CA Service Operations Insight

Web Services Reference Guide

r3.2

CA Technologies Product References

This document references the following CA Technologies products:

- CA Application Performance Management
- CA Business Intelligence
- CA Clarity™ Project and Portfolio Manager
- CA CMDB
- CA Configuration Automation (formerly CA Application Configuration Manager)
- CA eHealth® Performance Manager (eHealth)
- CA Embedded Entitlements Manager (CA EEM)
- CA Event Integration
- CA Insight™ Database Performance Manager
- CA NSM
- CA Process Automation
- CA Service Desk
- CA Server Automation (formerly CA Spectrum® Automation Manager)
- CA SiteMinder®
- CA Spectrum®
- CA Systems Performance for Infrastructure Managers
- CA SystemEDGE
- CA Virtual Assurance

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: About This Guide

This guide covers the Representational State Transfer (REST) and WS-Management (WS-MAN) web services that are available with CA SOI. The document describes the resources exposed and the available operations to perform against these web services.

This section contains the following topics:

Intended Audience

This guide is intended for product administrators, domain manager administrators, or integration developers who are interested in extending the CA SOI solution by integrating its data with other products through REST and WS-MAN web services.

The guide assumes experience working with these web services and prior knowledge of the related concepts.

Related Publications

The following publications, provided on the installation media and the CA SOI online bookshelf, provide complete information about CA SOI:

Administration Guide

Provides information about administering and maintaining the product after installation.

Connector Guide

Provides general information about connectors and the connector infrastructure and details about creating integrations through custom connectors and other methods.

Event and Alert Management Best Practices Guide

Provides concepts, procedures, and best practices for managing the event and alert stream that CA SOI receives from connectors.

Implementation Guide

Provides information about installing and implementing the product.

Online Help

Provides information about performing tasks in CA SOI user interfaces.

Readme

Provides information about known issues and information that is discovered after the guides were finalized. A CA SOI release may not have a Readme.

Release Notes

Provides information about operating system support, system requirements, database requirements, web browser support, and international support.

Service Modeling Best Practices Guide

Provides procedures and best practices for modeling services including the following methods: service imports, service discovery, and manual service modeling.

Troubleshooting Guide

Provides information and procedures to diagnose and resolve problems with CA SOI.

User Guide

Provides information for nonadministrative users about using the product, such as responding to alerts and viewing reports.

The following publications provide information about CA Catalyst connectors and are located on each downloadable connector package:

<Product Name> Connector Guide

Provides information about a specific CA Catalyst connector, including prerequisites, installation, configuration, and data mapping.

<Product Name> Connector Readme

Provides known issues for a specific CA Catalyst connector and information discovered after the product-specific *Connector Guide* was finalized.

Local Documentation and Online Bookshelf

CA SOI provides access to the documentation locally and online.

Local Documentation

The local documentation is installed in the SOI_HOME\Documentation folder and includes the PDFs for all guides. The online help is also installed with CA SOI and accessed through the Dashboard (PC and Mobile) and USM Web View. The local documentation is updated with specific releases only.

Online Bookshelf

The online bookshelf is on support.ca.com and provides the most current documentation set, which can be updated between releases. The online bookshelf also provides the documentation for the latest supported versions of CA Business Intelligence, CA EEM, and CA Process Automation. For a list of Bookshelf updates, click the Update History link on the Bookshelf.

CA SOI provides access to the online bookshelf in the following locations:

- The Dashboard provides a Bookshelf link.
- The Operations Console provides a menu link under Help, Bookshelf.

Note: If you are unable to access the online bookshelf, contact your system administrator to provide the documentation set PDFs.

Chapter 2: CA SOI REST Web Services

This section provides information about REST web services in CA SOI.

This section contains the following topics:

REST Web Services Overview

This section helps administrators and integration developers understand how CA SOI REST web services work, including the base URL, supported methods, endpoints, and common requirements.

Representational State Transfer (REST) is a client-server architectural style of building applications that leverages the fundamental properties of HTTP to manage objects accessible at a URL. REST architecture and applications are stateless, which means that no client context information is stored between requests. Each request contains all the information necessary to service the request. REST web services are lightweight, HTTP-based, easy to create and use, and have the desirable property of relating the classes of data to each other using hyperlinks. REST web services provide a simple yet powerful mechanism to interact with data. Using these web services, integration developers can configure the product and can make it communicate through the REST interface. They can use REST web services directly to send HTTP requests to the server for the resources they want to manipulate.

CA SOI lets you expose CA SOI data over REST web services. Because of the inherent standards in the REST architecture, CA SOI REST web services make the CA SOI data accessible to many different development environments. Several resources such as CA SOI user interfaces and third-party interfaces can then consume the exposed data. This ability helps integration developers extend the CA SOI solution by integrating its data with other products through REST web services. These interfaces provide an HTTP-based integration point to the CA SOI data, allowing read or write access. Using these web services, you can access the CA SOI data directly from a browser or can integrate it into your own applications. You can use these web services with any language that understands how to manage HTTP integration.

REST web services access resources by using a Uniform Resource Identifier (URI)—a character string that identifies a name or resource on the Internet. An application using REST web services makes an HTTP request to a URI and parses the response. Such identification enables interaction with representations of the resource over a network. Each client-to-server request contains all the information necessary to understand the request, and does not use any stored context on the server.

CA SOI REST web services follow Hypertext As The Engine Of Application State (HATEOAS) principle. This principle implies that the resources that a request returns to the server contain the next state changes the client can navigate as links. The representations of the resources are interrelated using URLs, enabling you to move from one state to another.

This section helps you understand CA SOI REST web services as follows:

- Provides general concepts and common requirements about using CA SOI REST web services
- Provides information about how to retrieve, manipulate, and set data using various CA SOI REST web services
- Defines information that you can pass to the REST interface
- Provides information about how to send web services requests to perform specific tasks and verify that you get a meaningful response

Note: REST web services use the HTTP protocol for communication. Familiarity with both the HTTP protocol and the REST architecture is required. This section, therefore, assumes that you have experience working with REST web services and prior knowledge of related concepts (such as WADL and Hypertext As The Engine Of Application State).

Base CA SOI REST Web Services URL

The base URL for all CA SOI REST web services is as follows:

```
http://server:port/rest/
```

server

Specifies the server where the REST web service is located.

port

Specifies the port number where the REST web service is located. The default port numbers are 7403 (secure) and 7070 (non-secure).

Examples of base URLs are https://ServerABC:7403/rest/ (secure) and http://ServerXYZ:7070/rest/ (non-secure).

Note: By default, the non-secure interface is not allowed with the Basic authentication (user name and password), where the user name and password are sent as a plain text (*Base64* encoded). If you want to allow Basic authentication over the non-secure connection, you can configure the web.xml file. You can always use the non-secure interface with other types of authentications: CA EEM token and JSESSION. However, you can use the secure interface with all three types of authentications: Basic (user name and password), CA EEM token, and JSESSION. For more information about these authentication methods, see the REST Web Services Authentication (see page 17) .

Supported REST HTTP Methods

As an integration developer, you use REST HTTP methods with the REST web service URLs to manage information in your environment. REST HTTP methods help you achieve the following objectives:

- Access and modify associated resources
- Send an HTTP request to the server for the resource that you want to manipulate
- Control the attributes that you want to retrieve using HTTP headers

CA SOI REST web services support the following REST HTTP methods:

POST (Create)

Creates a resource. The web service can respond with data or status indicating a success or failure.

GET (Read)

Performs a query on a resource and retrieves data. The data that is returned from the web service is a representation of the requested resource.

PUT (Update)

Updates an existing resource.

DELETE (Delete)

Removes an existing resource.

Endpoints

REST web services URLs are specific endpoints. Endpoints are types of items that you use with appropriate HTTP request methods to return a list of results or create, update, or delete an item.

For example, the endpoint (used with the GET method) to get a list of links that let you find more information about a specific service (identified by a service ID) is as follows:

```
GET http://server:7070/rest/service/<serviceId>/entry
```

Note: For more information about specific endpoints, see Available CA SOI REST Web Services (see page 23).

Web Application Description Language URL

You can obtain the complete Web Application Description Language (WADL) for CA SOI REST web services by adding *application.wadl?format=xml* to the base URL. The endpoint is used with the GET method as follows:

```
GET http://server:port/rest/application.wadl?format=xml
```

This WADL file outlines the available operations.

Common Requirements

The following requirements are applicable to all the requests:

- CA SOI REST web services require one of the authentication (see page 17) methods for each call.
- CA SOI REST web services produce the output in Atom format. Atom represents a document format that is based on XML, and that describes lists of associated resources. For Atom version, do one of the following tasks:
  - Send the *Accept* header *Accept: application/atom+xml*.
  - Specify the URL query parameter *format=atom*.
- The query parameter *format* takes precedence over the format defined in the *Accept* header.
- CA SOI REST web services follow Hypertext As The Engine Of Application State (HATEOAS) principle. This principle implies that the resources that a request returns to the server contain the next state changes the client can navigate as links.
- The typical output of a CA SOI REST web service request is the Atom feed, which can be *ordered* and *paged* to organize the output properly. For ordering and paging, you can use the following parameters:

  metric

  Specifies the ordering domain. The values depend on the type of the returning object. The Ordering Metric (see page 19) section includes detailed information about what values you can use.

desc

(If true) Returns the result in the descending order.

size

Specifies the size of the page; that is, how many results to show on a single page. The default value is 25. The default value is used when you do not specify any value for the parameter, or you provide an invalid value (for example, a negative value).

start

Specifies the page from where to start; that is, how many results to skip from the beginning. The default value is 0; therefore, *skip zero records* implies start from the beginning.

Note: Examples in this section demonstrate how CA SOI REST web services use create, read, update, and delete HTTP operations on CA SOI objects. Use these examples to understand how each REST operation interacts with CA SOI objects.

REST Web Services Authentication

This section helps administrators and integration developers understand the different authentication methods that the REST web services support and how to customize the acceptable authentication methods.

CA SOI REST web services support authentication in three ways: using a user name and password (Basic authentication), using a CA EEM artifact, or using a JSESSIONID. Typical communication with CA SOI REST web services is authenticated through the user name/password or CA EEM artifact in the first call, and then using the JSESSIONID for all other subsequent calls. The last call is the logout call that invalidates the session.

Therefore, for the first call, use one of the two methods: user name/password or CA EEM artifact. This first call returns the JSESSIONID, which the clients can then start using for making subsequent calls. The JSESSIONID authentication is the best performance authentication method, because it does not require the REST web service to verify the password or CA EEM artifact against CA EEM.

Additional information about these authentications is as follows:

- Using the user name and password (Basic authentication)

  The user name and password are sent in the form of the Basic HTTP authentication in the HTTP header. The password is encoded by using *Base64* encoding and can be easily decoded; therefore, this type of authentication is supported over a secure SSL channel (HTTPS protocol). For a typical CA SOI installation, it is port 7403, for example, https://server:7403/rest/.

  Note: Only CA EEM users have access to CA SOI REST API. Therefore the “samuser” user is not able to access the REST API.

- Using the one-time CA EEM artifact

  In this type of authentication, the CA EEM artifact can be obtained by calling the SafeSession.exportSession() method.

  You can use this type of authentication with both the protocols—HTTP and HTTPS.

  Note: For more information about CA EEM sessions, see the CA EEM documentation.

- Using the JSESSIONID

  This type of authentication sends the JSESSIONID in the cookie header parameter. Both the previously mentioned authentication methods (user name/password and CA EEM artifact) set a cookie with the JSESSIONID value. The JSESSIONID can then be used in all subsequent calls through HTTP or HTTPS protocol.

  The session expires after 30 seconds of inactivity or after a call to the logout endpoint http://server:port/rest/logout.

Note: You can also configure authentication-related security filter parameters in the web.xml configuration file. For more information, see Configure the Authentication-related Security Filter Parameters.

Configure Authentication-Related Security Filter Parameters

The web.xml file includes authentication-related security filter parameters. You can configure the values of these parameters based on your unique IT environment requirements. This file is available in the folder SOI_HOME\SAMUI\webapps\rest\WEB-INF.

Note: SOI_HOME represents the location where CA SOI is installed; for example, C:\Program Files\CA\SOI.

Follow these steps:

1. Locate the web.xml file in the folder SOI_HOME\SAMUI\webapps\rest\WEB-INF.
2. Open the file using a text editor and search for the following section in the file:

   `<filter-name>`**`SecurityFilter`**`</filter-name>`

   This section includes three parameters that you can configure.
3. Specify the appropriate value in the *<param-value>...</param-value>* field of the following parameters:

   allowPlainCredentials

   Specifies whether to allow Basic authentication over the non-SSL connection. Possible values are *true* and *false*. The default value is *false*, which implies that the Basic authentication is not allowed over the non-SSL connection.

Default: false

Note: If the REST interface is accessed using the loopback address (127.0.0.1 or ::1), the Basic authentication is always allowed.

cacheCredentials

Specifies whether the REST interface must cache the user credentials instead of verifying the password for each request. When you cache the user credentials, the password verification process becomes faster. Therefore, when you use the user name/password authentication method for all calls, the caching improves the response of the REST interface, because the CA EEM call is skipped.

Possible values are *true* and *false*. The default value is *true*, which implies that the interface must cache the user credentials.

Default: true

Note: User credentials are cached for 10 minutes. Therefore, if a user changes the password, the old password remains active in the REST interface until the cached record expires.

sessionInactivateInterval

Specifies the time (in seconds) for which the REST interface must keep the session active. If no request is using the session during the specified time, the client must authenticate again.

Default: 30

4. Review the new information and save the file.

   The changes are saved.

REST Web Services Ordering Metric

This section includes the ordering metric values for REST web services.

The ordering metric specifies the ordering domain of the web service request output to organize the output appropriately. This section includes ordering metrics values for the following output items:

- Service (see page 20)
- Group (see page 20)
- Customer (see page 21)
- CI (see page 21)
- Alert Queue (see page 22)
- Alert (see page 22)

- User (see page 22)
- Escalation Policy (see page 22)
- Escalation Policy Action (see page 23)
- Configuration (see page 23)
- Configuration Node (see page 23)

Note: All metric values are not case-sensitive.

Service Ordering

The following table includes the ordering metric values for Service:

Metric Value	Description
NAME	Ordered by the service name
HEALTH	Ordered by the service health
RISK	Ordered by the service risk
AVAILABILITY	Ordered by the service availability (last 24 hours)
QUALITY	Ordered by the service quality
SLA	Ordered by the service SLA status
PRIORITY	Ordered by the service priority
ALERT_COUNT	Ordered by the down alert count
DOWN_ALERT_COUNT	Ordered by the down alert count
CRITICAL_ALERT_COUNT	Ordered by the critical alert count
MAJOR_ALERT_COUNT	Ordered by the major alert count
MINOR_ALERT_COUNT	Ordered by the minor alert count

Group Ordering

The following table includes the ordering metric values for Group:

Metric Value	Description
NAME	Ordered by the group name

Metric Value	Description
PRIVILEGE_SET	Ordered by the privilege set, which implies that all administrator groups and user groups would be together

Customer Ordering

The following table includes the ordering metric values for Customer:

Metric Value	Description
NAME	Ordered by the customer name
HEALTH	Ordered by the customer health
RISK	Ordered by the customer risk
QUALITY	Ordered by the customer quality
PRIORITY	Ordered by the customer priority
DOWN_ALERT_COUNT	Ordered by the down alert count
CRITICAL_ALERT_COUNT	Ordered by the critical alert count
MAJOR_ALERT_COUNT	Ordered by the major alert count
MINOR_ALERT_COUNT	Ordered by the minor alert count

CI Ordering

The following table includes the ordering metric values for CI:

Metric Value	Description
NAME	Ordered by the CI name
HEALTH	Ordered by the CI health
USM_TYPE	Ordered by the USM type
IP_ADDRESS	Ordered by the IP address
OPERATIONAL_MODE	Ordered by the CI operational mode

Alert Queue Ordering

The following table includes the ordering metric values for Alert Queue:

Metric Value	Description
NAME	Ordered by the alert queue name
ALERT_COUNT	Ordered by the down alert count
DOWN_ALERT_COUNT	Ordered by the down alert count
CRITICAL_ALERT_COUNT	Ordered by the critical alert count
MAJOR_ALERT_COUNT	Ordered by the major alert count
MINOR_ALERT_COUNT	Ordered by the minor alert count

Alert Ordering

The following table includes the ordering metric values for Alert:

Metric Value	Description
SEVERITY	Ordered by the alert severity
TIME	Ordered by the occurrence time

User Ordering

The following table includes the ordering metric values for User:

Metric Value	Description
NAME	Ordered by the user name

Escalation Policy Ordering

The following table includes the ordering metric values for Escalation Policy:

Metric Value	Description
NAME	Ordered by the escalation policy name

Escalation Policy Action Ordering

The following table includes the ordering metric values for Escalation Policy Action:

Metric Value	Description
NAME	Ordered by the escalation policy action name

Configuration Ordering

The following table includes the ordering metric values for Configuration:

Metric Value	Description
NAME	Ordered by the configuration name

Configuration Node Ordering

The following table includes the ordering metric values for Configuration Node:

Metric Value	Description
NAME	Ordered by the configuration node name

Available CA SOI REST Web Services

The following CA SOI REST web services are available:

- Alert Queue (see page 24)
- Alert (see page 25)
- CI (see page 25)
- Configuration (see page 26)
- Connector
- Customer (see page 27)
- Email (see page 28)
- Escalation Policy Action (see page 28)
- Escalation Policy (see page 29)

- Group (see page 29)
- Meta (see page 31)
- Schedule (see page 31)
- Service (see page 31)
- User (see page 32)

Note: The complete documentation for CA SOI REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/

Alert Queue REST Web Services

Alert queues are user-defined alert groups. Alert queues let you group alerts as they come in based on specific criteria to monitor the status of your infrastructure more efficiently.

Using the Alert Queue REST web services, you can perform various operations on alert management queues in CA SOI:

- Create an alert queue
- Get a list of alert queues
- Get a list of alerts in an alert queue
- Get the alert queue definition
- Get hyperlink entries associated with an alert queue
- Get the status information for an alert queue
- Update an alert queue
- Delete an alert queue

The GET, PUT, POST, and DELETE HTTP methods are used to perform these tasks, as appropriate. For example, to delete an alert queue, the HTTP method DELETE is used.

Note: The complete documentation for the CA SOI Alert Queue REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_AlertQueueResource.html

Alert REST Web Services

Alerts are fault conditions that the integrated domain manager reports. Each alert is associated with a CI and contains properties such as severity, a summary of the condition, and when the condition occurred. Alerts are service impacting when they affect a CI that is part of a managed service. They are non-service impacting when they affect CIs that are not part of a managed service.

Using the Alert REST web services, you can perform various alert-related operations in CA SOI:

- Get a list of alerts
- Get hyperlink entries associated with an alert
- Get the alert definition
- Get the status information for an alert
- Get a list of escalation policy actions associated with an alert
- Perform an escalation policy action on a specific alert
- Update an alert
- Delete an alert

The GET, PUT, POST, and DELETE HTTP methods are used to perform these tasks, as appropriate. For example, to delete an alert, the HTTP method DELETE is used.

Note: The complete documentation for the CA SOI Alert REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_AlertResource.html

CI REST Web Services

CIs in CA SOI represent IT elements managed by a domain manager. Each CI belongs to a type (defined in the USM schema) such as ComputerSystem, Database, Process, and Relationship. Connectors transform managed objects from domain managers to adhere to the USM schema and import the objects into CA Catalyst as CIs.

The CI REST web services let you perform the following operations:

- Get hyperlink entries associated with a CI
- Get the status information for a CI
- Get the CI USM information
- Get a list of alerts impacting a specific CI
- Get a list of children for a specific CI

- Get a list of parents for a specific CI
- Get a list of services for a CI

The GET HTTP method is used to perform these tasks.

Note: The complete documentation for the CA SOI CI REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_CIResource.html

Configuration REST Web Services

The CA SOI Configuration REST web services give you the flexibility to view and update integration configuration values as and when required using the programmable interface. You can override the static configuration values (such as CA EEM configuration, email configuration) specified at the time of installation. For example, CA SOI has a number of integrations with various external products. This integration information is stored in configuration files, which can require updates based on some changes in the deployment environment. You can use the Configuration REST web services to view and update these configurations.

You can perform the following tasks using the Configuration REST web services:

- Get a list of configuration nodes
- Get a list of configuration sections for a configuration node
- Get the configuration section definition
- Get hyperlink entries associated with a configuration section
- Update a configuration section

The GET and PUT HTTP methods are used to perform these tasks, as appropriate. For example, to update a configuration section, the HTTP method PUT is used.

Note: The complete documentation for the CA SOI Configuration REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_ConfigurationResource.html

Customer REST Web Services

A customer in CA SOI is any consumer of a managed service. You create customers and associate them with service models to see the impact of service degradation on the service consumer. Customer management provides an extra layer of insight into how end users dependent on provided services are affected when those services experience downtime or degraded performance.

Using the Customer REST web services, you can perform customer-related operations in CA SOI:

- Get a list of customers
- Get hyperlink entries associated with a customer
- Get the status information for a specific customer
- Get a list of services associated with a customer
- Get a list of alerts on all services associated with a customer
- Get a list of subcustomers
- Get information about the parent customer
- Get information about the selected customers
- Get information about services of a specific customer
- Get the customer definition
- Get a list of services associated with a customer (as XML)
- Create a top-level customer
- Create a subcustomer
- Set customer services
- Update a customer
- Delete a customer

The GET, PUT, POST, and DELETE HTTP methods are used to perform these tasks, as appropriate. For example, to delete a customer, the HTTP method DELETE is used.

Note: The complete documentation for the CA SOI Customer REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_CustomerResource.html

Email REST Web Services

The CA SOI Email REST web services give you the option to create and send emails using the REST programming interface. For example, you can create and send emails about alerts from your application using the Email web services.

You can perform the following task using the Email REST web services:

- Send an email

The POST HTTP method is used to perform this task.

Note: The complete documentation for the CA SOI Email REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_EmailResource.html

Escalation Policy Action REST Web Services

Escalation policy action defines the action to perform when the policy criteria are met. For example, you can set an escalation policy action where when an alert matches escalation policy criteria, the alert triggers an action that sends an email to the technician responsible for the affected service.

Using the Escalation Policy Action REST web services, you can perform escalation policy action-related operations in CA SOI:

- Get a list of escalation policy actions
- Get hyperlink entries associated with an escalation policy action
- Get the escalation policy action definition

The GET HTTP method is used to perform these tasks.

Note: The complete documentation for the CA SOI Escalation Policy Action REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_EscalationPolicyActionResource.html

Escalation Policy REST Web Services

Escalation policy automates alert escalation according to user-defined criteria. When the policy criteria are met, a specified escalation action runs. Using the Escalation Policy REST web services, you can perform the following operations:

- Get a list of escalation policies
- Get hyperlink entries associated with an escalation policy
- Get the escalation policy definition
- Create an escalation policy
- Update an escalation policy
- Delete an escalation policy
- Get a list of assigned service IDs for an escalation policy
- Get a list of assigned alert queue IDs for an escalation policy
- Get a list of assigned escalation policy action IDs for an escalation policy
- Get a list of assigned schedule IDs for an escalation policy
- Set a list of escalation policy services
- Set a list of escalation policy alert queues
- Set a list of escalation policy actions
- Set a list of escalation policy schedules

The GET, PUT, POST, and DELETE HTTP methods are used to perform these tasks, as appropriate. For example, to delete an escalation policy, the HTTP method DELETE is used.

Note: The complete documentation for the CA SOI Escalation Policy REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_EscalationPolicyResource.html

Group REST Web Services

A user group in CA SOI includes users having the same level of access privileges. You can access CA SOI user groups using the Group REST web services. The Group REST web services let you perform the following user group-related tasks:

- Get a list of groups
- Get hyperlink entries associated with a group
- Get the status information for a group
- Create a group

- Update a group
- Delete a group
- Get a list of users assigned to a group
- Assign users to a group
- Remove a user from a group
- Get the user definition in the assigned group
- Get the user group access status for all alert queues
- Set the user group access status for all alert queues
- Get a list of specific privileged alert queues for a group
- Set a list of specific privileged alert queues for a group
- Get the user group access status for all services
- Set the user group access status for all services
- Get a list of specific privileged services for a group
- Set a list of specific privileged services for a group
- Get the user group access status for all customers
- Set the user group access status for all customers
- Get a list of specific privileged customers for a group
- Set a list of specific privileged customers for a group
- Get a list of all the privileges for the Administrator role
- Get a list of all the privileges for the Operator role

The GET, PUT, POST, and DELETE HTTP methods are used to perform these tasks, as appropriate. For example, to delete a group, the HTTP method DELETE is used.

Note: The complete documentation for the CA SOI Group REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_GroupResource.html

Meta REST Web Services

Using the Meta REST web services, you can retrieve administrator and operator group privileges:

- Get a definition of the Administrator group
- Get a definition of the Operator group

The GET method is used to perform these tasks.

Note: The complete documentation for the CA SOI Meta REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_MetaResource.html

Schedule REST Web Services

Using Schedule REST web services, you can manage, create, or delete escalation schedules:

- Get a list of schedules
- Create a schedule
- Get schedule details
- Delete a schedule
- Get hyperlink entries associated with a schedule

The GET, POST, and DELETE methods are used to perform these tasks.

Note: The complete documentation for CA SOI Schedule REST web service is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_ScheduleResource.html

Service REST Web Services

Services in CA SOI represent discrete business functions that can contain CIs that multiple domain managers manage. A service typically consists of several CIs, which can be grouped to represent things like web server farms or clusters. Services can also contain subservices, which are subordinate service models. Service models typically represent high-level abstract entities; for example, a web-based retail transaction service, an application server service, and a printing service.

Using the Service REST web services, you can retrieve data about CA SOI services. The HTTP method GET is used to retrieve this information. You make an HTTP request, which uses the GET method, to a specific URL. The web service interprets the method in the URI as the action it must perform. The web service then retrieves the required data from the request and returns the output based on that data.

You can perform the following operations using the Service REST web services:

- Get a list of services
- Get hyperlink entries associated with a service
- Get the status information for a service
- Get the service USM information
- Get the service metric history information
- Get a list of service alerts
- Get a list of children for a service
- Get a list of parents for a service
- Get a list of services based on service IDs
- Get a list of subservices
- Create a service
- Update a service
- Delete a service

The GET, PUT, DELETE, and POST HTTP methods are used to perform these tasks, as appropriate. For example, to retrieve a list of services, the HTTP method GET is used.

Note: The complete documentation for the CA SOI Service REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_ServiceResource.html

User REST Web Services

You can access CA SOI users using the User REST web services. This ability lets you manage CA SOI users; for example, you can create, list, and update users.

The User REST web services let you perform the following user-related tasks:

- Create a user
- Get a list of users
- Get hyperlink entries associated with a user
- Get the user definition
- Get a list of groups associated with a user
- Update a user

The GET, PUT, and POST HTTP methods are used to perform these tasks, as appropriate. For example, to update a user, the HTTP method PUT is used.

User creation and assignment works as follows:

1. The endpoint information in the Create a User section lets you create users in CA EEM. The created users remain in the *unassigned* state. You can list the unassigned users using the *unassigned* parameter while getting a list of users.
2. The created user is assigned to the CA SOI application and the group by using the endpoint information specified in the Assign Users to a Group section.

Therefore, this way, you cannot only create users and add them to a group, but also add existing users (for example, users that are created manually) to a group.

Note: The complete documentation for the CA SOI User REST web services is available at:

https://<ui-server>:<ssl port>/rest/docs/rest/resource_UserResource.html

Chapter 3: REST Web Services Example

This section provides an example use case for leveraging the REST web services in Perl scripts to perform bulk operations such as user and group creation.

This section contains the following topics:

Chapter 4: Calling CA SOI REST Web Services from Perl Scripts

The following section provides information about how to use REST web services to retrieve, post, and process CA SOI data, using a Perl script. Perl scripts are useful for the initial setup of the CA SOI platform because they allow you to automatically enter or process large amounts of data. You can use Perl scripts for making all types of calls, but this document highlights the main ways to embed REST web services calls in your scripts.

This section contains the following topics:

Prerequisites

You must have:

- Perl version 5.14.2.1 or higher installed.

  Note: Verify that the Perl module XML::XPath is installed.

- Access to the Perl Script Example files in the following location:

  *SOI_HOME**tools**examples**REST_Perl_Example*\

Set Up Users in a CA SOI Environment

You can use the Perl script to process an input file and assign users to groups according to the file. This method of entering users into the system and assigning them to groups is useful when you want to process large amounts of data. If a user does not exist in the system, it creates the user. The format of the input file is comma-separated with user name first and group name last on each line.

Example

The following example shows three users being assigned to two user groups, using REST web services from Perl.

```
username1, groupA
username2, groupA
username3, groupB
```

A prerequisite is that all groups exist in the system.

Note: If a group does not exist in the system, the line containing that group is skipped. The script then continues with the next line.

Configure the Perl Script

The Perl script starts with an environment definition constant where you can set up connectivity to REST web services, user credentials, and so on.

Example

The following code snippet shows configuration constants from the beginning of the Perl script.

```
my $headless_server   = 'soi-r2-test06';    # hostname
my $headless_port     = '7070';             # port for plain HTTP access
my $headless_ssl_port = '7403';             # port for HTTPS access
my $headless_user     = 'samuser';         # admin user name
my $headless_password = 'P@ssword01';       # admin user password

my $password_for_created_users ='changeit';   # password for the newly created users
my $page_size = 10000;    # page size in which the data are fetched
                          # larger value more data fetched per one call.
```

Code Overview

PERL code contains detailed comments. This code overview highlights some key components of the PERL script. The following section provides examples that cover all aspects of calling REST web services, such as authentication, retrieving paged data, and parsing and posting data. You can use these examples to help create your own scripts to call REST web services.

You can find the Perl Script Example file in the following location:

*SOI_HOME**tools**examples**REST_Perl_Example*\

HTTP Access

To make HTTP and HTTPS calls, you can use the LWP::Simple module.

Example

The following example shows how to tell Perl to use the LWP::Simple module:

```
use LWP::Simple;  # LWP package to work with HTTP requests
```

The following classes are the most important of the package:

LWP::UserAgent

Behaves as a simple browser, where you can configure basic authentication, cookies, and so on.

LWP::Request

Represents the HTTP request.

Note: The LWP::Request is not required for the HTTP GET method, because the request comprises URL only. However, it is required for HTTP POST and HTTP PUT methods where you pass the URL and the payload.

LWP::Result

The class representing the HTTP response, which contains the result code, http header and body.

HTTP GET Call

Use the PERL script to perform an HTTP GET call.

Example

The following code snippet shows an HTTP GET call.

Note: The code snippet does not contain authentication to keep the example simple.

```
my $browser = LWP::UserAgent->new();                          # create browser
my $response = $browser->get('http://server:7070/rest/group'); # call HTTP get
if ($response->is_success)                                    # check HTTP status
  return $response->content;                                  # return the result
die "Error while calling http get";                           # error
```

SSL Communication and Basic Authentication Method

To use a secure communication channel, you can use SSL communication and a basic authentication method.

Example

The following code snippets shows SSL communication and a basic authentication method:

```
$ENV{PERL_LWP_SSL_VERIFY_HOSTNAME}=0;          # skip SSL certificate verification
my $browser = LWP::UserAgent->new();                           # create browser
$browser->credentials(                                         # BASIC auth
        'server:7403',
        'REST Authentication',                                 # REALM
        'bob' => 'bobspasswd');
my $response = $browser->get('https://server:7403/rest/group'); # call HTTPS get
if ($response->is_success)                                    # check HTTP status
  return $response->content;                                  # return the result
die "Error while callings http get";
```

When you authenticate using user name and password, REST web services returns a cookie with the session ID.

Example

The following code snippet shows the code that reads the cookies and then how they are used.

```
# regular expression to select the cookie
if ($response->header('SET-COOKIE') !~ /(JSESSIONID=\w+);/) {
    die "cookie with jsession not found\n Aborting";
}
```

```
my $cookie = $1;                                    # the first group is the value
```

The cookie can be used for authentication in all other calls by setting the cookie in the request headers.

```
$browser->default_header('Cookie' => $cookie); #set the sessionID as cookie
```

HTTP POST with LWP::Request Creation

In addition to HTTP GET, the PERL code calls HTTP POST in REST web services.

Note: The Post call is similar to the HTTP GET, but it requires that you set the payload by creating an LWP::Request.

Example

The following code snippet shows the creation of an LWP::Request.

```
 my $request = HTTP::Request->new(POST => 'http://server:7070/rest/user';
$request->header('Accept' => 'application/xml');
$request->content("<user>
                      <userName>bob</userName>
                      <password>changeit</password>
                   </user>");
$request->content_type("application/xml; charset=utf-8");
my $response = $browser->request($request);
if ($response->code == 201) {
    # location header has the URL to the newly created resource
    return $response->header('Location');
}
die "Error while calling HTTP_POST"
```

Parsing Response

REST web services responses are typically ATOM feeds which are XML. An easy way to get certain information from XML is to use an XPath expression to query information. To download the XPath module to your PERL environment, enter the following call from your shell command line:

cpan XML::XPath

The call downloads the required module.

You can start using the following module in your script:

```
use XML::XPath;   # XPath to parse XML (Atom) outputs
use XML::XPath::XMLParser;
```

Example

The following ATOM feed shows an example of a response from the REST interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Users</title>
  <link rel="next-page" type="application/atom+xml"
href="http://server:7070/rest/user?desc=false&amp;start=4&amp;size=4"
title="next-page" />
  <author>
    <name>CA SOI</name>
  </author>
  <id>http://server:7070/rest/user?unassigned=true</id>
  <updated>2012-03-29T13:36:17Z</updated>
  <restApi:totalCountHint
xmlns:restApi="http://ca.com/2011/soi/rest">75</restApi:totalCountHint>
  <entry>
    <title>zaiwenuser</title>
    <link rel="entry" type="application/atom+xml"
href="http://server:7070/rest/user/zaiwenuser/entry" title="entry" />
    <id>http://server:7070/rest/user/zaiwenuser/entry</id>
    <published>2012-03-29T13:36:17Z</published>
  </entry>
  <entry>
    <title>zaiwen400</title>
    <link rel="entry" type="application/atom+xml"
href="http://server:7070/rest/user/zaiwen400/entry" title="entry" />
    <id>http://server:7070/rest/user/zaiwen400/entry</id>
    <published>2012-03-29T13:36:17Z</published>
  </entry>
  <entry>
    <title>zaiwen300</title>
    <link rel="entry" type="application/atom+xml"
href="http://server:7070/rest/user/zaiwen300/entry" title="entry" />
    <id>http://server:7070/rest/user/zaiwen300/entry</id>
    <published>2012-03-29T13:36:17Z</published>
  </entry>
  <entry>
    <title>zaiwen200</title>
    <link rel="entry" type="application/atom+xml"
href="http://server:7070/rest/user/zaiwen200/entry" title="entry" />
    <id>http://server:7070/rest/user/zaiwen200/entry</id>
    <published>2012-03-29T13:36:17Z</published>
  </entry>
</feed>
```

If you want to return only one value in your XPath query, for example, to get a link to the next page, you can use XPath::findvalue() method which returns the one element.

Example

The following code snippet shows how to get the next page URL from the ATOM feed.

```
my $xp = XML::XPath->new($response->content);
my $next_page_url =
$xp->findvalue('/feed/link[@rel=\'next-page\']/@href')->value();
```

In case your XPath returns list of values, use XPath::findnodes() method.

Example

This code snippet shows how to get all titles from the entries in the ATOM feed.

```
my $xp = XML::XPath->new($response->content);
my @usernames = $xp->findnodes('/feed/entry/title');
foreach my $username (@usernames) {
   print $username->string_value;
}
```

Chapter 5: WS-MAN Web Services

This section provides information about the WS-MAN web services that are available in CA SOI.

Note: WS-MAN web services should be considered an obsolete API. Instead, we recommend using the REST (see page 13) API.

This section contains the following topics:

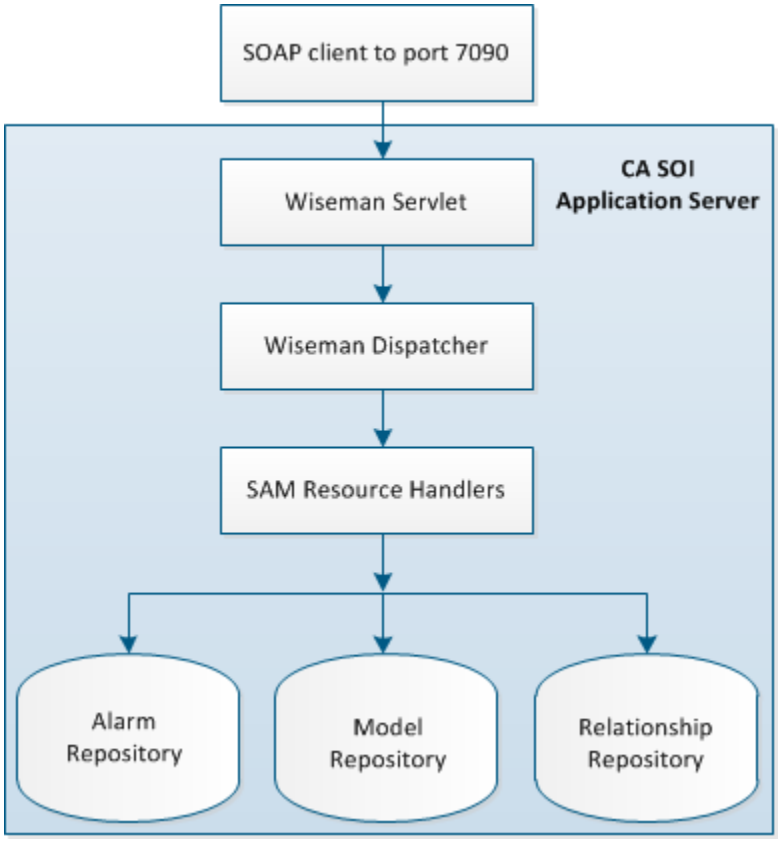
Introduction to WS-MAN Web Services

This section introduces the architecture of the CA SOI WS-MAN web services and the mechanisms used to interact with CA SOI data.

Web Services Architecture

CA SOI provides web services that comply with the WS-Management standard using a Wiseman implementation. The web services interface with the SA Manager to provide access to service, CI, alert, and other data. The web services use basic authentication to let clients access these resources through SOAP. Several Resource URIs are available through which the web services access the requested data.

The following diagram illustrates how the web service obtains data when it receives a request:



The WSManServlet intercepts client SOAP requests directed at the endpoint URL and dispatches them to the appropriate Resource Handler instance identified by the ResourceURI contained in the EPR element of the SOAP message. Wiseman automatically identifies the appropriate Resource Handler class in the following format: *Resource*_Handler.java.

Resources and Operations

The Endpoint Reference transport address for all CA SOI WS-Management web services is as follows:

`http://samanager:port/sam/webservice`

samanager

Defines the name of the server that contains the SA Manager.

port

Defines the SA Manager Tomcat port, which is 7090 by default.

WS-Management defines a resource addressing model based on the WS-Addressing standard. It uses the ReferenceParameter field in the WS-Addressing EndpointReference element to contain the following specific elements that identify the resource to act upon:

ResourceID

Defines the resource type or class.

SelectorSet

Defines the specific resource instance.

USM Schema Based Resources

The USM schema currently used as the CA SOI and CA Catalyst internal schema exposes CA SOI resources for web service operations using the following ResourceURI:

`http://ns.ca.com/2009/07/usm-core/`*`resource-class`*

resource-class

Defines the type of USM object, which can be Entity, BinaryRelationship, Alert, or Notification.

The following WSDL file outlines the available operations:

`http://`*`samanager`*`:`*`port`*`/sam/webservice/wsdls/usm-core-200907.wsdl`

Access the USM schema as follows:

`http://`*`samanager`*`:`*`port`*`/sam/webservice/schemas/usm-core-200907.xsd`

The initial USM schema web services implementation exposes the following USM resource types:

Entity

Refers to any CI in CA SOI, including services.

BinaryRelationship

Refers to the relationship between CIs. Note that this is different from the propagation type, which used the term relationship in previous releases.

Alert

Refers to CA SOI alerts.

Notification

Refers to the subscription to notifications for changes to Entities, Alerts, and BinaryRelationships.

Note: Notification resource is applicable only for CA SOI; it does not appear in CA Catalyst. By default, the USM schema does not contain any definition for the Notification resource. To ensure that it is available in the USM schema for CA SOI, special Java classes have been added explicitly for CA SOI.

Any of these resource types may require the following identifiers in the web service request as SelectorSets:

MdrProduct

Defines the connector data source. Each connector has a specific MdrProduct value formatted as a five-digit number prefixed by 'CA:'. For example, the MdrProduct value for resources created by web services is CA:09996. For a list of MdrProduct values, see the *Connector Guide*.

MdrProdInstance

Defines the host name associated with the resource.

MdrElementID

Defines a value that uniquely identifies the resource.

You can retrieve these values from existing resources from the Operations Console or through an Enumerate web services operation.

USM 01-2009 Based Resources

A robust set of web services is available that uses a previous version of the USM schema (referred in the document as USM 01-2009) from the following ResourceURI:

`http://ns.ca.com/2009/01/usm-data/`*`resource-class`*

resource-class

Defines the type of USM object, which can be Queue, Customer, RelationshipPolicy, EscalationPolicy, or EscalationPolicyAction.

The following WSDL file outlines the available operations:

`http://`*`samanager`*`:`*`port`*`/sam/webservice/wsdls/usm2.wsdl`

Access the USM 01-2009 schema as follows:

`http://`*`samanager`*`:`*`port`*`/sam/webservice/schemas/usm2.xsd`

This implementation supports resources that are not accessible from the current USM schema web services, such as propagation policy, escalation policy, and escalation actions. Use this implementation to retrieve resources not available from the current USM schema web services, or for backward compatibility with previous web service implementations.

Any of these resource types may require one of the following identifiers in the web service request as the SelectorSet:

ASBOID

Defines the resource using the following properties:

ASBOID.source

Defines the connector data source. Each data source has a unique value that you can obtain from the usm2.xsd file or an Enumerate operation.

ASBOID.id

Defines the unique ID value for the resource, which you can obtain from an Enumerate operation.

If you use ASBOID to uniquely identify the resource, include these properties separately.

USMID

Defines a value that uniquely identifies the resource by concatenating the ASBOID.source and ASBOID.id values.

WS-Transfer Operations

The web service resource handlers support all WS-Transfer operations as defined in the WS-Management specification:

- Get: http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
- Put: http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
- Create: http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
- Delete: http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete

The SelectorSet element is required to identify the appropriate resource instance. Each web service type (USM 01-2009 (see page 48), and USM (see page 47)) requires different SelectorSet values.

Fragment-Level WS-Transfer Operations

Fragment-level WS-Transfer is a WS-Management extension that lets you access a subset of resource properties. The web service resource handlers support fragment-level WS-Transfer for some operations where applicable using the default XPath dialect specified by the following URI:

`http://www.w3.org/TR/1999/REC-xpath-19991116`

The syntax is a list of property identifiers separated by "|" characters, such as the following:

```
<wsman:FragmentTransfer Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116"
env:mustUnderstand="true">AlertID|SvcDeskTicket|SDTicketProp|AnnotationList
</wsman:FragmentTransfer>
```

WS-Enumeration Operations

The web service resource handlers support the following WS-Enumeration operations as defined in the WS-Management specification:

- Enumerate: http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
- Pull: http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull
- Release: http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release

Wiseman 1.0 does not implement the following operations that may be supported in the future:

- Renew: http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew
- GetStatus: http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus
- EnumerationEnd: http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerationEnd

WS-Enumeration Filtering

WS-Enumeration defines XPath as the default filter dialect. XPath is designed to operate on XML elements, so all instances first must be retrieved, converted to XML, and then passed through the filter.

Configure an enumeration filter as follows:

```
final String xpathFilter = /alarm:alarm[alarm:ClassName='SA_Service']";
final Map<String, String> namespaces = new HashMap<String, String>(1);
namespaces.put("alarm","http://schemas.sam.ca.com/webservice/1/alarm.xsd");
```

This filter retrieves only those alerts that are associated with services.

WS-Eventing Operations

The web service resource handlers Entity, Alert, and BinaryRelationship support the following WS-Eventing operations as defined in the WS-Management specification (http://schemas.xmlsoap.org/ws/2004/08/eventing):

- Subscribe: http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
- Unsubscribe: http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe

This release does not support the following operations:

- GetStatus: http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus
- SubscriptionEnd: http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd
- Renew: http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew

Available Web Services

The available web services and their resource URLs are as follows:

- USM Entity (see page 52): http://ns.ca.com/2009/07/usm-core/Entity
- USM BinaryRelationship (see page 57): http://ns.ca.com/2009/07/usm-core/BinaryRelationship
- Alert (see page 85): http://ns.ca.com/2009/07/usm-core/Alert
- Propagation Policy (see page 92): http://ns.ca.com/2009/01/usm-data/RelationshipPolicy
- Escalation Policy (see page 29): http://ns.ca.com/2009/01/usm-data/EscalationPolicy
- Escalation Action (see page 28): http://ns.ca.com/2009/01/usm-data/EscalationpolicyAction
- Queue (see page 74): http://ns.ca.com/2009/01/usm-data/Queue
- Notification (see page 68): http://ns.ca.com/2009/07/usm-core/Notification
- Customer (see page 27): http://ns.ca.com/2009/01/usm-data/Customer

Available Web Clients

This section summarizes the available web clients for testing and using the web services.

SoapUI

SoapUI is an open source application that lets you inspect, invoke, develop, simulate, and test web services. Developers and testers who are responsible for providing or consuming WSDL-based web services primarily use soapUI. CA SOI web services have been tested using this tool.

Java Web Client using Axis2

You can use an Axis2 Java web client from which you can generate Axi2-based stub classes for the CA SOI web services using the wsdl2java utility and then using the classes to generate a SOAP-based request.

For information about using Axis2 clients, see the *Wiseman Client Developer's Guide*.

USM Entity Web Services

This section provides information about the operations performed in USM entity web services.

Entity Web Services Overview

Entity web services use the USM schema to perform operations on USM entities, which include CIs, services, and alerts. Use the following endpoint URI when invoking the entity web services resource:

`http://ns.ca.com/2009/07/usm-core/Entity`

The following WSDL file outlines the available operations:

`http://samanager:port/sam/webservice/wsdls/usm-core-200907.wsdl`

Access the USM schema as follows:

`http://samanager:port/sam/webservice/schemas/usm-core-200907.xsd`

Using the entity web services requires basic knowledge of the USM schema and its properties. In addition to the schema itself, HTML documentation is available. For information about how to access the USM schema documentation, see the *Connector Guide*.

Get an Entity

Use the Get request to retrieve a specific entity. The following selectors are required as part of the request to identify a unique instance of a CA SOI CI:

Note: This request also retrieves the KPI properties (Risk, Quality, and Health) of an entity.

MdrProduct

Defines the connector data source. Each connector has a specific MdrProduct value formatted as a five-digit number prefixed by 'CA:'. For example, the MdrProduct value for resources created by web services is CA:09996. For a list of MdrProduct values, see the *Connector Guide*.

MdrProdInstance

Defines the host name associated with the resource.

MdrElementID

Defines a value that uniquely identifies the resource.

To get an entity, use the following properties in the request:

Operation: Get

Endpoint: http://*samanager:port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Entity

Selector: MdrProduct

Selector: MdrProdInstance

Selector: MdrElementID

The EntityHandlerImpl.Get() method returns an entity of the type of USM resource that represents the CA SOI CI in USM terms, including the CI type and its properties.

Get a List of Entities

To retrieve a list of active CIs, the web services use a combination of WS-Management Enumeration and Pull operations. You can filter the returned list using the WS-Management Filter element to pass a valid XPath expression to limit the number and type of CIs returned.

Use the className selector to filter the collection by USM type. For example, if you pass in a selector of “className=Service”, the collection is limited to entities of the Service type.

To get a list of entities, use the following properties in the request:

Operation: Enumerate & Pull

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Entity

Selector: className

The EntityIteratorImpl() method creates a collection of all the active CIs in CA SOI of the USM type defined in the className selector, and the Pull operation retrieves CIs in batches as defined by the MaxElements tag.

Create an Entity

Use the Create operation to create a CI in CA SOI. You define the CI type and USM property values for the new CI in the body of the request.

Note: For information about the required properties to include for an entity, see the USM schema documentation.

To create a CI, use the following properties in the request:

Operation: Create

Endpoint: http://*samanager*:*port/*sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Entity

Selector: null

The EntityHandlerImpl.Create() method extracts all defined property values from the body of the SOAP message and passes a map of the property names and values to the Connector Manager, which is the same interface that all connectors use to create CIs.

Note: Refer to the USM schema for a list of USM properties and their appropriate values. For information about how to access the USM schema documentation, see the *Connector Guide*.

Update an Entity

Use the Put operation to update the writable properties of a CI. Perform the update by passing in all of the USM properties and their new values in the body of the request.

To update an entity, use the following properties in the request:

Operation: Put

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Entity

Selector: null

The EntityHandlerImpl.Put method extracts the entity property name value pairs and passes them into a CI Update request to the Connector Manager.

Note: Refer to the USM schema for a list of USM properties and their appropriate values. For information about how to access the USM schema documentation, see the *Connector Guide*.

Delete an Entity

Use the Delete operation to delete a CI. If the CI exists and has active alerts, the operation clears the associated alerts before deleting the CI. Pass the USM properties of the CI to delete in the body of the request.

Note: For information about the required properties to include for an entity, see the USM schema documentation.

To delete an entity, use the following properties in the request:

Operation: Delete

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Entity

Selector: Null

The EntityHandlerImpl.Delete() method verifies that the CI exists, clears any active associated alerts, and deletes the CI. The CI is deleted only if the connector is a trusted source, otherwise the CI is marked as 'Unmanaged.' A trusted source connector is the trusted source of record for a CI. Imported information uses the source connector as a trusted source unless you change the trusted source in the Operations Console.

Entity Web Services Examples

The following example shows the SOAP messages of many of the available entity web services.

Example: Update a ComputerSystem CI

The following example SOAP message is a Put request to update a set of properties in a specific ComputerSystem CI:

```
<env:Envelope>
<env:Header>
    <wsa:ReplyTo xmlns:usm="http://ns.ca.com/2009/07/usm-core">
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID xmlns:usm="http://ns.ca.com/2009/07/usm-core"
    env:mustUnderstand="true">
        uuid:4add8974-25a0-4fc4-84f6-c7f88158f8f7
    </wsa:MessageID>
    <wsa:To xmlns:usm="http://ns.ca.com/2009/07/usm-core"
    env:mustUnderstand="true">
        http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI xmlns:usm="http://ns.ca.com/2009/07/usm-core">
        http://ns.ca.com/2009/07/usm-core/Entity
    </wsman:ResourceURI>
    <wsman:OperationTimeout xmlns:usm="http://ns.ca.com/2009/07/usm-core">
        PT30.000S
    </wsman:OperationTimeout>
    <wsa:Action xmlns:usm="http://ns.ca.com/2009/07/usm-core"
    env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
    </wsa:Action>
</env:Header>
<env:Body>
    <usm:ComputerSystem >
         <ssa_connector_name>server1.ca.com</ssa_connector_name>
         <ssa_silo_name>WebServiceForSSA_server1@server1.ca.com</ssa_silo_name>
         <usm:MdrProduct>CA:09996</usm:MdrProduct>
         <usm:MdrProdInstance>server1.ca.com</usm:MdrProdInstance>
         <usm:MdrElementID>ComputerSystem:server1</usm:MdrElementID>
         <usm:InstanceName>CA:server1</usm:InstanceName>
         <usm:Label>USM-WSServer3</usm:Label>
         <usm:Description>New description</usm:Description>
         <usm:AdministrativeStatus>Managed</usm:AdministrativeStatus>
         <usm:Vendor>Dell</usm:Vendor>
```

```
            <usm:PrimaryDnsName>server1.ca.com</usm:PrimaryDnsName>
            <usm:PrimaryIPV4Address>111.11.11.11</usm:PrimaryIPV4Address>
            <usm:ComputerName>server1</usm:ComputerName>
            <usm:MemoryInGB>512</usm:MemoryInGB>
        </usm:ComputerSystem>
</env:Body>
</env:Envelope>
```

The web service updates the properties listed in bold in the body of the request. The SOAP response to this request is as follows:

```
<env:Envelope>
<env:Header>
    <wsa:Action env:mustUnderstand="true"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
        http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse
    </wsa:Action>
    <wsa:MessageID env:mustUnderstand="true"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
        uuid:f1ccc796-28a6-4afc-9766-d1438fe7d011
    </wsa:MessageID>
    <wsa:RelatesTo xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
        uuid:4add8974-25a0-4fc4-84f6-c7f88158f8f7
    </wsa:RelatesTo>
    <wsa:To env:mustUnderstand="true"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
</env:Header>
<env:Body/>
</env:Envelope>
```

Subscribe to Notifications for Entity Changes

To subscribe to notifications for changes to Entity, use the Notification web services. The Entity resource supports the WS-Eventing functionality that enables a web client to subscribe to notification events when an entity is created, deleted, or updated.

Note: For more information, see Notification Web Services.

USM Binary Relationship Web Services

This section provides information about the operations performed in USM Binary Relationship web services.

BinaryRelationship Web Services Overview

BinaryRelationship web services use the USM schema to perform operations on USM BinaryRelationships, which define the relationships between CIs in CA SOI.

USM BinaryRelationships (referred to as relationships) are not the same as the entities known as relationships in previous releases of CA SOI. Propagation types define how the impact is propagated between related CIs (which was the previous role of relationships), and relationships are USM entities that show how CIs are linked. Every relationship in CA SOI has a corresponding propagation type. The USM Semantic property indicates the propagation type for each relationship. Therefore, when you interact with relationships, you can view, create, or edit their corresponding propagation types at the same time.

Note: For a list of the available relationships and propagation types, see the *Service Modeling Best Practices Guide*.

Use the following endpoint URI when invoking the relationship web services resource:

`http://ns.ca.com/2009/07/usm-core/BinaryRelationship`

The following WSDL file outlines the available operations:

`http://samanager:port/sam/webservice/wsdls/usm-core-200907.wsdl`

Access the USM schema as follows:

`http://samanager:port/sam/webservice/schemas/usm-core-200907.xsd`

Using the relationship web services requires basic knowledge of the USM schema and its properties. In addition to the schema itself, HTML documentation is available. For information about how to access the USM schema documentation, see the *Connector Guide*.

Get a Relationship

Use the Get request to get a specific relationship. Each relationship belongs to one of the USM BinaryRelationship types. The following selectors are required as part of the USM definition to identify a unique instance of a relationship:

MdrProduct

Defines the connector data source. Each connector has a specific MdrProduct value formatted as a five-digit number prefixed by 'CA:'. For example, the MdrProduct value for resources created by web services is CA:09996. For a list of MdrProduct values, see the *Connector Guide*.

MdrProdInstance

Defines the host name associated with the resource.

MdrElementID

Defines a value that uniquely identifies the resource.

To get a relationship, use the following properties in the request:

Operation: Get

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/BinaryRelationship

Selector: MdrProduct

Selector: MdrProdInstance

Selector: MdrElementID

The BinaryRelationshipImpl.Get() method returns an entity of the type of USM BinaryRelationship that represents the CA SOI relationship in USM terms.

Get a List of Relationships

To retrieve a list of relationships for a given CA SOI service, the web services use a combination of WS-Management Enumeration and Pull operations. You can filter the returned list using the WS-Management Filter element to pass a valid XPath Expression to limit the number and type of relationships returned.

Use the serviceName selector to filter the collection by service.

To get a list of relationships, use the following properties in the request:

Operation: Enumerate & Pull

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/BinaryRelationship

Selector: serviceName

The BinaryRelationshipIteratorImpl() method creates a collection of all the relationships for the service defined in the selector. The Pull operation retrieves relationships in batches as defined by the MaxElements tag.

Create a Relationship

Use the Create operation to create a relationship between two CIs in CA SOI. You define the type and USM property values for the new relationship in the body of the request. You should include the following information:

- Unique identifiers (MdrProduct, MdrProdInstance, MdrElementID)
- Unique identifiers for the source CI
- Unique identifiers for the target CI
- Unique identifiers for the service in which to include the relationship
- Semantic value to define the propagation type

Note: For information about the required properties to include for a relationship, see the USM schema documentation. For more information about property names and formatting, see USM Binary Relationship Web Services Examples (see page 62).

To create a relationship, use the following properties in the request:

Operation: Create

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/BinaryRelationship

Selector: null

The BinaryRelationshipHandlerImpl.Create() method extracts all defined properties from the body of the SOAP message and passes a map of the property names and values to the Connector Manager, which is the same interface that all connectors use to create relationships.

Note: Refer to the USM schema for a list of USM properties and their appropriate values. For information about how to access the USM schema documentation, see the *Connector Guide*. For a list of all available BinaryRelationship types, see the *Service Modeling Best Practices Guide*.

Update a Relationship

Use the Put operation to update the writable properties of a relationship. Perform the update by passing in all of the USM BinaryRelationship properties and their new values in the body of the request. You should include the following information:

- Unique identifiers (MdrProduct, MdrProdInstance, MdrElementID)
- Unique identifiers for the source CI
- Unique identifiers for the target CI
- Unique identifiers for the service in which to include the relationship
- Semantic value to define the propagation type

Note: For more information, see USM Binary Relationship Web Services Examples (see page 62).

To update a relationship, use the following properties in the request:

Operation: Put

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/BinaryRelationship

Selector: null

The BinaryRelationshipHandlerImpl.Put method extracts the relationship property name value pairs and passes them into a Relationship Update request to the Connector Manager.

Note: Refer to the USM schema for a list of USM properties and their appropriate values. For information about how to access the USM schema documentation, see the *Connector Guide*. For a list of all available BinaryRelationship types, see the *Service Modeling Best Practices Guide*.

Delete a Relationship

Use the Delete operation to delete a relationship. Pass the USM properties of the relationship to delete in the body of the request.

Note: For information about the required properties to include for a relationship, see the USM schema documentation.

To delete a relationship, use the following properties in the request:

Operation: Delete

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/BinaryRelationship

Selector: Null

The BinaryRelationshipHandlerImpl.Delete() method passes the delete request to the Connector Manager to process.

Relationship Web Services Examples

The following examples show the SOAP messages of many of the available relationship web services.

Example: Get a list of relationships for a specific service

The following example SOAP messages are Enumerate and Pull requests to retrieve a list of relationships associated with the servicetest service:

```
<env:Envelope>
<env:Header>
    <wsa:Action xmlns:ns13="http://ns.ca.com/2009/07/usm-core"
    env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
    </wsa:Action>
    <wsa:ReplyTo xmlns:ns13="http://ns.ca.com/2009/07/usm-core">
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID xmlns:ns13="http://ns.ca.com/2009/07/usm-core"
    env:mustUnderstand="true">
        uuid:c189ea6b-b07f-48c4-8060-819c1dd2da12
    </wsa:MessageID>
```

```
<wsa:To xmlns:ns13="http://ns.ca.com/2009/07/usm-core"
env:mustUnderstand="true">
    http://localhost:7090/sam/webservice
</wsa:To>
<wsman:ResourceURI xmlns:ns13="http://ns.ca.com/2009/07/usm-core">
    http://ns.ca.com/2009/07/usm-core/BinaryRelationship
</wsman:ResourceURI>
<wsman:SelectorSet xmlns:ns13="http://ns.ca.com/2009/07/usm-core">
<wsman:Selector Name="serviceName">SA_Service:servicetest
</wsman:Selector>
</wsman:SelectorSet>
<wsman:RequestTotalItemsCountEstimate
xmlns:ns13="http://ns.ca.com/2009/07/usm-core"/>
</env:Header>
<env:Body>
    <wsen:Enumerate xmlns:ns13="http://ns.ca.com/2009/07/usm-core">
    <wsman:EnumerationMode>EnumerateObjectAndEPR</wsman:EnumerationMode>
    </wsen:Enumerate>
</env:Body>
</env:Envelope>
<env:Envelope>
<env:Header>
<wsa:Action xmlns:ns13="http://ns.ca.com/2009/07/usm-core"
env:mustUnderstand="true">
    http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull
</wsa:Action>
<wsa:ReplyTo xmlns:ns13="http://ns.ca.com/2009/07/usm-core">
<wsa:Address env:mustUnderstand="true">
    http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
</wsa:Address>
</wsa:ReplyTo>
<wsa:MessageID xmlns:ns13="http://ns.ca.com/2009/07/usm-core"
env:mustUnderstand="true">
    uuid:630fa96e-7f01-46bd-b3aa-7f79dc57445c
</wsa:MessageID>
<wsa:To xmlns:ns13="http://ns.ca.com/2009/07/usm-core"
env:mustUnderstand="true">
    http://localhost:7090/sam/webservice
</wsa:To>
<wsman:ResourceURI xmlns:ns13="http://ns.ca.com/2009/07/usm-core">
    http://ns.ca.com/2009/07/usm-core/BinaryRelationship
</wsman:ResourceURI>
<wsman:OperationTimeout xmlns:ns13="http://ns.ca.com/2009/07/usm-core">
    P0Y0M0DT0H0M30.000S
</wsman:OperationTimeout>
</env:Header>
<env:Body>
<wsen:Pull xmlns:ns13="http://ns.ca.com/2009/07/usm-core">
```

```
      <wsen:EnumerationContext>7a26ba28-7c8e-46dc-b456-f70c67d2f2b6
      </wsen:EnumerationContext>
      <wsen:MaxTime>P0Y0M0DT0H0M30.000S</wsen:MaxTime>
      <wsen:MaxElements>20</wsen:MaxElements>
    </wsen:Pull>
</env:Body>
</env:Envelope>
```

The bold SelectorSet syntax defines the service for which to list relationships. The SOAP response to this request is as follows:

```
<env:Envelope>
<env:Header>
    <wsa:Action env:mustUnderstand="true"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
        http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse
    </wsa:Action>
    <wsa:MessageID env:mustUnderstand="true">
        uuid:6bbbc6ad-3f91-4191-ac58-b17dce91d581
    </wsa:MessageID>
    <wsa:RelatesTo >uuid:630fa96e-7f01-46bd-b3aa-7f79dc57445c</wsa:RelatesTo>
    <wsa:To env:mustUnderstand="true" >
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
    </env:Header>
    <env:Body>
    <wsen:PullResponse >
    <wsen:Items>
    <wsman:Item>
    <BinaryRelationship>
       <MdrProduct>CA:00047</MdrProduct>
              <MdrProdInstance>server1</MdrProdInstance>
              <MdrElementID>21:21:22</MdrElementID>
              <ssa_usm_rel_id>IsAffectedBy</ssa_usm_rel_id>
              <ssa_impact>0</ssa_impact>
              <ssa_policy_id>0</ssa_policy_id>
              <ssa_relevance>0</ssa_relevance>
              <ssa_root_cause>false</ssa_root_cause>
              <SourceMdrProduct>CA:00047</SourceMdrProduct>
              <SourceMdrProdInstance>server1</SourceMdrProdInstance>
              <SourceMdrElementID>21</SourceMdrElementID>
              <ns12:TargetMdrProduct>CA:00047</TargetMdrProduct>
              <ns12:TargetMdrProdInstance>server2</TargetMdrProdInstance>
              <ns12:TargetMdrElementID>22</TargetMdrElementID>
              <ns12:Semantic>Depends On</Semantic>
              <ns12:ScopeMdrProduct>CA:00047</ScopeMdrProduct>
              <ns12:ScopeMdrProdInstance>server1</ScopeMdrProdInstance>
```

```
   <ns12:ScopeMdrElementID>21</ScopeMdrElementID>
   <ns12:Significance>5</Significance>
</BinaryRelationship>
<wsa:EndpointReference>
<wsa:Address env:mustUnderstand="true">
     http://localhost:7090/sam/webservice</wsa:Address>
<wsa:ReferenceParameters>
<wsman:ResourceURI>
     http://ns.ca.com/2009/07/usm-core/BinaryRelationship
</wsman:ResourceURI>
<wsman:SelectorSet>
<wsman:Selector Name="MdrProduct">CA:00047</wsman:Selector>
<wsman:Selector Name="MdrElementID">21:21:22</wsman:Selector>
<wsman:Selector Name="MdrProdInstance">server1</wsman:Selector>
</wsman:SelectorSet>
</wsa:ReferenceParameters>
</wsa:EndpointReference>
</wsman:Item>
<wsman:Item>
<ns12:BinaryRelationship>
  <ns12:MdrProduct>CA:00047</ns12:MdrProduct>
  <ns12:MdrProdInstance>server3</ns12:MdrProdInstance>
  <ns12:MdrElementID>21:21:9</ns12:MdrElementID>
  <ssa_usm_rel_id>IsAffectedBy</ssa_usm_rel_id>
  <ssa_impact>0</ssa_impact>
  <ssa_policy_id>0</ssa_policy_id>
  <ssa_relevance>0</ssa_relevance>
  <ssa_root_cause>false</ssa_root_cause>
  <ns12:SourceMdrProduct>CA:00047</ns12:SourceMdrProduct>
  <ns12:SourceMdrProdInstance>server3</ns12:SourceMdrProdInstance>
  <ns12:SourceMdrElementID>21</ns12:SourceMdrElementID>
  <ns12:TargetMdrProduct>CA:00047</ns12:TargetMdrProduct>
  <ns12:TargetMdrProdInstance>server4</ns12:TargetMdrProdInstance>
  <ns12:TargetMdrElementID>9</ns12:TargetMdrElementID>
  <ns12:Semantic>Depends On</ns12:Semantic>
  <ns12:ScopeMdrProduct>CA:00047</ns12:ScopeMdrProduct>
  <ns12:ScopeMdrProdInstance>server3</ns12:ScopeMdrProdInstance>
  <ns12:ScopeMdrElementID>21</ns12:ScopeMdrElementID>
  <ns12:Significance>5</ns12:Significance>
</BinaryRelationship>
<wsa:EndpointReference>
<wsa:Address
env:mustUnderstand="true">http://localhost:7090/sam/webservice
</wsa:Address>
```

```
            <wsa:ReferenceParameters>
            <wsman:ResourceURI>http://ns.ca.com/2009/07/usm-core/BinaryRelationshi
    p
            </wsman:ResourceURI>
            <wsman:SelectorSet>
            <wsman:Selector Name="MdrProduct">CA:00047</wsman:Selector>
            <wsman:Selector Name="MdrElementID">21:21:9</wsman:Selector>
            <wsman:Selector Name="MdrProdInstance">symbe01-5</wsman:Selector>
            </wsman:SelectorSet>
            </wsa:ReferenceParameters>
            </wsa:EndpointReference>
            </wsman:Item>
            </wsen:Items>
            <wsen:EndOfSequence xsi:type="xs:string"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
    </wsen:PullResponse>
</env:Body>
</env:Envelope>
```

The bold syntax shows the details of the returned relationships for the servicetest service. Note the returned properties, including the relationship type (ssa_usm_rel_id), propagation type (Semantic), and source and target CIs.

Example: Create a relationship

The following example SOAP message is a Create request to create a new relationship with Aggregates propagation:

```
<env:Envelope>
<env:Header>
    <wsa:ReplyTo xmlns:usm="http://ns.ca.com/2009/07/usm-core">
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID xmlns:usm="http://ns.ca.com/2009/07/usm-core"
    env:mustUnderstand="true">
        uuid:4add8974-25a0-4fc4-84f6-c7f88158f8f7
    </wsa:MessageID>
    <wsa:To xmlns:usm="http://ns.ca.com/2009/07/usm-core"
    env:mustUnderstand="true">
        http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI xmlns:usm="http://ns.ca.com/2009/07/usm-core">
        http://ns.ca.com/2009/07/usm-core/BinaryRelationship
    </wsman:ResourceURI>
```

```
    <wsman:OperationTimeout xmlns:usm="http://ns.ca.com/2009/07/usm-core">
        PT30.000S
    </wsman:OperationTimeout>
    <wsa:Action xmlns:usm="http://ns.ca.com/2009/07/usm-core"
    env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
    </wsa:Action>
    </env:Header>
    <env:Body>
    <BinaryRelationship xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
       <MdrProduct>CA:09997</MdrProduct>
       <MdrProdInstance>server1.ca.com</MdrProdInstance>
       <MdrElementID>Agg_Service:Agg_Service:Dep_Server</MdrElementID>
       <SourceMdrProduct>CA:09997</SourceMdrProduct>
       <SourceMdrProdInstance>server1.ca.com</SourceMdrProdInstance>
       <SourceMdrElementID>SA_Service:Agg_Service</SourceMdrElementID>
       <TargetMdrProduct>CA:09997</TargetMdrProduct>
       <TargetMdrProdInstance>server2.ca.com</TargetMdrProdInstance>
       <TargetMdrElementID>SA_Server:Dep_Server</TargetMdrElementID>
       <Semantic>Aggregates</Semantic>
       <ScopeMdrProduct>CA:09997</ScopeMdrProduct>
       <ScopeMdrProdInstance>server1.ca.com</ScopeMdrProdInstance>
       <ScopeMdrElementID>SA_Service:Agg_Service</ScopeMdrElementID>
       <Significance>5</Significance>
       <ssa_connector_name>server1.ca.com</ssa_connector_name>
       <ssa_silo_name>WebServiceForSSA_server1.ca.com@server1.ca.com</ssa_silo_na
    me>
    </BinaryRelationship>
</env:Body>
</env:Envelope>
```

The web service creates the relationship using the properties listed in bold in the body of the request. Note that the Semantic property defines the propagation type for the relationship as Aggregates. No relationship type is defined, so the relationship automatically obtains the default type that maps to Aggregates propagation. The SOAP response to this request is as follows:

```
<env:Envelope>
<env:Header>
    <wsa:Action env:mustUnderstand="true"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
        http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
    </wsa:Action>
    <wsa:MessageID env:mustUnderstand="true"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
        uuid:d23a181b-5e2d-4dae-85a0-8ee4415c227b
    </wsa:MessageID>
```

```
<wsa:RelatesTo xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    uuid:4add8974-25a0-4fc4-84f6-c7f88158f8f7
</wsa:RelatesTo>
<wsa:To env:mustUnderstand="true"
xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
</wsa:To>
</env:Header>
<env:Body>
<wxf:ResourceCreated xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
<wsa:Address env:mustUnderstand="true">
http://localhost:7090/sam/webservice/</wsa:Address>
<wsa:ReferenceParameters>
<wsman:ResourceURI>http://ns.ca.com/2009/07/usm-core/BinaryRelationship
</wsman:ResourceURI>
</wsa:ReferenceParameters>
</wxf:ResourceCreated>
</env:Body>
</env:Envelope>
```

Subscribe to Notifications for Relationship Changes

To subscribe to notifications for changes to Relationship, use the Notification web services. The Relationship resource supports the WS-Eventing functionality that enables a web client to subscribe to notification events when a relationship is created, deleted, or updated.

Note: For more information, see Notification Web Services.

Notification Web Services

This section provides information about the operations performed in Notification web services.

Notification Web Services Overview

Notification web services are used to subscribe to notifications for changes to Entities, Alerts, and Relationships.

Use the following endpoint URI when invoking the Notification web services resource:

```
http://ns.ca.com/2009/07/usm-core/Notification
```

The following WSDL file outlines the available operations:

`http://samanager:port/sam/webservice/wsdls/usm-core-200907.wsdl`

Access the USM schema as follows:

`http://samanager:port/sam/webservice/schemas/usm-core-200907.xsd`

Using the Notification web services requires basic knowledge of the USM schema and its properties. In addition to the schema itself, HTML documentation is available. For information about how to access the USM schema documentation, see the *Connector Guide*.

How to Subscribe to Notifications for Entity, Relationship, and Alert Changes

Addition of the WS-Eventing functionality to the Entity (see page 52), Relationship (see page 58), and Alert (see page 86) resources allows a web client to subscribe to notification events when an entity (CI), relationship, or alert is created, deleted, or updated. The web client sends a subscription request with a filter detailing what notification events the client wants to receive and the mode of delivery. If the request is successful, the web service sends a response with a subscription ID. The subscription provides two delivery modes: Pull and Push. In case of Pull, a client periodically polls for notification events, and in case of Push, the notification events are published to an *event sink*, where the web client can process them.

You can perform the following steps to complete the task:

1. Send a Subscribe request for the Notification resource.
2. Add a filter with the following events as appropriate:
   - entityCreated, entityModified, and entityDeleted for the Entity resource
   - binaryrelationshipCreated, binaryrelationshipModified, and binaryrelationshipDeleted for the Relationship resource
   - alertCreated, alertModified, and alertCleared for the Alert resource
3. Update authentication details (user ID and password).
4. Run the request, and review the response for the subscription ID.

Note: For Pull mode, send a Pull request for the Notification resource.

Create a Subscription

Use the Subscribe operation to create a subscription request for notification. This request is sent to the event source.

To create a subscription, use the following properties in the request:

Operation: Subscribe

Endpoint: http://samanager:port/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Notification

Creating a subscription request does not require any selector; it requires a filter. Therefore, the web client must provide the following information:

- Address of the web service
- Resource URL
- Action (Subscribe)
- Reply-to Address (default)
- MessageID (uuid)
- Delivery Mode (Pull in this case)
- Heartbeats (Timeout value for subscription)
- Filter (what events to subscribe for)
- Bookmark (from what point to start sending events)

Note: For more information about how to create this request, see the first example in the Notification Web Services Examples (see page 72) section.

Delete a Subscription

Use the Unsubscribe operation to explicitly delete a subscription when you do not want notifications associated with the subscription.

To delete a subscription, use the following properties in the request:

Operation: Unsubscribe

Endpoint: http://samanager:port/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Notification

Deleting a subscription does not require any selector; it requires a filter. Therefore, the web client must provide the following information:

- Address of the web service
- Resource URL
- Action (Unsubscribe)
- Reply-to Address (default)
- MessageID (uuid)
- Identifier (Subscription ID)

Note: For more information about how to create this request, see the second example in the Notification Web Services Examples (see page 72) section.

Pull a Subscription Notification

Use the WS-Management Pull operation in combination with the delivery mode *Pull* to retrieve periodically any notification events such as entityModified, entityCreated, and entityDeleted for Entity; binaryrelationshipModified, binaryrelationshipCreated, and binaryrelationshipDeleted for Relationship; and alertModified, alertCreated, and alertDeleted for Alert.

To pull a subscription notification, use the following properties in the request:

Operation: Pull

Endpoint: http://samanager:port/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Notification

EnumerationContext: subscriptionID

Note: For more information about how to create this request, see the third example in the Notification Web Services Examples (see page 72) section.

Notification Web Services Examples

The following examples show the SOAP requests to create a subscription, retrieve notification events, and delete a subscription for Entity.

Example 1: Create a subscription

The following example SOAP request shows how you can create a subscription request:

```
<env:Envelope>
<env:Header>
    <wsa:To env:mustUnderstand="true">
        http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI>
        http://ns.ca.com/2009/07/usm-core/Notification
    </wsman:ResourceURI>
    <wsa:Action  env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
    </wsa:Action>
    <wsa:ReplyTo>
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID env:mustUnderstand="true">
        uuid:afb3d3ee-cdb2-4587-9087-d09d77ab5d8d
    </wsa:MessageID>
</env:Header>
    <env:Body>
    <wse:Subscribe>
        <wse:Delivery Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull">
        <wsman:Heartbeats>PT5M0.000S</wsman:Heartbeats>
    </wse:Delivery>
    <wse:Filter Dialect="http://ns.ca.com/2009/07/usm-core/NotificationFilter">
        entityModified;entityCreated;entityDeleted
    </wse:Filter>
    <wsman:Bookmark>
    <ns15:Bookmark>
        http://schemas.dmtf.org/wbem/wsman/1/wsman/bookmark/earliest
    </ns15:Bookmark>
    </wsman:Bookmark>
    </wse:Subscribe>
    </env:Body>
</env:Envelope>
```

Example 2: Delete a Subscription

The following example SOAP request shows how to delete a subscription:

```
<env:Envelope >
<env:Header>
    <wsa:To env:mustUnderstand="true">
        http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI >
        http://ns.ca.com/2009/07/usm-core/Notification
    </wsman:ResourceURI>
    <wsa:Action env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe
    </wsa:Action>
    <wsa:ReplyTo>
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID env:mustUnderstand="true">
        uuid:0f589af8-1b38-4948-a171-f4dc419f49db
    </wsa:MessageID>
    <wse:Identifier >
        5264d7de-c6ac-46f4-80ea-fd59c11a7561
    </wse:Identifier>
</env:Header>
    <env:Body>
        <wse:Unsubscribe/>
    </env:Body>
</env:Envelope>
```

Example 3: Retrieve Notification Events

The following example SOAP request shows how to retrieve notification events:

```
<env:Envelope>
<env:Header>
    <wsa:To env:mustUnderstand="true">
        http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI >
        http://ns.ca.com/2009/07/usm-core/Notification
    </wsman:ResourceURI>
```

```
    <wsman:OperationTimeout>
        PT0.100S
    </wsman:OperationTimeout>
    <wsa:Action env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull
    </wsa:Action>
    <wsa:ReplyTo >
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID env:mustUnderstand="true">
        uuid:7f71fa61-1e94-4265-97e4-f4bf0ddd7e68
    </wsa:MessageID>
</env:Header>
    <env:Body>
        <wsen:Pull >
        <wsen:EnumerationContext>
                5264d7de-c6ac-46f4-80ea-fd59c11a7561
        </wsen:EnumerationContext>
        <wsen:MaxTime>
                PT0.100S
        </wsen:MaxTime>
        <wsen:MaxElements>
                10
        </wsen:MaxElements>
        </wsen:Pull>
    </env:Body>
</env:Envelope>
```

Queue Web Services

This section provides information about the operations performed in queue web services.

Note: WS-MAN web services should be considered obsolete. We recommend using the Alert Queue (see page 24) REST web services instead.

Queue Web Services Overview

Queue web services use the USM 01-2009 schema to perform operations on alert management queues in CA SOI. Alert queues collect and logically organize closely related groups of alerts in CA SOI.

Use the following endpoint URI when invoking the queue web services resource:

`http://ns.ca.com/2009/01/usm-data/Queue`

The following WSDL file outlines the available operations:

`http://samanager:port/sam/webservice/wsdls/usm2.wsdl`

Access the USM 01-2009 schema as follows:

`http://samanager:port/sam/webservice/schemas/usm2.xsd`

Note: For more information about queues, see the *Event and Alert Management Best Practices Guide*.

Get a Queue

Use the Get request to retrieve a specific queue. The following selectors are required as part of the request to identify a unique instance of a queue:

ASBOID.id

Uniquely identifies the queue using the Action ID value. Derive this value using an Enumerate operation.

ASBOID.source

Defines the DomainID of the CA SOI model repository. This value is constant for the SA Manager. Derive the value using an Enumerate operation.

To get a queue, use the following properties in the request:

Operation: Get

Endpoint: http://*samanager:port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/Queue

Selector: ASBOID.id

Selector: ASBOID.source

The QueueHandlerImpl.Get() method returns the queue based on the selectors.

Get a List of Queues

To retrieve a list of queues, the web services use a combination of WS-Management Enumeration and Pull operations. You can filter the returned list using the WS-Management Filter element to pass a valid XPath expression to limit the number and type of queues returned.

To get a list of queues, use the following properties in the request:

Operation: Enumerate & Pull

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/Queue

Selector: Null

The QueueIteratorImpl() method creates a collection of all alert queues in CA SOI. The Pull operation retrieves queues in batches as defined by the MaxElements tag.

Create a Queue

Use the Create operation to create an alert queue in CA SOI. You define the queue type and property values for the new queue in the body of the request.

Note: For information about the required properties to include for a queue, see the USM 01-2009 schema.

To create a queue, use the following properties in the request:

Operation: Create

Endpoint: http://*samanager*:*port/*sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/Queue

Selector: Null

The QueueHandlerImpl.Create() method extracts all defined property values from the body of the SOAP message and creates the queue in CA SOI.

Update a Queue

Use the Put operation to update the writeable properties of a queue. Perform the update by passing in all of the properties and their new values in the body of the request.

To update a queue, use the following properties in the request:

Operation: Put

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/Queue

Selector: ASBOID.id

Selector: ASBOID.source

Fragment: property (Description indicating which property to update)

The QueueHandlerImpl.Put() method inspects the SOAP Header and determines if it is a fragment-based update. If so, only the attributes specified in the Fragment are updated; otherwise, all of the writable attributes are updated.

Note: For information about the required properties to include for a queue, see the USM 01-2009 schema.

Delete a Queue

Use the Delete operation to delete a queue. Use the queue ASBOID.id and ASBOID.source values as selectors.

To delete a queue, use the following properties in the request:

Operation: Delete

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/Queue

Selector: ASBOID.id

Selector: ASBOID.source

The QueueHandlerImpl.Delete() method verifies that the queue exists and deletes it.

Queue Web Services Examples

The following examples show the SOAP messages of many of the available Queue web services.

Example 1: Get a Queue

The following example SOAP message is a Get request to retrieve a queue:

```
<env:Envelope>
<env:Header>
    <wsa:To>http://localhost:7090/sam/webservice</wsa:To>
    <wsman:ResourceURI >
        http://ns.ca.com/2009/01/usm-data/Queue
    </wsman:ResourceURI>
    <wsman:SelectorSet>
        <wsman:Selector Name="ASBOID.source">4503599627370496</wsman:Selector>
        <wsman:Selector Name="ASBOID.id">2</wsman:Selector>
    </wsman:SelectorSet>
    <wsa:Action>
        http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
    </wsa:Action>
    <wsa:ReplyTo>
    <wsa:Address>
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
        <wsa:MessageID>uuid:78372ad8-46e7-4d27-b632-7e2de827f29a</wsa:MessageID>
</env:Header>
<env:Body/>
</env:Envelope>
```

Example 2: Get a Queue Response

The following example SOAP message shows how you can get a queue response:

```
<env:Envelope>
<env:Header>
    <wsa:Action>
        http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
    </wsa:Action>
    <wsa:MessageID>
        uuid:607204b7-daf5-4ba1-966a-ebab3628b498
    </wsa:MessageID>
```

```
    <wsa:RelatesTo>uuid:78372ad8-46e7-4d27-b632-7e2de827f29a</wsa:RelatesTo>
    <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:
    To>
</env:Header>
<env:Body>
    <ns13:Queue>
    <ssa_escalation_policy_id_1>2</ssa_escalation_policy_id_1>
    <sam_alert_queue_id>0x13400000000002</sam_alert_queue_id>
    <USMID>4503599627370496:2</USMID>
    <ASBOID>
        <source>4503599627370496</source>
        <id>2</id>
    </ASBOID>
    <ns13:item_name>WebServices Created Queue</ns13:item_name>
    <ns13:item_description>WS alarm queue10</ns13:item_description>
    <ns13:item_creation_date>2012-02-15T13:17:54.713+11:00</ns13:item_creation_da
    te>
    <ns13:item_creation_user>symbe01</ns13:item_creation_user>
    <ns13:CriteriaXml><![CDATA[<attr-filter><and><equals-ignore-case><attribute
    id="0x11f57"><value>symbe01</value></attribute></equals-ignore-case><equals-i
    gnore-case><attribute
    id="0x12a08"><value>Windows</value></attribute></equals-ignore-case></and></a
    ttr-filter>]]></ns13:CriteriaXml>
    <ns13:CriteriaDrool>package com.ca.sam.manager.rules
    import com.ca.sam.manager.rules.AlarmObject;
    rule "Queue 2"
        when
                $alarm : AlarmObject( ((assignedTo matches "(?i)symbe01" &amp;&amp;
situationType matches "(?i)Windows" ))  )
        then
                $alarm.assignQueue(2);
    end</ns13:CriteriaDrool>
    <ns13:CompileTime>2012-02-28T08:21:21.797+11:00</ns13:CompileTime>
    <ns13:Priority>2</ns13:Priority>
    <ns13:NumberOfAlerts>0</ns13:NumberOfAlerts>
    <ns13:CountMinor>0</ns13:CountMinor>
    <ns13:CountMajor>0</ns13:CountMajor>
    <ns13:CountCritical>0</ns13:CountCritical>
    <ns13:CountDown>0</ns13:CountDown>
    </ns13:Queue>
</env:Body>
</env:Envelope>
```

Example 3: Create a Queue

The following example SOAP message is a Create request to create a queue:

```
<env:Envelope>
<env:Header>
    <wsa:ReplyTo>
    <wsa:Address>
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>uuid:697da93f-3bc4-479f-8007-fcd0d860d2cb</wsa:MessageID>
    <wsa:To>http://au-symbe01-w8e2:7090/sam/webservice</wsa:To>
    <wsman:ResourceURI>http://ns.ca.com/2009/01/usm-data/Queue
    </wsman:ResourceURI>
    <wsman:OperationTimeout>P0Y0M0DT0H0M30.000S</wsman:OperationTimeout>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Create</wsa:Action
    >
    </env:Header>
    <env:Body>
    <ns13:Queue>
        <ns13:item_name>NewQueue</ns13:item_name>
        <ns13:item_description>My New Queue description</ns13:item_description>
        <ns13:item_creation_user>symbe01</ns13:item_creation_user>
        <ns13:CriteriaXml><![CDATA[<attr-filter><and><equals-ignore-case><attribu
        te
        id="0x11f57"><value>symbe01</value></attribute></equals-ignore-case><equa
        ls-ignore-case><attribute
        id="0x12a08"><value>Windows</value></attribute></equals-ignore-case></and
        ></attr-filter>]]></ns13:CriteriaXml>
    </ns13:Queue>
</env:Body>
</env:Envelope>
```

Example 4: Update a Queue

The following example SOAP message is a Put request that shows how you can update a queue:

```
<env:Envelope>
<env:Header>
    <wsa:To>http://localhost:7090/sam/webservice</wsa:To>
    <wsman:ResourceURI>http://ns.ca.com/2009/01/usm-data/Queue</wsman:ResourceURI
    >
    <wsman:OperationTimeout>P0Y0M0DT0H0M30.000S</wsman:OperationTimeout>
    <wsman:SelectorSet>
        <wsman:Selector Name="ASBOID.source">4503599627370496</wsman:Selector>
        <wsman:Selector Name="ASBOID.id">2</wsman:Selector>
    </wsman:SelectorSet>
```

```
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Put</wsa:Action>
    <wsa:ReplyTo>
        <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonym
        ous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>uuid:f4ea8e55-a4bc-4faf-9dd1-010d1767c8f4</wsa:MessageID>
    <wsman:FragmentTransfer>item_description</wsman:FragmentTransfer>
</env:Header>
<env:Body>
    <ns11:Queue>
        <ASBOID>
        <source>4503599627370496</source>
        <id>2</id>
        </ASBOID>
        <ns11:item_description>Updated desc</ns11:item_description>
    </ns11:Queue>
</env:Body>
</env:Envelope>
```

Example 5: Delete a Queue

The following example SOAP message is a Delete request that shows how you can delete a queue:

```
<env:Envelope>
<env:Header>
    <wsa:To>http://localhost:7090/sam/webservice</wsa:To>
    <wsman:ResourceURI>http://ns.ca.com/2009/01/usm-data/Queue</wsman:ResourceURI
    >
    <wsman:OperationTimeout>P0Y0M0DT0H0M30.000S</wsman:OperationTimeout>
    <wsman:SelectorSet>
    <wsman:Selector Name="ASBOID.source">4503599627370496</wsman:Selector>
    <wsman:Selector Name="ASBOID.id">2</wsman:Selector>
    </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete</wsa:Ac
        tion>
    <wsa:ReplyTo>
        <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonym
        ous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>uuid:f4ea8e55-a4bc-4faf-9dd1-010d1767c8f4</wsa:MessageID>
    <wsman:FragmentTransfer>item_description</wsman:FragmentTransfer>
</env:Header>
<env:Body/>
</env:Envelope>
```

Customer Web Services

This section provides information about the operations that you can perform in CA SOI using the Customer web services resource.

Note: WS-MAN web services should be considered obsolete. We recommend using the Customer (see page 27) REST web services instead.

Customer Web Services Overview

Customer web services resource uses the USM 01-2009 schema to perform customer-related operations in CA SOI. A customer in CA SOI is any consumer of a managed service. The CA SOI administrator creates customers and associates them with service models to see the impact of service degradation on the customer.

Use the following endpoint URI when invoking the customer web services resource:

```
http://ns.ca.com/2009/01/usm-data/Customer
```

The following WSDL file outlines the available operations:

```
http://samanager:port/sam/webservice/wsdls/usm2.wsdl
```

Access the USM 01-2009 schema as follows:

```
http://samanager:port/sam/webservice/schemas/usm2.xsd
```

Note: For more information about customers, see the *Service Modeling Best Practices Guide*.

Get a Customer

Use the Get request to retrieve a specific customer. The following selectors are required as part of the request to identify a unique customer:

ASBOID.id

Uniquely identifies the customer using the Action ID value. Derive this value using an Enumerate operation.

ASBOID.source

Defines the DomainID of the CA SOI model repository. This value is constant for the SA Manager. Derive the value using an Enumerate operation.

To get a customer, use the following properties in the request:

Operation: Get

Endpoint: http://*samanager:port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/Customer

Selector: ASBOID.id

Selector: ASBOID.source

The CustomerHandlerImpl.Get() method returns the customer based on the selectors.

Get a List of Customers

To retrieve a list of customers, the web services use a combination of WS-Management Enumeration and Pull operations. You can filter the returned list using the WS-Management Filter element to pass a valid XPath expression to limit the number of customers returned.

To get a list of customers, use the following properties in the request:

Operation: Enumerate & Pull

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/Customer

Selector: Null

The CustomerIteratorImpl() method creates a collection of all customers in CA SOI. The Pull operation retrieves customers in batches as defined by the MaxElements tag.

Customer Web Services Examples

The following example shows the SOAP messages of the customer web services.

Example: Get a Customer

The following example SOAP message is a Get request to retrieve a customer:

```
<env:Envelope>
  <env:Header>
        <wsa:To>http://localhost:7090/sam/webservice</wsa:To>
        <wsman:ResourceURI>
             http://ns.ca.com/2009/01/usm-data/Customer
        </wsman:ResourceURI>
        <wsman:SelectorSet>
             <wsman:Selector Name="ASBOID.source">4503599627370496
        </wsman:Selector>
             <wsman:Selector Name="ASBOID.id">1</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>
             http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
        </wsa:Action>
        <wsa:ReplyTo>
          <wsa:Address>
             http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
          </wsa:Address>
        </wsa:ReplyTo>
        <wsa:MessageID>
             uuid:78372ad8-46e7-4d27-b632-7e2de827f29a
        </wsa:MessageID>
  </env:Header>
<env:Body/>
</env:Envelope>
```

The response of the Get request is as follows:

```
<env:Envelope>
  <env:Header>
     <wsa:Action>
        http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
     </wsa:Action>
     <wsa:MessageID>
        uuid:6b58e83b-b02a-4a5e-8650-61fdc8842a77
     </wsa:MessageID>
     <wsa:RelatesTo>
        uuid:78372ad8-46e7-4d27-b632-7e2de827f29a
     </wsa:RelatesTo>
```

```
      <wsa:To>
         http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
      </wsa:To>
   </env:Header>
   <env:Body>
      <ns13:Customer>
         <USMID>4503599627370496:1</USMID>
         <ASBOID>
             <source>4503599627370496</source>
             <id>1</id>
         </ASBOID>
         <ns13:item_name>customer1</ns13:item_name>
         <ns13:item_description>first customer</ns13:item_description>
         <ns13:priority>10</ns13:priority>
         <ns13:identifier>c12345</ns13:identifier>
         <ns13:quality>0</ns13:quality>
         <ns13:risk>5</ns13:risk>
         <ns13:health>1</ns13:health>
         <ns13:topLevel>true</ns13:topLevel>
         <ns13:services>3</ns13:services>
         <ns13:services>16</ns13:services>
       </ns13:Customer>
    </env:Body>
</env:Envelope>
```

Alert Web Services

This section describes the operations performed in alert web services.

Note: WS-MAN web services should be considered obsolete. We recommend using the Alert (see page 25) REST web services instead.

Alert Web Services Overview

Alert web services let you interact with CA SOI alerts. Use the following endpoint URI when invoking the alert web services resource:

`http://ns.ca.com/2009/07/usm-core/Alert`

The following WSDL file outlines the available operations:

`http://samanager:port/sam/webservice/wsdls/usm-core-200907.wsdl`

Access the USM schema as follows:

`http://samanager:port/sam/webservice/schemas/usm-core-200907.xsd`

Using the alert web services requires basic knowledge of the USM schema and its properties. In addition to the schema itself, HTML documentation is available. The USM schema documentation is available on the online bookshelf.

Get an Alert

Use the Get request to retrieve an alert. The following selectors are required as part of the request to identify a unique instance of an alert:

MdrProduct

Defines the connector data source. Each connector has a specific MdrProduct value formatted as a five-digit number prefixed by 'CA:'. For example, the MdrProduct value for resources created by web services is CA:09996. For a list of MdrProduct values, see the *Connector Guide*.

MdrProdInstance

Defines the host name associated with the resource.

MdrElementID

Defines a value that uniquely identifies the resource.

Note: This request also retrieves the root cause property ssa_rootcause_alert_id of service-related alerts to indicate the alert that is the root cause alert, and ssa_is_rootcause property of CI alerts to indicate whether CI alerts are root cause alerts.

To get an alert, use the following properties in the request:

Operation: Get

Endpoint: http://*samanager*:*port*/sam/webservice

ResourceURI: http://ns.ca.com/2009/07/usm-core/Alert

Selector: MdrProduct

Selector: MdrProdInstance

Selector: MdrElementID

The AlertHandlerImpl.Get() method returns an alert of the type of USM resource that represents the CA SOI alert in USM terms.

Get a List of Alerts

To retrieve a list of alerts, the web services use a combination of WS-Management Enumeration and Pull operations. You can filter the returned list using the WS-Management Filter element to pass a valid XPath expression to limit the number and type of alerts returned.

When you retrieve a list of alerts, the web service returns the AlertID property for each alert in the MdrElementID property. For more information about a specific alert, use the retrieved AlertID value to get an alert.

To get a list of alerts, use the following properties in the request:

Operation: Enumerate & Pull

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Alert

Selector: Null

Note: You can also retrieve a list of alerts associated with a service and all of its children CIs. Use the Recursive selector to achieve this task. For example, if you specify MdrElementID=2 and Recursive=True, then if MdrElementID represents a service, the alerts for the service and all of its children CIs are returned in the list.

The AlertIteratorImpl.java program retrieves the list of current alerts from the alert repository context and the CI with which the alert is associated from the Model Repository and creates a collection object that contains the alert list. An enumeration context is returned, which you can use to retrieve the alerts through the Pull operation.

Create an Alert

Use the Create operation to create an alert and associate it with a CI in CA SOI. You define the alert type and USM property values for the new alert in the body of the request.

Note: For information about the required properties to include for an alert, see the USM schema documentation. The USM schema documentation is provided on the online bookshelf.

To create an alert, use the following properties in the request:

Operation: Create

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Alert

Selector: Null

The AlertHandlerImpl.Create() method extracts the alert detail from the body of the SOAP message, and creates the alert in CA SOI. If successful, the CreateResponse includes a unique AlertID in the MdrElementID of the response.

Update an Alert

Use the Put operation to update the writable attributes of an alert. Perform the update by passing in all of the USM properties and their new values in the body of the request.

Note: Refer to the USM schema for a list of USM properties and their appropriate values. The USM schema documentation is provided on the online bookshelf.

To update an alert, use the following properties in the request:

Operation: Put

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Alert

Selector: Null

The AlertHandlerImpl.Put() method examines the contents of the SOAP header, and if the request contains a Fragment element, only the attributes specified in the Fragment are updated. Otherwise, all writable alert attributes are updated. A PutResponse Operation is returned after successful updates.

Clear an Alert

Use the Delete operation to clear an alert. Pass the USM properties of the alert to delete in the body of the request.

Note: For information about the required properties to include for an alert, see the USM schema documentation.

To clear an alert, use the following properties in the request:

Operation: Delete

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/07/usm-core/Alert

Selector: MdrElementID

The AlertHandlerImpl.Delete() method uses MdrElementID to select the appropriate alert and runs the clear backend function to delete the alert from the Operations Console.

If the alert is successfully cleared, the DeleteResponse operation is returned. Otherwise, a SOAP fault exception is returned.

Alert Web Services Examples

The following examples show the SOAP messages of many of the available alert web services.

Example: Get an alert

The following example SOAP message is a Get request to retrieve a specific alert:

```
<env:Envelope>
<env:Header>
    <wsa:To>http://localhost:7090/sam/webservice</wsa:To>
    <wsman:ResourceURI>http://ns.ca.com/2009/07/usm-core/Alert</wsman:ResourceURI
    >
    <wsman:SelectorSet>
        <wsman:Selector Name="AlertID">32</wsman:Selector>
    </wsman:SelectorSet>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Get</wsa:Action>
    <wsa:ReplyTo>
        <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonym
        ous</wsa:Address>
```

```
    </wsa:ReplyTo>
    <wsa:MessageID>uuid:079e3c5a-7c5e-41a0-b5a8-992238aa55e3</wsa:MessageID>
</env:Header>
<env:Body/>
</env:Envelope>
```

The SOAP response to this request is as follows:

```
<env:Envelope>
<env:Header>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse</wsa:A
    ction>
    <wsa:MessageID>uuid:09af3775-21b1-475e-91ca-4a3b2da059c8</wsa:MessageID>
    <wsa:RelatesTo>uuid:079e3c5a-7c5e-41a0-b5a8-992238aa55e3</wsa:RelatesTo>
    <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:
    To>
</env:Header>
<env:Body>
    <ns14:Alert>
        <ns14:MdrProduct>CA:00047</ns14:MdrProduct>
        <ns14:MdrProdInstance>AB-XY01-W8</ns14:MdrProdInstance>
        <ns14:MdrElementID>32</ns14:MdrElementID>
        <ns14:UrlParams>http://AB-XY01-W8.ca.com:8080?id=A00002</ns14:UrlParams>
        <ns14:OccurrenceTimestamp>2012-03-05T09:58:59.701+11:00</ns14:OccurrenceT
        imestamp>
        <ns14:ReportTimestamp>2012-03-05T09:58:59.000+11:00</ns14:ReportTimestamp
        >
        <ns14:AlertType>Risk</ns14:AlertType>
        <ns14:Severity>Minor</ns14:Severity>
        <ns14:AlertedMdrProduct>CA:00047</ns14:AlertedMdrProduct>
        <ns14:AlertedMdrProdInstance>AB-XY01-W8</ns14:AlertedMdrProdInstance>
        <ns14:AlertedMdrElementID>19</ns14:AlertedMdrElementID>
        <ns14:Summary>UC_Server has an infrastructure alarm..</ns14:Summary>
        <ns14:Message>The Detailed message associated with this
        alert..</ns14:Message>
        <ns14:IsAcknowledged>false</ns14:IsAcknowledged>
        <ns14:Assignee/>
        <ns14:RelatedIncident/>
        <ssa_instance_id>ComputerSystem:dnsname,ucserver.ca.com:UCServer</ssa_ins
        tance_id>
        <ssa_classname>SA_Server</ssa_classname>
        <ssa_class_id>21</ssa_class_id>
        <ssa_connector_id>2</ssa_connector_id>
        <ssa_connector_name>UniversalConnector running on host
        AB-XY01-W8.ca.com</ssa_connector_name>
        <ssa_ticket_props/>
        <ssa_ticket_id_url/>
        <ssa_ticket_url/>
```

```
        <ssa_userattribute_1/>
        <ssa_userattribute_2/>
        <ssa_userattribute_3/>
        <ssa_userattribute_4/>
        <ssa_userattribute_5/>
        <ssa_is_rootcause>true</ssa_is_rootcause>
        <ssa_domain_id>4503599627370496</ssa_domain_id>
        <ssa_queue_id>2</ssa_queue_id>
    </ns14:Alert>
</env:Body>
</env:Envelope>
```

Note: The tag structure for the root cause property of service-related alert and CI alert is as follows. In the following example code snippet, false specifies that the CI alerts are not the root cause alerts and 3 implies the ID of the service-related alert that is the root cause alert :

```
    <ssa_is_rootcause>false</ssa_is_rootcause>
    <ssa_rootcause_alert_id>3</ssa_rootcause_alert_id>
```

Example: Update the ticket number of an alert

The following example SOAP message is a Put request for the RelatedIncident property of the Alert:

```
<env:Envelope>
<env:Header>
    <wsa:ReplyTo>
        <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonym
        ous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>uuid:4add8974-25a0-4fc4-84f6-c7f88158f8f7</wsa:MessageID>
    <wsa:To>http://localhost:7090/sam/webservice</wsa:To>
    <wsman:ResourceURI>http://ns.ca.com/2009/07/usm-core/Alert</wsman:ResourceURI
    >
    <wsman:OperationTimeout>PT30.000S</wsman:OperationTimeout>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Put</wsa:Action>
</env:Header>
<env:Body>
    <ns14:Alert>
        <ns14:MdrProduct>CA:00047</ns14:MdrProduct>
        <ns14:MdrProdInstance>symbe01-5</ns14:MdrProdInstance>
        <ns14:MdrElementID>6</ns14:MdrElementID>
        <ns14:UrlParams>http://symbe01-5.ca.com:8080?id=mdr_id</ns14:UrlParams>
        <ns14:AlertType>Quality</ns14:AlertType>
        <ns14:Severity>Critical</ns14:Severity>
        <ns14:AlertedMdrProduct>CA:00047</ns14:AlertedMdrProduct>
```

```
            <ns14:AlertedMdrProdInstance>symbe01-5</ns14:AlertedMdrProdInstance>
            <ns14:AlertedMdrElementID>4</ns14:AlertedMdrElementID>
            <ns14:Summary>UC_Server has a updated client alarm..</ns14:Summary>
            <ns14:Message>UC_Server has a detailed client alarm..</ns14:Message>
            <ns14:IsAcknowledged>false</ns14:IsAcknowledged>
            <ns14:Assignee/>
            <ns14:RelatedIncident>T12345</ns14:RelatedIncident>
            <ssa_ticket_props/>
            <ssa_userattribute_1/>
            <ssa_userattribute_2/>
            <ssa_userattribute_3/>
            <ssa_userattribute_4/>
            <ssa_userattribute_5/>
        </ns14:Alert>
</env:Body>
</env:Envelope>
```

Subscribe to Notifications for Alert Changes

To subscribe to notifications for changes to Alert, use the Notification web services. The Alert resource supports the WS-Eventing functionality that enables a web client to subscribe to notification events when an alert is created, deleted, or updated.

Note: For more information, see Notification Web Services.

Propagation Policy Web Services

This section provides information about the operations performed in propagation policy web services.

Propagation Policy Web Services Overview

Propagation policy web services use the USM 01-2009 schema to perform operations on propagation policies, which define specific policy instructions for how impact propagates in related CIs in a service.

Propagation policy was known as relationship policy in previous versions of CA SOI. The following propagation types require propagation policy:

- Custom (formerly DependsOn)
- Operative (formerly Requires)

The propagation policy web services let you interact with Custom propagation policy assigned to a relationship and propagation between CIs. Use the term DependsOn in web service requests to refer to Custom propagation policy.

Note: The web services cannot interact with Operative propagation policy.

Use the following endpoint URI when invoking the propagation policy web services resource:

`http://ns.ca.com/2009/01/usm-data/RelationshipPolicy`

The following WSDL file outlines the available operations:

`http://samanager:port/sam/webservice/wsdls/usm2.wsdl`

Access the USM 01-2009 schema as follows:

`http://samanager:port/sam/webservice/schemas/usm2.xsd`

Note: For more information about propagation policy, see the *Service Modeling Best Practices Guide*.

Get a Propagation Policy

Use the Get request to retrieve a propagation policy associated with a custom propagation type. View the propagation policy in the Service Modeler on the Policies tab.

The following selectors are required to identify a unique instance of a propagation policy:

ASBOID.id

Uniquely identifies the propagation policy using the Policy ID value. Derive this value using an Enumerate operation.

ASBOID.source

Defines the DomainID of the CA SOI model repository. This value is constant for the SA Manager. Derive the value using an Enumerate operation.

To get a propagation policy, use the following properties in the request:

Operation: Get

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/RelationshipPolicy

Selector: ASBOID.id

Selector: ASBOID.source

Get a List of Propagation Policies

To retrieve a list of propagation policies for a service, the web services use a combination of WS-Management Enumeration and Pull operations. You can filter the returned list using the WS-Management Filter element to pass a valid XPath Expression to limit the number and type of propagation policies returned.

Use the item_name selector to filter the collection by service. Format the selector as follows:

`SA_Service:`*`servicename`*

servicename

Defines the name of the service.

To get a list of propagation policies, use the following properties in the request:

Operation: Enumerate & Pull

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/RelationshipPolicy

Selector: item_name

Create a Propagation Policy

Use the Create operation to create a propagation policy between CIs in CA SOI that belong to an existing service. Include the following information in the body of the request:

- CI ID for the service in which the propagation exists
- CI ID for the source CI
- CI ID for the target CI or CIs
- Policy properties, such as impact and thresholds

The associated propagation type must be custom (referred to as DependsOn in the web service request).

Note: For information about the required properties to include for a propagation policy, see the USM 01-2009 schema. For more information about property names and formatting, see Propagation Policy Web Services Examples (see page 97).

To create a propagation policy, use the following properties in the request:

Operation: Create

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/RelationshipPolicy

Selector: null

Update a Propagation Policy

Use the Put operation to update the writable attributes of a propagation policy. You can update all writable attributes or only certain attributes that you specify in a Fragment statement. To identify the unique propagation policy, pass the ASBOID.id and ASBOID.source propagation policy values (see page 93) as selectors.

You can update the following propagation policy attributes:

- item_description
- sam_policy_type
- sam_policy_threshold_1 (Rule 1 threshold)
- sam_policy_threshold_2 (Rule 2 threshold)
- sam_policy_threshold_3 (Rule 3 threshold)
- sam_policy_threshold_4 (Rule 4 threshold)
- sam_policy_action_1 (Rule 1 impact)
- sam_policy_action_2 (Rule 2 impact)
- sam_policy_action_3 (Rule 3 impact)
- sam_policy_action_4 (Rule 4 impact)
- sam_policy_bnode_id (CIs that the policy applies to)

The valid sam_policy_type values are as follows:

Any

Sets the impact of the parent item when any CIs associated with the policy have the impact specified in the rule.

All

Sets the impact of the parent item when all CIs have the impact specified in the rule.

Percentage

Sets the impact of the parent item based on a percentage of CIs that have the impact specified in the rule.

Average

Sets the impact of the parent item based on the average impact values of CIs associated with the policy.

To update a propagation policy, use the following properties in the request:

Operation: Put

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/RelationshipPolicy

Selector: ASBOID.id

Selector: ASBOID.source

Fragment: any attributes

Delete a Propagation Policy

Use the Delete operation to delete a propagation policy. Use the propagation policy ASBOID.id and ASBOID.source values (see page 93) as selectors.

To delete a propagation policy, use the following properties in the request:

Operation: Delete

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/RelationshipPolicy

Selector: ASBOID.id

Selector: ASBOID.source

Propagation Policy Web Services Examples

The following examples show the SOAP messages of many of the available propagation policy web services.

Example: Create a custom propagation policy

The following example SOAP message is a Create request to create a custom propagation policy:

```
<env:Header>
    <wsa:ReplyTo xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
        uuid:e5d2bbd4-04dc-42d0-ab7a-2bd1692c87a0
    </wsa:MessageID>
    <wsa:To xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
        http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
        http://ns.ca.com/2009/01/usm-data/RelationshipPolicy
    </wsman:ResourceURI>
    <wsman:OperationTimeout xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        PT30.000S
    </wsman:OperationTimeout>
    <wsa:Action xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
    </wsa:Action>
</env:Header>
<env:Body>
    <ns11:RelationshipPolicy xmlns:ns10="http://www.w3.org/2003/05/soap-envelope"
    xmlns:ns11="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:ns3="http://schemas.xmlsoap.org/ws/2004/08/eventing"
```

```
    xmlns:ns4="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
    xmlns:ns5="http://schemas.xmlsoap.org/ws/2004/09/transfer"
    xmlns:ns6="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
    xmlns:ns7="http://schemas.xmlsoap.org/ws/2004/09/mex"
    xmlns:ns8="http://schemas.wiseman.dev.java.net/metadata/messagetypes"
    xmlns:ns9="http://schemas.sam.ca.com/webservice/1/alarm.xsd">
            <sam_policy_type>Percentage</sam_policy_type>
            <sam_policy_service_id>22</sam_policy_service_id>
            <sam_policy_anode_id>22</sam_policy_anode_id>
            <sam_policy_action_1>2</sam_policy_action_1>
            <sam_policy_action_2>3</sam_policy_action_2>
            <sam_policy_action_3>0</sam_policy_action_3>
            <sam_policy_action_4>0</sam_policy_action_4>
            <sam_policy_threshold_1>15</sam_policy_threshold_1>
            <sam_policy_threshold_2>35</sam_policy_threshold_2>
            <sam_policy_threshold_3>01</sam_policy_threshold_3>
            <sam_policy_threshold_4>01</sam_policy_threshold_4>
            <sam_usm_reltype_id>105</sam_usm_reltype_id>
            <sam_policy_bnode_id>23</sam_policy_bnode_id>
            <ns11:item_name>SAM2818_2</ns11:item_name>
            <ns11:item_description>SAM2818_2 Policy</ns11:item_description>
    </ns11:RelationshipPolicy>
</env:Body>
</env:Envelope>
```

The bold syntax shows the properties defined for the custom propagation policy. The SOAP response to this request is as follows:

```
<env:Header>
    <wsa:Action env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
    </wsa:Action>
    <wsa:MessageID env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        uuid:449e3af7-12e2-4f91-a3f4-360a4dffe484
    </wsa:MessageID>
    <wsa:RelatesTo xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
        uuid:e5d2bbd4-04dc-42d0-ab7a-2bd1692c87a0
    </wsa:RelatesTo>
    <wsa:To env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
</env:Header>
    <env:Body>
```

```
    <wxf:ResourceCreated xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
    <wsa:Address env:mustUnderstand="true">
        http://localhost:7090/sam/webservice/
    </wsa:Address>
    <wsa:ReferenceParameters>
    <wsman:ResourceURI>
        http://ns.ca.com/2009/01/usm-data/RelationshipPolicy
    </wsman:ResourceURI>
    <wsman:SelectorSet>
       <wsman:Selector Name="ASBOID.id">8</wsman:Selector>
       <wsman:Selector Name="ASBOID.source">4503599627370496</wsman:Selector>
    </wsman:SelectorSet>
    </wsa:ReferenceParameters>
    </wxf:ResourceCreated>
</env:Body>
</env:Envelope>
```

The bold syntax shows the returned selector properties for the created custom propagation policy.

Example: Delete a propagation policy

The following example SOAP message is a Delete request to delete a propagation policy:

```
<env:Header>
    <wsa:ReplyTo xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
        uuid:202fc0d4-7bf6-4e28-87cf-1a11afef97db
    </wsa:MessageID>
    <wsa:To xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
    http://localhost:7090/sam/webservice</wsa:To>
    <wsman:ResourceURI xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
        http://ns.ca.com/2009/01/usm-data/RelationshipPolicy
    </wsman:ResourceURI>
    <wsman:OperationTimeout xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
        PT30.000S
    </wsman:OperationTimeout>
    <wsman:SelectorSet xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
       <wsman:Selector Name="ASBOID.source">4503599627370496</wsman:Selector>
       <wsman:Selector Name="ASBOID.id">8</wsman:Selector>
    </wsman:SelectorSet>
```

```
    <wsa:Action xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
    </wsa:Action>
</env:Header>
<env:Body/>
</env:Envelope>
```

The bold syntax shows the selectors that uniquely identify the propagation policy. The SOAP response to this request is as follows:

```
<env:Header>
    <wsa:Action env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse
    </wsa:Action>
    <wsa:MessageID env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        uuid:baa81209-3bc6-4e67-aed4-2050a89b8279
    </wsa:MessageID>
    <wsa:RelatesTo xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
    uuid:202fc0d4-7bf6-4e28-87cf-1a11afef97db</wsa:RelatesTo>
    <wsa:To env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
</env:Header>
<env:Body/>
</env:Envelope>
```

Escalation Policy Web Services

This section provides information about the operations performed in escalation policy web services.

Note: WS-MAN web services should be considered obsolete. We recommend using the Escalation Policy (see page 29) REST web services instead.

Escalation Policy Web Services Overview

Escalation policy web services use the USM 01-2009 schema to perform operations on escalation policies, which define specific policy instructions for when to escalate alerts. Escalation policies can be global or service-specific.

Use the following endpoint URI when invoking the escalation policy web services resource:

`http://ns.ca.com/2009/01/usm-data/EscalationPolicy`

The following WSDL file outlines the available operations:

`http://samanager:port/sam/webservice/wsdls/usm2.wsdl`

Access the USM 01-2009 schema as follows:

`http://samanager:port/sam/webservice/schemas/usm2.xsd`

Note: For more information about escalation policy, see the *Event and Alert Management Best Practices Guide*.

Get an Escalation Policy

Use the Get request to retrieve an escalation policy. The following selectors are required to identify a unique instance of an escalation policy:

ASBOID.id

Uniquely identifies the escalation policy using the Policy ID value. Derive this value using an Enumerate operation.

ASBOID.source

Defines the DomainID of the CA SOI model repository. This value is constant for the SA Manager. Derive the value using an Enumerate operation.

To get an escalation policy, use the following properties in the request:

Operation: Get

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/EscalationPolicy

Selector: ASBOID.id

Selector: ASBOID.source

Get a List of Escalation Policies

To retrieve a list of escalation policies, the web services use a combination of WS-Management Enumeration and Pull operations. You can filter the returned list using the WS-Management Filter element to pass a valid XPath Expression to limit the number and type of escalation policies returned.

To get a list of escalation policies, use the following properties in the request:

Operation: Enumerate & Pull

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/EscalationPolicy

Selector: null

Create an Escalation Policy

Use the Create operation to create an escalation policy. You define escalation policy properties in the body of the request, such as the following:

- Policy type (global or non-global)
- The types of alerts to which the policy applies

  Note: Global policy type specifies that the policy applies to all alerts. Non-global type specifies that the policy applies to alerts of an assigned service or alert queue. When you create a queue (see page 76), you can associate it with an existing escalation policy by using the attribute <ssa_escalation_policy_id_1>.
- Strings that indicate policy rules

Note: For information about the required properties to include for an escalation policy, see the USM 01-2009 schema. For more information about property names and formatting, see Escalation Policy Web Services Examples (see page 104).

To create an escalation policy, use the following properties in the request:

Operation: Create

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/EscalationPolicy

Selector: null

Update an Escalation Policy

Use the Put operation to update the writable attributes of an escalation policy. You can update all writable attributes or only certain attributes that you specify in a Fragment statement. To identify the unique escalation policy, pass the ASBOID.id and ASBOID.source escalation policy values (see page 101) as selectors.

You can update the following escalation policy attributes:

- sam_policy_enabled (Enable)
- sam_is_global (Global or Non-global)

  Note: You can create an escalation policy as a non-global policy and add the attribute <sam_is_local_service_name> to identify the service.
- sam_root_cause_alarm
- sam_symptom_alarm
- sam_service_alarm
- sam_infrastructure_alarm
- sam_maintenance_alarm
- sam_service_maintenance_alarm

To update an escalation policy, use the following properties in the request:

Operation: Put

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/EscalationPolicy

Selector: ASBOID.id

Selector: ASBOID.source

Fragment: any property

Delete an Escalation Policy

Use the Delete operation to delete an escalation policy. Use the escalation policy ASBOID.id and ASBOID.source values (see page 101) as selectors.

To delete an escalation policy, use the following properties in the request::

Operation: Delete

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/EscalationPolicy

Selector: ASBOID.id

Selector: ASBOID.source

Escalation Policy Web Services Examples

The following examples show the SOAP messages of many of the available escalation policy web services.

Example: Create a global escalation policy

The following example SOAP message is a Create request to create a global escalation policy:

```
<env:Header>
    <wsa:ReplyTo xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
        uuid:51f6d9ca-767f-4be7-bd3e-d1f13cdd6759
    </wsa:MessageID>
    <wsa:To xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
        http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
    http://ns.ca.com/2009/01/usm-data/EscalationPolicy
    </wsman:ResourceURI>
    <wsman:OperationTimeout xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        PT30.000S
    </wsman:OperationTimeout>
```

```
    <wsa:Action xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
    </wsa:Action>
</env:Header>
<env:Body>
    <ns11:EscalationPolicy xmlns:ns10="http://www.w3.org/2003/05/soap-envelope"
    xmlns:ns11="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns12="http://ns.ca.com/2009/07/usm-core"
    xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:ns3="http://schemas.xmlsoap.org/ws/2004/08/eventing"
    xmlns:ns4="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
    xmlns:ns5="http://schemas.xmlsoap.org/ws/2004/09/transfer"
    xmlns:ns6="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
    xmlns:ns7="http://schemas.xmlsoap.org/ws/2004/09/mex"
    xmlns:ns8="http://schemas.wiseman.dev.java.net/metadata/messagetypes"
    xmlns:ns9="http://schemas.sam.ca.com/webservice/1/alarm.xsd">
        <sam_is_global>true</sam_is_global>
        <sam_policy_enabled>true</sam_policy_enabled>
        <sam_root_cause_alarm>true</sam_root_cause_alarm>
        <sam_symptom_alarm>true</sam_symptom_alarm>
        <sam_service_alarm>true</sam_service_alarm>
        <sam_infrastructure_alarm>true</sam_infrastructure_alarm>
        <sam_maintenance_mode>true</sam_maintenance_mode>
        <sam_service_maintenance_mode>false</sam_service_maintenance_mode>
        <sam_schedule_type>0</sam_schedule_type>
        <sam_calendar_id>0</sam_calendar_id>
        <sam_xml_rule_string>&amp;lt;esc-policy&amp;gt;&amp;lt;/esc-policy&amp;gt
    ;
        </sam_xml_rule_string>
        <sam_drl_rule_string>package com.ca.sam.manager.rules
         import com.ca.sam.manager.rules.AlarmObject;
         rule "d68f435d-9a7e-4926-b383-d2e4676920ae"
            when
            $alarm : AlarmObject()</sam_drl_rule_string>
        <ns11:item_name>Escalation_Policy_eleven</ns11:item_name>
        <ns11:item_description>Policy created by the
        webservice...</ns11:item_description>
    </ns11:EscalationPolicy>
</env:Body>
</env:Envelope>
```

The bold syntax defines the following properties for the escalation policy:

- It is global and enabled
- It includes root cause, symptom, service, and infrastructure alerts
- It includes the rules defined in the sam_xml_rule_string and sam_drl_rule_string properties

The SOAP response to this request is as follows:

```
<env:Header>
    <wsa:Action env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
    </wsa:Action>
    <wsa:MessageID env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        uuid:be232d8e-d755-4627-9365-643dbe47cf85
    </wsa:MessageID>
    <wsa:RelatesTo xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
        uuid:51f6d9ca-767f-4be7-bd3e-d1f13cdd6759
    </wsa:RelatesTo>
    <wsa:To env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
</env:Header>
    <env:Body>
       <wxf:ResourceCreated xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
       <wsa:Address
       env:mustUnderstand="true">http://localhost:8888/sam/webservice/
       </wsa:Address>
       <wsa:ReferenceParameters>
       <wsman:ResourceURI>
        http://ns.ca.com/2009/01/usm-data/EscalationPolicy
       </wsman:ResourceURI>
       <wsman:SelectorSet>
            <wsman:Selector Name="ASBOID.id">32</wsman:Selector>
            <wsman:Selector
    Name="ASBOID.source">4503599627370496</wsman:Selector>
       </wsman:SelectorSet>
       </wsa:ReferenceParameters>
       </wxf:ResourceCreated>
    </env:Body>
</env:Envelope>
```

The bold syntax shows the returned selector properties for the created escalation policy.

Example: Get an escalation policy

The following example SOAP message is a Get request to retrieve a specific escalation policy:

```
<env:Header>
    <wsa:To xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
```

```
        http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
        http://ns.ca.com/2009/01/usm-data/EscalationPolicy
    </wsman:ResourceURI>
    <wsman:SelectorSet xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
       <wsman:Selector Name="ASBOID.source">0</wsman:Selector>
       <wsman:Selector Name="ASBOID.id">27</wsman:Selector>
    </wsman:SelectorSet>
    <wsa:Action xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
    </wsa:Action>
    <wsa:ReplyTo xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
    uuid:78372ad8-46e7-4d27-b632-7e2de827f29a
    </wsa:MessageID>
</env:Header>
<env:Body/>
</env:Envelope>
```

The request retrieves the escalation policy with the ASBOID values in the bold SelectorSet syntax. The SOAP response to this request is as follows:

```
<env:Header>
    <wsa:Action env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
    </wsa:Action>
    <wsa:MessageID env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        uuid:a608934d-17a5-4e7a-b5eb-d983bc6e1d8d
    </wsa:MessageID>
    <wsa:RelatesTo xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
        uuid:78372ad8-46e7-4d27-b632-7e2de827f29a
    </wsa:RelatesTo>
    <wsa:To env:mustUnderstand="true"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
```

```
</env:Header>
<env:Body>
   <ns13:EscalationPolicy xmlns:ns13="http://ns.ca.com/2009/01/usm-data" >
      <sam_unique_id>4edb0241-f08a-416d-a7ab-16e32c920651</sam_unique_id>
      <sam_compiled_time>2010-03-16 09:00:01.27</sam_compiled_time>
      <sam_policy_enabled>true</sam_policy_enabled>
      <sam_root_cause_alarm>false</sam_root_cause_alarm>
      <sam_symptom_alarm>false</sam_symptom_alarm>
      <sam_service_alarm>true</sam_service_alarm>
      <sam_infrastructure_alarm>true</sam_infrastructure_alarm>
      <sam_maintenance_mode>true</sam_maintenance_mode>
      <sam_is_global>true</sam_is_global>
      <sam_service_maintenance_mode>false</sam_service_maintenance_mode>
      <sam_schedule_type>2</sam_schedule_type>
      <sam_escalation_schedule_id>0x11000000000001</sam_escalation_schedule_i
d>
      <sam_escalation_schedule_desc>Daily</sam_escalation_schedule_desc>
      <sam_calendar_id>0</sam_calendar_id>
      <sam_xml_rule_string><![CDATA[&lt;esc-policy&gt;&lt;time-filter&gt;&lt;
or&gt;&lt;greater-than&gt;&lt;attribute
id=&quot;0x20027&quot;&gt;&lt;value&gt;30&lt;/value&gt;&lt;/attribute&gt;&lt;
/greater-than&gt;&lt;/or&gt;&lt;/time-filter&gt;&lt;attr-filter&gt;&lt;and&gt
;&lt;equals-ignore-case&gt;&lt;attribute
id=&quot;0x11f57&quot;&gt;&lt;value&gt;symbe01&lt;/value&gt;&lt;/attribute&gt
;&lt;/equals-ignore-case&gt;&lt;/and&gt;&lt;/attr-filter&gt;&lt;/esc-policy&g
t;]]>
      </sam_xml_rule_string>
      <sam_drl_rule_string>package com.ca.sam.manager.rules
import com.ca.sam.manager.rules.AlarmObject;
rule "4edb0241-f08a-416d-a7ab-16e32c920651"
   when
      $alarm : AlarmObject(</sam_drl_rule_string>
      <USMID>0:27</USMID>
      <ASBOID>
         <source>0</source>
         <id>27</id>
      </ASBOID>
      <ns13:item_name>global_3</ns13:item_name>
      <ns13:item_description/>
      <ns13:item_creation_date>2010-03-09T14:28:45.693+11:00
      </ns13:item_creation_date>
      <ns13:item_creation_user>Web Service</ns13:item_creation_user>
   </ns13:EscalationPolicy>
</env:Body>
</env:Envelope>
```

The bold syntax shows the details of the returned escalation policy.

Escalation Action Web Services

This section provides information about the operations performed in escalation action web services.

Note: WS-MAN web services should be considered obsolete. We recommend using the Escalation Policy Action (see page 28) REST web services instead.

Escalation Action Web Services Overview

Escalation action web services use the USM 01-2009 schema to perform operations on escalation actions, which define specific actions to perform when associated escalation policy criteria are met. Examples of available actions include the following:

- Send an email
- Create a help desk ticket
- Run a command
- Create a help desk announcement
- Run the CA Process Automation process
- Clear an alert

Use the following endpoint URI when invoking the escalation action web services resource:

```
http://ns.ca.com/2009/01/usm-data/EscalationpolicyAction
```

The following WSDL file outlines the available operations:

```
http://samanager:port/sam/webservice/wsdls/usm2.wsdl
```

Access the USM 01-2009 schema as follows:

```
http://samanager:port/sam/webservice/schemas/usm2.xsd
```

Note: For more information about escalation actions, see the *Event and Alert Management Best Practices Guide*.

Get an Escalation Action

Use the Get request to retrieve an escalation action. The following selectors are required to identify a unique instance of an escalation action:

ASBOID.id

Uniquely identifies the escalation action using the Action ID value. Derive this value using an Enumerate operation.

ASBOID.source

Defines the DomainID of the CA SOI model repository. This value is constant for the SA Manager. Derive the value using an Enumerate operation.

To get an escalation action, use the following properties in the request:

Operation: Get

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/EscalationpolicyAction

Selector: ASBOID.id

Selector: ASBOID.source

Get a List of Escalation Actions

To retrieve a list of escalation actions, the web services use a combination of WS-Management Enumeration and Pull operations. You can filter the returned list using the WS-Management Filter element to pass a valid XPath Expression to limit the number and type of escalation actions returned.

To get a list of escalation actions, use the following properties in the request:

Operation: Enumerate & Pull

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/EscalationpolicyAction

Selector: null

You can also retrieve a list of escalation actions based on the action name and action type. You can achieve this by using the WS-Management Filter parameter in the Enumerate operation as explained in the following two examples:

- The first example SOAP message snippet explains how you can get a list of escalation types based on the action type. This example limits the list to those action types that are equal to 1, which represents *Tickets*:

```
<env:Body xmlns:EscalationpolicyAction="http://ns.ca.com/2009/01/usm-data">
<wsen:Enumerate xmlns:ns11="http://ns.ca.com/2009/01/usm-data" >
<wsman:Filter
Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">/EscalationpolicyActio
n:EscalationpolicyAction[sam_action_type='1']</wsman:Filter>
<wsman:EnumerationMode>EnumerateObjectAndEPR</wsman:EnumerationMode>
</wsen:Enumerate>
</env:Body>
```

- The second example SOAP message snippet explains how you can get a list of escalation types based on the action name. This example limits the list to those escalation actions that have a name starting with *Tick*:

```
<env:Body xmlns:EscalationpolicyAction="http://ns.ca.com/2009/01/usm-data">
<wsen:Enumerate xmlns:ns11="http://ns.ca.com/2009/01/usm-data" >
<wsman:Filter
Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">/EscalationpolicyActio
n:EscalationpolicyAction[starts-with(EscalationpolicyAction:item_name,'Tick')
]</wsman:Filter>
<wsman:EnumerationMode>EnumerateObjectAndEPR</wsman:EnumerationMode>
</wsen:Enumerate>
</env:Body>
```

Create an Escalation Action

Use the Create request to create an escalation action. You define escalation action properties in the body of the request, such as the following:

- Action type (0-5)
- Action data, such as an email address

Note: For more information about property names and formatting, see Escalation Action Web Services Examples (see page 113).

The valid action type properties are as follows:

0

Corresponds to the Send Email escalation action type.

1

Corresponds to the Create Ticket escalation action type.

2

Corresponds to the Execute Command escalation action type.

3

Corresponds to the Create Announcement escalation action type.

4

Corresponds to the Execute Automated Process escalation action type.

5

Corresponds to the Clear Alert escalation action type.

To create an escalation action, use the following properties in the request:

Operation: Create

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/EscalationpolicyAction

Selector: null

Delete an Escalation Action

Use the Delete operation to delete an escalation action. Use the escalation action ASBOID.id and ASBOID.source values (see page 110) as selectors.

To delete an escalation action, use the following properties in the request:

Operation: Delete

Endpoint: http://*samanager*:*port*/sam/webservice

Resource: http://ns.ca.com/2009/01/usm-data/EscalationpolicyAction

Selector: ASBOID.id

Selector: ASBOID.source

Escalation Action Web Services Examples

The following examples show the SOAP messages of many of the available escalation action web services.

Example: Create an email escalation action

The following example SOAP message is a Create request to create an escalation action that sends an email when the associated escalation policy criteria are met:

```
<env:Header>
    <wsa:ReplyTo xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core" env:mustUnderstand="true">
    uuid:4a909925-71c2-4317-82e4-b27c0b4f89d0
    </wsa:MessageID>
    <wsa:To xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core" env:mustUnderstand="true">
    http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI
    xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    http://ns.ca.com/2009/01/usm-data/EscalationpolicyAction
    </wsman:ResourceURI>
    <wsman:OperationTimeout
    xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    PT30.000S
    </wsman:OperationTimeout>
    <wsa:Action xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core" env:mustUnderstand="true">
    http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
```

```
</wsa:Action>
</env:Header>
<env:Body>
    <ns11:EscalationpolicyAction
    xmlns:ns10="http://www.w3.org/2003/05/soap-envelope"
    xmlns:ns11="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns12="http://ns.ca.com/2009/07/usm-core"
    xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:ns3="http://schemas.xmlsoap.org/ws/2004/08/eventing"
    xmlns:ns4="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
    xmlns:ns5="http://schemas.xmlsoap.org/ws/2004/09/transfer"
    xmlns:ns6="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
    xmlns:ns7="http://schemas.xmlsoap.org/ws/2004/09/mex"
    xmlns:ns8="http://schemas.wiseman.dev.java.net/metadata/messagetypes"
    xmlns:ns9="http://schemas.sam.ca.com/webservice/1/alarm.xsd">
      <sam_action_type>0</sam_action_type>
      <sam_action_data>emailaddress@ca.com</sam_action_data>
      <sam_action_subject>new web service subject</sam_action_subject>
      <sam_action_msg>the message of the email</sam_action_msg>
      <ns11:item_name>MyEmailAction</ns11:item_name>
      <ns11:item_description>Escalation Action created via WS
      </ns11:item_description>
    </ns11:EscalationpolicyAction>
</env:Body>
</env:Envelope>
```

The bold syntax defines the following properties for the escalation action:

- A type of 0 indicates an email escalation action
- The action sends an email to the address emailaddress@ca.com with a subject of 'new web service subject'.

The SOAP response to this request is as follows:

```
<env:Header>
    <wsa:Action env:mustUnderstand="true"
    xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
    </wsa:Action>
    <wsa:MessageID env:mustUnderstand="true"
    xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    uuid:ec2a8c6a-4cd1-4b89-8380-05ffc9dbd488
    </wsa:MessageID>
```

```
    <wsa:RelatesTo xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    uuid:4a909925-71c2-4317-82e4-b27c0b4f89d0
    </wsa:RelatesTo>
    <wsa:To env:mustUnderstand="true" x
    mlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
</env:Header>
<env:Body>
    <wxf:ResourceCreated
    xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    <wsa:Address
    env:mustUnderstand="true">http://localhost:8888/sam/webservice/</wsa:Address>
    <wsa:ReferenceParameters>
    <wsman:ResourceURI>
    http://ns.ca.com/2009/01/usm-data/EscalationpolicyAction
    </wsman:ResourceURI>
       <wsman:SelectorSet>
          <wsman:Selector Name="ASBOID.id">18</wsman:Selector>
          <wsman:Selector Name="ASBOID.source">4503599627370496</wsman:Selector>
       </wsman:SelectorSet>
    </wsa:ReferenceParameters>
    </wxf:ResourceCreated>
</env:Body>
</env:Envelope>
```

The bold syntax shows the returned selector properties for the created escalation action.

Example: Delete an escalation action

The following example SOAP message is a Delete request to delete an escalation action:

```
<env:Header>
    <wsa:ReplyTo xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
    <wsa:Address env:mustUnderstand="true">
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
    </wsa:ReplyTo>
```

```
    <wsa:MessageID xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
    uuid:4f440ab3-9c4b-46db-824a-426faf11e9bd
    </wsa:MessageID>
    <wsa:To xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
    http://localhost:7090/sam/webservice
    </wsa:To>
    <wsman:ResourceURI xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
    http://ns.ca.com/2009/01/usm-data/EscalationpolicyAction
    </wsman:ResourceURI>
    <wsman:OperationTimeout xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
    PT30.000S
    </wsman:OperationTimeout>
    <wsman:SelectorSet xmlns:ns13="http://ns.ca.com/2009/01/usm-data">
        <wsman:Selector Name="ASBOID.id">18</wsman:Selector>
        <wsman:Selector Name="ASBOID.source">0</wsman:Selector>
    </wsman:SelectorSet>
    <wsa:Action xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    env:mustUnderstand="true">
    http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
    </wsa:Action>
</env:Header>
<env:Body/>
</env:Envelope>
```

The bold syntax shows the selectors that uniquely identify the escalation action. The SOAP response to this request is as follows:

```
<env:Header>
    <wsa:Action env:mustUnderstand="true"
    xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse
    </wsa:Action>
    <wsa:MessageID env:mustUnderstand="true"
    xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    uuid:5179bae4-8d0e-42a5-a944-b7f6c7c29eec
    </wsa:MessageID>
    <wsa:RelatesTo xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    uuid:4f440ab3-9c4b-46db-824a-426faf11e9bd
    </wsa:RelatesTo>
```

```
    <wsa:To env:mustUnderstand="true"
    xmlns:ns11="http://schemas.sam.ca.com/webservice/1/alarm.xsd"
    xmlns:ns13="http://ns.ca.com/2009/01/usm-data"
    xmlns:ns14="http://ns.ca.com/2009/07/usm-core">
    http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:To>
</env:Header>
<env:Body/>
</env:Envelope>
```