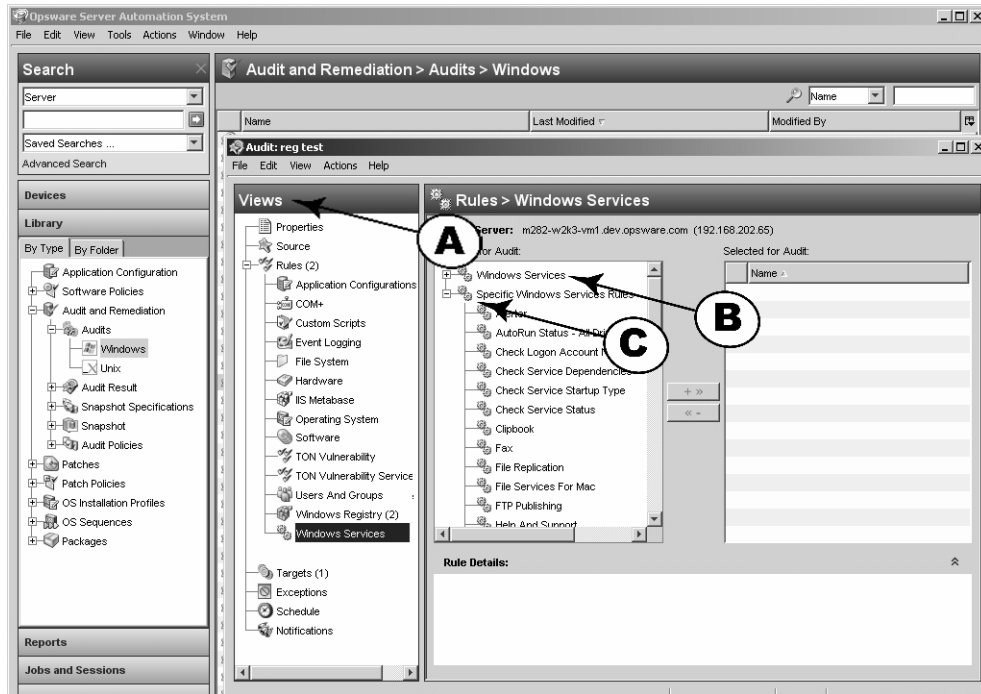
Audit and Remediation also simplifies system administration. For example, you might monitor a class of servers that run a home grown application built by your team, such as a database server or middleware application. As you configure and monitor the servers that run the application, you keep a list 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 6-5 includes the following callouts:

- **Callout A:** Any category listed in the Views panel may have Opware SAS non-modifiable capabilities, or modifiable pluggable checks.
- **Callout B:** This points to the SAS capabilities for dealing with Windows services.
- **Callout C:** This lists pluggable checks for working with Windows Services.

Figure 6-5: Windows Services Audit Rule



Each check evaluates one rule. Several checks can be bundled together into a policy.

The Opware SAS 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 Opware Network (TON). 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 SAS 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 SAS 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 by the Opsware Application Configuration feature, which is based upon the Microsoft Internet Information Services (IIS) Web server configuration file, `UrlScan.ini`. Application Configuration 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. You can use the Application Configuration Markup Language (CML) to manage configuration file values. This is discussed in detail in the *Opsware SAS CML Tutorial*.

---

The Audit and Remediation feature comes with a number of pre-designed 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 1.5.2, 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 6-3.

Table 6-3: Pluggable Check Contents

FILE NAME	DESCRIPTION
<code>config.xml</code>	(Required) The XML configuration file defining how this pluggable check executes, returns, and ultimately reports compliance or non-compliance.
<code>getName. {py   sh   BAT   vbs}</code>	(Required) The audit script, written in Python, VBS, BAT, or shell, that evaluates the audited object, and returns text and exit codes according to the <code>config.xml</code> definitions.
<code>setName. {py   sh   BAT   vbs}</code>	(Optional) The remediation script, written in Python, VBS, BAT, or shell, that remediates the condition checked by the audit script.
<i>Additional Code, Scripts, or Libraries</i>	(Optional) Helper and supplementary scripts used by either the audit or remediation scripts.



The file names for the audit and remediation scripts do not need to begin with `get` and `set`, but this convention simplifies file maintenance.

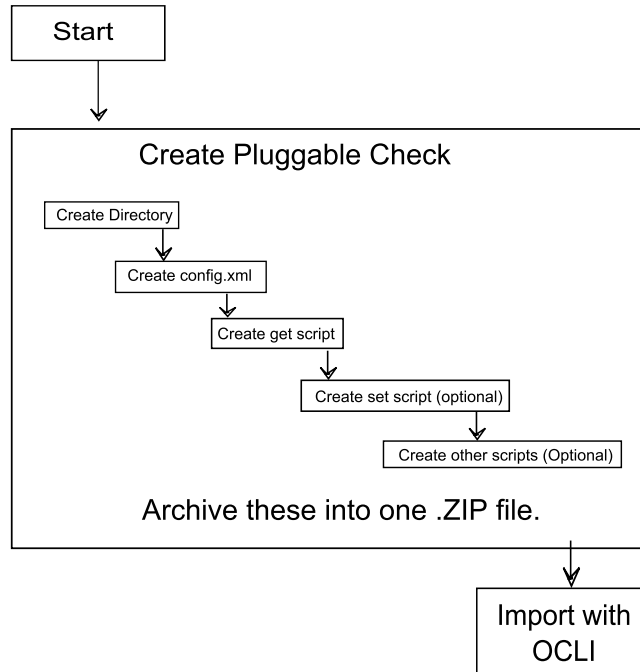
The following example shows a directory structure for a pluggable check:

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

## Development Process for Pluggable Checks

Figure 6-6 shows an overview for the development process, which takes place in a command-line environment.

Figure 6-6: Development Process



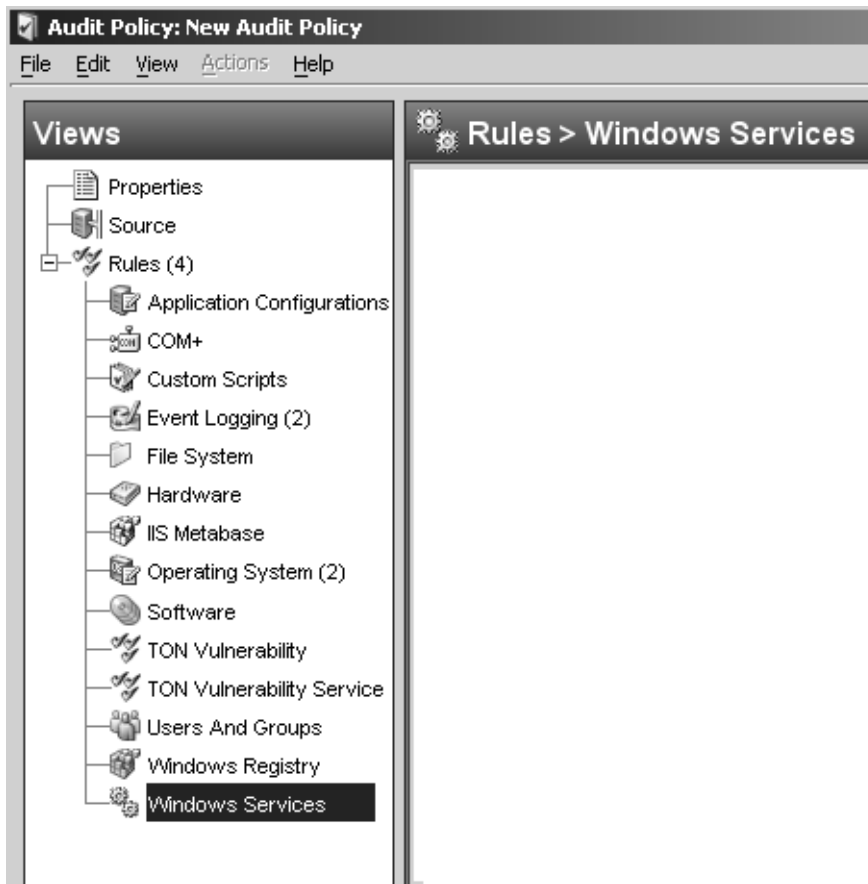
### Pluggable Check Configuration (config.xml)

The config.xml file is a specification file for the pluggable check that contains elements to control how this check appears in the SAS 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 SAS 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 6-7.

Figure 6-7: 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> ${CHECKREMIATIONDESCRIPTION}
</checkRemediationDefaultDescription>
 <checkGetScriptName>${GETSCRIPTNAME}</checkGetScriptName>
 <checkGetScriptType>PY</checkGetScriptType><!-- Or SH for shell,
BAT for Bat, VBS for Visual Basic -->
 <checkSetScriptName>${SETSCRIPTNAME}</checkSetScriptName><!--
Optional -->
 <checkSetScriptType>PY</checkSetScriptType><!-- Optional -->
 <checkVersion>32b.0-1.0</checkVersion>
 <checkReturnTypes>${RETURNTYPE}</checkReturnTypes> <!-- EXITCODE,
STRING, or NUMBER -->
 <checkTestIDs>
 <checkTestID>${CHECKTESTID}</checkTestID> <!-- Optional -->
</checkTestIDs>
 <checkPlatformTypes>
 <checkPlatform>${PLATFORMTYPE}</checkPlatform> <!-- Currently Unix
or Windows -->
</checkPlatformTypes>
 <checkCategories>
 <checkCategory>${CATEGORY}</checkCategory> <!-- Top-level GUI
category -->
</checkCategories>
 <checkGetArguments> <!-- All arguments are optional -->
 <checkGetArgument>
 <checkGetArgumentType>${GETARGTYPE}</checkGetArgumentType> <!--
STRING or NUMBER -->
 <checkGetArgumentDefaultLabel>${GETDEFAULTLABEL}</
checkGetArgumentDefaultLabel>

 <checkGetArgumentDefaultDescription>${GETDEFAULTDESCRIPTION}</
checkGetArgumentDefaultDescription>
 <checkGetArgumentDefaultValue>${GETDEFAULTVALUE}</
checkGetArgumentDefaultValue>
 </checkGetArgument>
 </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 123.

## 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 SAS Client when adding a set script to an already existing check.



Make sure your remediation scripts exit with exitcode 0 to indicate success. All other exitcodes 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.

---

### Zippping 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 Opsware core or mesh using the OCLI 1.0 utility, which is documented in the *Opsware® SAS 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

```

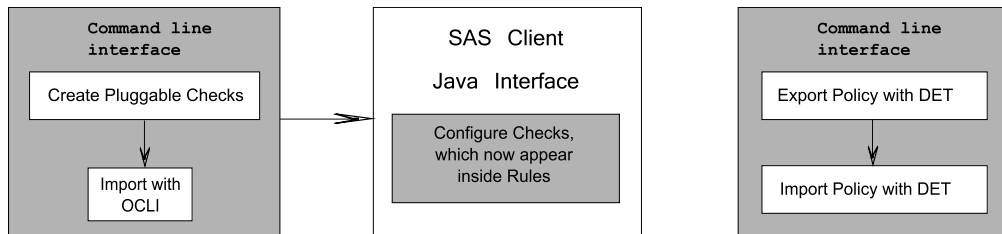


The `oupload` command uses "SunOS 5.8" to specify that the check falls into the generic Unix category in the SAS Client. To specify a check for the Windows category, use "Windows 2003."

## Audit Policy Creation

The audit policy creation procedure is illustrated in Figure 6-8 below:

Figure 6-8: 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 SAS Client. Figure 6-9 shows a list of the audit rules available on a model system.

Figure 6-9: List of Audit Rules

Name	Last Modified	Modified By
test 141560	Mon Dec 04 19:22:19 2006	tdolinsky
test 141555	Fri Dec 01 01:03:12 2006	paul
hello world	Mon Nov 20 21:56:41 2006	tsmedley
test double check unix	Fri Nov 10 19:05:54 2006	gshort
test audit new	Tue Nov 07 18:11:26 2006	ppeng
test 10.129.0.12	Fri Nov 03 00:09:42 2006	gshort
test 140570	Tue Oct 31 17:15:41 2006	gshort
SPARC printconf	Fri Oct 27 20:59:24 2006	nhansen
asimov audit policy	Mon Oct 16 14:54:38 2006	tdolinsky
test 14155	Mon Oct 02 22:01:52 2006	nhansen
nhansen policy test	Mon Oct 02 21:29:39 2006	nhansen
random	Thu Sep 21 23:19:47 2006	gshort
test pluggable	Fri Sep 01 22:54:00 2006	gshort

For detailed information on creating an audit policy, see the “Audit and Remediation” chapter in the *Opware® SAS 6.1 User's Guide: Application Automation*.

## Exporting the Audit Policy

To move a new audit policy to other Opsware 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 Opsware core with content, such as policies, from an existing core. For detailed instructions on this procedure, see the *Opsware® SAS 6.1 Content Utilities Guide*.

## Document Type Definition (DTD) for config.xml File

This file governs SAS Client display names and descriptions, default values, comparisons to be performed upon values returned by the check code, the category of the SAS 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 Opsware SAS:

```
<!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 6-4: DTD Elements and Attributes

ELEMENTS	ATTRIBUTES
checkConfiguration version	Set to 1.0, only change if the Audit and Remediation framework requires it.
checkName	The English name that displays in the SAS Client for the check/rule.

Table 6-4: DTD Elements and Attributes (continued)

checkGUID	<p>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.</p> <p>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).</p>
checkDefaultDescription	Displays in the SAS Client description box. Honors hard carriage returns and HTML. With HTML, the HTML tags need to be converted with &lt; and &gt;.
checkRemediationDefaultDescription	Displays in the SAS 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 SAS 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.

Table 6-4: DTD Elements and Attributes (continued)

<code>checkReturnType</code>	<p>Permissible values are EXITCODE, STRING, or NUMBER:</p> <p>EXITCODE – Standard script return via Wscript.Quit(), exit, return, etc.</p> <p>NUMBER – Audit and Remediation framework will grab from stdout and interpret it as numeric type.</p> <p>STRING – Audit and Remediation framework will grab from stdout and interpret as a string type.</p>
<code>checkTestIDs</code>	List of test IDs.
<code>checkTestID</code>	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 SAS Client, so be consistent in naming it to correspond with the TON Content.
<code>checkPlatformTypes</code>	List of valid platform types for a check.
<code>checkPlatform</code>	WINDOWS   UNIX (or both as individual elements).

Table 6-4: DTD Elements and Attributes (continued)

<p><code>checkExclusivePlatforms</code></p>	<p>List of exclusive platforms. Audit and Remediation currently separates things by Windows or Unix by default, but real world standards as well as limitations and/or differences across operating systems do not make this always desirable. You can limit Audit and Remediation to any platform specified by a platform ID retrieved from the spin.</p> <p>Supported platform IDs include, but are not limited to:</p> <ul style="list-style-type: none"> <li># Red Hat Enterprise Linux AS 2.1 (ID 960007)</li> <li># Red Hat Enterprise Linux AS 3 (ID 430007)</li> <li># Red Hat Enterprise Linux AS 3 IA64 (ID 30100)</li> <li># Red Hat Enterprise Linux AS 3 X86_64 (ID 10773)</li> <li># Red Hat Enterprise Linux AS 4 (ID 40099)</li> <li># Red Hat Enterprise Linux AS 4 X86_64 (ID 10099)</li> <li># Red Hat Enterprise Linux ES 2.1 (ID 10730013)</li> <li># Red Hat Enterprise Linux ES 3 (ID 10720013)</li> <li># Red Hat Enterprise Linux ES 3 IA64 (ID 40100)</li> <li># Red Hat Enterprise Linux ES 3 X86_64 (ID 10774)</li> <li># Red Hat Enterprise Linux ES 4 (ID 50099)</li> <li># Red Hat Enterprise Linux ES 4 X86_64 (ID 20099)</li> <li># SunOS 5.10 (ID 30007)</li> <li># SunOS 5.10 X86 (ID 10044)</li> <li># SunOS 5.6 (ID 130000)</li> <li># SunOS 5.7 (ID 90000)</li> <li># SunOS 5.8 (ID 150001)</li> <li># SunOS 5.9 (ID 920007)</li> <li># Windows 2000 (ID 120000)</li> <li># Windows 2003 (ID 10007)</li> <li># Windows 2003 x64 (ID 60100)</li> <li># Windows XP (ID 10008)</li> </ul>
<p><code>checkExclusivePlatform</code></p>	<p>Individual platform ID.</p>
<p><code>checkExcludePlatforms</code></p>	<p>List of excluded platforms. If the PlatformType claims UNIX, you can supply platform IDs to exclude from the UNIX set (all Linux + all Unixes).</p>



Table 6-4: DTD Elements and Attributes (continued)

<code>checkExcludePlatform</code>	Individual platform ID
<code>checkCategory</code>	<p>This is the SAS Client Category that a check displays in. Currently, a check can only display in a single category. If a category does not exist, it will be created upon upload. The following standard categories for existing checks should be used where possible:</p> <ul style="list-style-type: none"> <li>Event Logging</li> <li>File System</li> <li>Operating System</li> <li>Operating System Domain Controller (sub-category)</li> <li>Operating System Network (sub-category)</li> <li>Registry</li> <li>Services</li> <li>Users and Groups</li> </ul>
<code>checkGetArgument</code> <code>(checkGetArgumentType,</code> <code>checkGetArgumentDefaultLabel,</code> <code>checkGetArgumentDefaultDescription,</code> <code>checkGetArgumentDefaultValue?</code> <code>,</code> <code>checkGetArgumentValidValues?)</code> <code>&gt;</code>	Specifies parameters to the get script.
<code>checkGetArgumentType</code>	NUMBER   STRING
<code>checkGetArgumentDefaultLabel</code>	SAS Client tag next to the input box or drop-down.
<code>checkGetArgumentDefaultDescription</code>	Hover text with further explanation.
<code>checkGetArgumentDefaultValue</code>	Default value for this get parameters.
<code>checkGetArgumentValidValue</code> <code>(checkGetArgumentValidValueItem,</code> <code>checkGetArgumentValidValueDisplayName</code>	<code>checkGetArgumentValidValueItem (#PCDATA)&gt;</code> <code>checkGetArgumentValidValueDisplayName (#PCDATA)&gt;</code>

Table 6-4: DTD Elements and Attributes (continued)

<code>checkGetArgumentValidValues</code> <code>(checkGetArgumentValidValue+)</code>	(Optional) Useful for limiting the parameters for example to 0/disable and 1/enable.
<code>checkSetArguments</code> <code>(checkSetArgument+)</code>	<code>checkSetArgument</code> ( <code>checkSetArgumentType</code> , <code>checkSetArgumentDefaultLabel</code> , <code>checkSetArgumentDefaultDescription</code> , <code>checkSetArgumentDefaultValue?</code> , <code>checkSetArgumentValidValues?</code> )  setArgument elements are identical to the GetArguments, but for the remediation/set script if it exists.  The exception is:  <code>checkSetArgument populateFromRule</code> – 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.
<code>checkSetArgumentType</code>	NUMBER   STRING
<code>checkSetArgumentDefaultLabel</code>	SAS Client tag next to the input box or drop-down.
<code>checkSetArgumentDefaultDescription</code>	Hover text with further explanation.
<code>checkSetArgumentDefaultValue</code>	Default value for this set parameter.
<code>checkSetArgumentValidValues</code> <code>(checkSetArgumentValidValue+)</code> >	
<code>checkSetArgumentValidValue</code> <code>(checkSetArgumentValidValueItem,</code> <code>checkSetArgumentValidValueDisplayName)</code> >	<code>checkSetArgumentValidValueItem (#PCDATA)&gt;</code> <code>checkSetArgumentValidValueDisplayName (#PCDATA)&gt;</code> <code>checkSetArgumentValidValueItem (#PCDATA)&gt;</code> <code>checkSetArgumentValidValueDisplayName (#PCDATA)&gt;</code>
<code>checkSetArgumentValidValueItem</code>	(Optional) Useful for limiting the parameters for example to 0/disable and 1/enable.

Table 6-4: DTD Elements and Attributes (continued)

<code>checkSetArgumentValidValueDisplay</code> <code>playName</code>	
<code>&lt;!ELEMENT</code> <code>checkComparisonDefaults</code> <code>(checkComparisonDefaultOperator?,</code> <code>checkComparisonDefaultValues)</code> <code>&gt;</code>	<code>checkComparisonDefaultOperator</code> not – negation of operator specified, TRUE   FALSE <code>checkComparisonDefaultOperator</code> caseInsensitive – only valid for STRING types.
<code>&lt;!ELEMENT</code> <code>checkComparisonDefaultOperator</code> (#PCDATA) <code>&gt;</code>	List of default values for comparator. Useful for field or development outside the TON build framework.
<code>checkComparisonDefaultValues</code> <code>(checkComparisonDefaultValue+</code> <code>)</code>	<code>checkComparisonDefaultValue</code> ( <code>checkComparisonDefaultValueItem</code> , <code>checkComparisonDefaultValueDisplayName</code> ).
<code>checkComparisonDefaultValueItem</code>	Value for default, passed to code.
<code>checkComparisonDefaultValueDisplayName</code>	Display name for the value, seen in the SAS Client.
<code>checkCompareValidValues</code> <code>(checkCompareValidValue+)</code> <code>&gt;</code>  <code>checkCompareValidValue</code> <code>(checkCompareValidValueItem,</code> <code>checkCompareValidValueDisplayName)</code> <code>&gt;</code>  <code>checkCompareValidValueItem</code> <code>(#PCDATA)</code> <code>&gt;</code>  <code>checkCompareValidValueDisplayName</code> <code>(#PCDATA)</code> <code>&gt;</code>	

Table 6-4: DTD Elements and Attributes (continued)

<pre>checkSuccessExitCodes (checkSuccessExitCode+) checkSuccessExitCode (checkSuccessExitCodeValue, checkSuccessExitCodeDefaultDe scription, checkSuccessExitCodeDefaultDi splayName) &gt;</pre>	<p>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 SAS Client, as well as in reporting.</p>
<pre>checkSuccessExitCodeValue</pre>	<p>Value for script completion, for example, 0 (for <i>disabled</i> typically).</p>
<pre>checkSuccessExitCodeDefaultDe scription</pre>	<p>Hover text for the DisplayName/Value.</p>
<pre>checkSuccessExitCodeDefaultDi splayName</pre>	<p>Value or text shown to user for this value, for example, Disabled.</p>

# Chapter 7: Job Approval Integration

## IN THIS CHAPTER

This chapter contains the following topics:

- Overview of Job Approval Integration
- The Opsware PAS Connector
- Managing Blocked Jobs With the Opsware API

## Overview of Job Approval Integration

An Opsware SAS job is a major task such as Remediate Policies, Install Patch, and Run OS Sequence. When launching a job in the SAS Client, you specify when a job runs: immediately, once in the future, or repeatedly in the future. The SAS Client displays the status of jobs in the Job Logs, Recurring Schedules, and Job Status windows. For example, if a job will run once in the future, in the Job Logs window the status is Scheduled.

In many IT environments, operations such as those performed by Opsware SAS jobs must be approved and assigned tickets before they can be executed. Opsware SAS includes a connector that communicates with Opsware Process Automation System (PAS), which can automate the workflow for approving jobs and tracking tickets. This chapter explains how to set up Opsware SAS so that certain types of jobs wait for approval before executing. It also explains how to configure the connector to run an Ops flow that approves blocked jobs.

This chapter is intended for system integrators and software developers who are familiar with Opsware SAS jobs, Ops flows, and ticketing systems.

## Scenario for Job Approvals

This scenario is just one example of how end users might participate in a job approval process that has been integrated with Opsware SAS. In this scenario, Sam is a system administrator responsible for managing Linux servers in a data center. Cheryl is a member of the IT organization's Change Review Board. She is responsible for approving change requests from the Linux system administrators.

- 1** Sam logs onto the SAS Client and goes to the compliance dashboard. He notices that one of the Linux servers is non-compliant because it needs an RPM that is specified by a software policy.
- 2** To install the RPM on the server, Sam remediates the software policy, choosing to run the job immediately. After Sam clicks Start Job, the SAS Client displays the job status as Pending Approval.
- 3** Cheryl logs onto BMC Remedy Help Desk and searches for recent change requests. The search results include the remediation job launched by Sam. Cheryl goes to the ticket associated with the remediation job. In the ticket details, she notes the server name, the type of job, and the user, Sam.
- 4** Cheryl decides that the remediation can be applied now, so she approves the job.
- 5** In the SAS Client, Sam goes to the Job Logs window and locates his remediation job. He notices that the job has a ticket ID that its status is now In Progress.
- 6** A few minutes later, Sam receives an email notifying him that the job has completed successfully. In the SAS Client, the status of the job is Completed.
- 7** In BMC Remedy Help Ticket, Cheryl checks the ticket and sees that it has been closed and that the remediation was successful. The ticket details include information about the job's results, such as the start and end times, the name of the changed, and the RPM installed on the server.

## Behind the Scenes

While Sam and Cheryl interact with the UIs in the preceding scenario, Opsware SAS and Opsware PAS perform the following operations behind the scenes:

- 1** When Sam starts the job, Opsware SAS verifies that the job type, Remediate Policies, is one of the job types that require approval. Opsware SAS sets the job status to Pending Approval. At this point, the job is blocked and will not run until it has been approved.

- 2** Opsware SAS runs the PAS connector, which reads a configuration file, connects to Opsware PAS, and executes an Ops flow, passing along the job ID.
- 3** The flow invokes the `JobService.getJobInfoVO` method of the Opsware API. From the value object (VO), the flow gets information such as the servers associated with the job, the type of job, and the user who started the job.
- 4** The flow creates a help ticket and fills in the ticket details with the job information.
- 5** The flow invokes the `JobService.updateBlockedJob` method, assigns the ticket ID to the job, and then ends
- 6** Cheryl approves the job in BMC Remedy Help Ticket, an action that invokes a new Ops flow.
- 7** The flow invokes the `JobService.approveBlockedJob` method.
- 8** Opsware SAS runs the remediation job, setting the job status to In Progress.
- 9** The remediation job installs the missing RPM on the server.
- 10** After the job finishes, Opsware SAS sets job the status to Completed.
- 11** The flow invokes the `JobService.getResult` method and determines that the job has completed successfully. The flow updates the ticket details with the job results and then closes the ticket.

## The Opsware PAS Connector

The PAS connector is the software in the Opsware core that communicates with Opsware PAS when an Opsware SAS job is blocked (Pending Approval). The connector resides on the core server running the Command Engine. By default, the connector is not enabled. For instructions on setting up the connector, see “Configuring Opsware SAS for Job Approval Integration” on page 136.

### Prerequisites for the PAS Connector

Make sure that your environment meets the following requirements:

- Opsware SAS is version 6.5 or later.
- Opsware PAS is version 2.1 or later.
- Opsware PAS is installed on a server that has network connectivity to the Opsware SAS core.

- The flow for approving Opsware SAS jobs is installed and tested on Opsware PAS.
- Port 8443 on the Opsware PAS server is open. This port number is configurable, as described in Table 7-5.
- The Opsware SAS user specified by the Ops flow has the required Opsware SAS permissions: Edit All Jobs and View All Jobs. For instructions on setting up permissions, see the *Opsware® SAS Administration Guide*.

Before configuring the connector, gather the following information:

- Host name or IP address of the server running Opsware PAS.
- Name (path in the Library) of the Ops flow that approves the Opsware SAS jobs.

### **Configuring Opsware SAS for Job Approval Integration**

This section explains how to set up Opsware SAS for job approval integration with the PAS connector. For instructions on configuring Opsware PAS and creating Ops flows, see the Opsware PAS technical documentation.

In a multimaster mesh, perform steps 2 - 4 on every Command Engine server in the mesh. Perform step 5 one time for the entire mesh.

To set up job approval integration, perform the following steps:

- 1** Review “Prerequisites for the PAS Connector” on page 135.
- 2** As `root`, log onto the Opsware core server running the Command Engine.
- 3** In a text editor, open the the PAS connector configuration file (`iconclude.conf`), edit the required properties, and save the file. Initially, you can create `iconclude.conf` by copying `iconclude.conf.samp`. For details, see “PAS Connector Configuration File” on page 136.
- 4** Remain logged on as `root` and create the password file (`iconclude.pwd`), as described in “Securing the PAS Password” on page 138.
- 5** Log onto the SAS Client and follow the instructions in “Enabling Job Approval Integration for Opsware SAS” on page 138.

### **PAS Connector Configuration File**

This text file contains name-value pairs that specify properties such as the PAS host and flow. The configuration file resides on the Command Engine server at the following location:



```
/etc/opt/opsware/iconclude-connector/iconclude.conf
```

In the following example of the `iconclude.conf` file, the first line indicates that the PAS connector is enabled:

```
iconclude.enabled: 1
iconclude.host: flowmaster.opsware.com
iconclude.port: 8443
iconclude.proto: https
iconclude.flow.approve: Library/Test Flows/Approve SAS Job
iconclude.user: iconclude
```

Opware SAS includes a sample configuration file, `iconclude.conf.samp`, which you can copy to `iconclude.conf`. During an Opware SAS upgrade, `iconclude.conf.samp` is replaced but `iconclude.conf` is unchanged.

Table 7-5 describes all properties of the `iconclude.conf` file. Required properties are noted in the Default column of the table.

Table 7-5: PAS Connector Configuration File

PROPERTY	DEFAULT	DESCRIPTION
<code>iconclude.enabled</code>	0	An integer, either: 1 - enable the connector 0 - disable the connector
<code>iconclude.host</code>	None (required)	Host name or IP address of the Opware PAS server.
<code>iconclude.proto</code>	https	Protocol (http or https) for connecting to the Opware PAS server.
<code>iconclude.port</code>	8443	Port of the Opware PAS listener.
<code>iconclude.flow.approve</code>	None (required)	The name (path) in the Opware PAS Library of the flow that is run when an Opware SAS job requires approval.
<code>iconclude.user</code>	None (required)	The Opware PAS user name.
<code>iconclude.password</code>	The encrypted password in the <code>iconclude.pwd</code> file. See "Securing the PAS Password" on page 138.	The clear text password of the Opware PAS user. Do not include this property in a production environment.

## Securing the PAS Password

The PAS connector needs a PAS user name and password for authentication. You specify the user name in the `iconclude.conf` file, as described previously. Although you can also specify the password in `iconclude.conf`, this approach is not secure because the contents of `iconclude.conf` are in clear text.

To secure the PAS password, perform the following steps:

- 1** As root, log onto the Opware core server running the Command Engine.  
  
If the `iconclude.conf` file contains a line with the `iconclude.password` property, delete that line from the file.
- 2** Create the directory that will contain the `iconclude.pwd` file:  

```
mkdir -p /var/opt/opware/crypto/iconclude-connector/
```
- 3** Enter a password in `iconclude.pwd`. The following command, for example, enters the password `secret`:  

```
echo -n "secret" > \
/var/opt/opware/crypto/iconclude-connector/iconclude.pwd
```

  
At this point, the password in `iconclude.pwd` is in clear text. However, the next time the PAS connector runs, the password in `iconclude.pwd` is encrypted.
- 4** Change the access to the directory containing `iconclude.pwd`:  

```
chmod -R go-rwx /var/opt/opware/crypto/iconclude-connector
```

## Enabling Job Approval Integration for Opware SAS

The steps in this section require the Manage Approval Integration permission of Opware SAS.

To enable job approval integration and to select the types of jobs that require approval, perform the following steps:

- 1** In the SAS Client, from the Navigation panel select **Opware Administration** > **Approval Integration**.
- 2** Select Enable Approval Integration.
- 3** Under Job Types Requiring Approval, select Yes for one or more types.
- 4** Click **Apply**.

After you have performed the steps in this section, new jobs of the types you select in step 3 are blocked until they are either approved or canceled. In the SAS Client, the status of a blocked job is Pending Approval. Because jobs cannot be approved from within the

SAS Client, be sure to set up the PAS connector. Otherwise, the jobs launched by your end users will stay blocked and will not run. In a testing environment, you can approve a job by starting a Global Shell session and invoking the method described in “Updating Blocked Jobs” on page 140.

### Troubleshooting the PAS Connector

If Opware SAS cannot contact Opware PAS because of incorrect settings in the `iconclude.conf` file, error messages are logged in the following file on the Command Engine server:

```
/var/log/waybot/waybot.err
```

The error messages do not appear in the SAS Client.

### Managing Blocked Jobs With the Opware API

In the API, `JobService` provides the following methods for managing blocked jobs:

```
approveBlockedJob
updateBlockedJob
cancelScheduledJob
findJobRefs
```

These methods are the callbacks into Opware SAS that enable job approval integration. For example, an Ops flow can specify the `approveBlockedJob` method for the `ssh` command.

To run the first three methods in the list, the Opware SAS user must have the Edit All Jobs and View All Jobs permissions. For the `findJobRefs` method to return jobs launched by other users, the user invoking `findJobRefs` needs the View All Jobs permission.




---

In the SAS Client, the status of a blocked job is Pending Approval.

---

### Approving Blocked Jobs

To approve (unblock) a job, invoke the `JobService.approveBlockedJob` method. The end users of the SAS Client cannot approve a blocked job. The following example invokes the OCLI method from within a Global Shell session:

```
cd /opsw/api/com/opsware/job/JobService/method
./approveBlockedJob self:i=$job_id
```

## Updating Blocked Jobs

The `JobService.updateBlockedJob` method enables you to change the value of the Ticket ID and Reason fields displayed in the Job Status window of the SAS Client. The end users of the SAS Client cannot change these fields. The TicketID field corresponds to the `userTag` parameter of `updateBlockedJob` and the Reason field corresponds to the `blockReason` parameter. Here's an OCLI example:

```
cd /opsw/api/com/opsware/job/JobService/method
./updateBlockedJob self:i=$job_id userTag=$ticket_id \
blockReason="This type of job requires approval of CMB."
```

## Canceling Blocked Jobs

To cancel a blocked job, invoke the `JobService.updateBlockedJob` method. In the following example, note that the ID parameter is `jobRef`, not `self`:

```
cd /opsw/api/com/opsware/job/JobService/method
./cancelScheduledJob jobRef:i=$job_id \
reason="Job was scheduled to run outside of change window."
```

A job that is currently running (`job_status = "ACTIVE"`) cannot be canceled.

## Searching for Blocked Jobs

To find blocked jobs, invoke the `findJobRefs` method. The following OCLI method call returns the IDs of all blocked jobs:

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

For related examples, see "Finding Jobs" on page 55.

To search for jobs of a particular status with the `findJobRefs` method, specify the `job_status` string in the filter, not the `JobInfoVO.status` integer. Table 7-6 lists the allowed values of the `job_status` searchable attribute. Note that a `job_status` of `BLOCKED` means that the job is Pending Approval, whereas a `job_status` of `PENDING` indicates that the job is Scheduled. The table also lists the corresponding integer values for `JobInfoVO.status`, which you can examine if your client code has already retrieved the VO. In a Java client, you can compare `JobInfoVO.status` with field constants such as `STATUS_ACTIVE`, instead of the integers listed in the table.

Table 7-6: Job Status in Opsware SAS

Value of job_status Searchable Attribute	Value of JobInfoVO. status	Job Status Displayed by SAS Client	Description
ABORTED	0	Command Engine Script Failure	The job has finished running and a Command Engine failure has been detected.
ACTIVE	1	In Progress	The job is currently running.
BLOCKED	11	Pending Approval	The job has been launched, but requires approval before it can run.
CANCELLED	2	N/A	A schedule has been deleted.
DELETED	3	Canceled	The job was scheduled but was later canceled.
EXPIRED	13	Expired	The current date is later than the job schedule's end date, so the job schedule is no longer in effect.
FAILURE	4	Completed With Errors	The job has finished running and an error has been detected.
PENDING	5	Scheduled	The job is scheduled to run once in the future.
RECURRING	12	Recurring	The job is scheduled to run repeatedly in the future.
STALE	10	Stale	
SUCCESS	6	Completed	The job has finished running successfully.
TAMPERED	9	Tampered	
UNKNOWN	7	Unknown	
WARNING	8	Completed With Warnings	The job has finished running and a warning has been detected.
ZOMBIE	14	Orphaned	



# Appendix A: Search Filter Syntax

## IN THIS APPENDIX

This appendix discusses the following topics:

- Filter Grammar
- Usage Notes

### 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 Opsware SAS 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)`