CA Service Operations Insight

Connector 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 (CA 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

The *Connector Guide* details information about the connectors provided by CA Service Operations Insight (CA SOI) and the interaction of other connectors interfacing to CA SOI.

Audience

The following roles comprise the primary audience for this guide:

CA SOI administrators

Install, configure, and maintain the CA SOI solution. These administrators work closely with the operations team and the service owners to define and build service models.

Domain manager administrators

Install, configure, and maintain domain-specific connectors and understand how connectors can assist them in development and integration.

Integration developers

Deliver new integrations with CA SOI.

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.

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.

Web Services Reference Guide

Provides information about the CA SOI web services for interacting with resources such as CIs, services, alerts, relationships, and escalation policy.

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.

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: Introduction to Connectors

CA SOI uses connectors and the CA Catalyst infrastructure to connect to different domain managers. This section defines connectors and provides installation and configuration information for connectors.

This section contains the following topics:

Connectors Overview

A connector is software that provides the interface for data exchange between the CA Catalyst infrastructure and a domain manager. Connectors are the gateway through which data is retrieved from various domain managers for consolidated management. Each integrated product has its own connector that supports one or both of the following operation types:

Outbound from connector

Outbound from connector operations collect data (such as services, CIs, topology, alerts, and status) from the source domain manager. Data retrieved by connectors flows through the integration framework (IFW) and ActiveMQ server to the manager components. The manager components facilitate the reconciliation and display of the data on the product interfaces. All provided connectors support outbound operations.

Inbound to connector

Inbound to connector operations use records in the CA Catalyst Persistent Store to create, update, or delete items in the source domain manager. Inbound operations enable domain manager synchronization with changes spurred by CI reconciliation, CI creation, and CI updates in other domain managers. This synchronization helps reflect these changes in all domain managers. Many provided connectors support inbound operations.

You can configure, start, and stop connectors from the Administration UI.

Note: Each CA Catalyst Connector Guide contains information about connector installation, configuration, how the connector interprets data from its domain manager, and whether inbound operations are supported.

Bidirectional Connector

A *bidirectional connector* supports both inbound and outbound operations. Outbound-only connectors contain one connector policy file that transforms the gathered data to the standard USM format. Bidirectional connectors contain two connector policy files that transform outbound data to the USM format and transform inbound data to the source format of the domain manager.

Custom Integrations

CA SOI also provides the following tools for defining custom integrations:

Universal connector **(see page 67)**

Provides a web services interface that products can use to publish new services, CIs, and events, which are normalized to a common format and made available to the SA Manager. The Universal connector can retrieve services, CIs, and status events from various CA Technologies and third-party products.

Connector SDK **(see page 125)**

Provides the ability to develop custom connectors. The SDK includes a Sample connector, which provides the framework for writing a connector to integrate with important applications in your enterprise.

Event connector **(see page 97)**

Collects events from low-level event sources, transforms them into the CA SOI alert format, and displays them as infrastructure alerts in CA SOI associated with existing or created CIs.

Connector Documentation

This guide contains the following information:

- General information that applies to all connectors, such as architecture, basic configuration, and troubleshooting
- Detailed information about connectors provided on the CA SOI image, such as the Event and Sample connectors
- Advanced information about how to build custom connector integrations (see page 16)

All CA Catalyst connectors, which are provided separately from the CA SOI image as published solutions, contain a Connector Guide and Readme specific to that connector. This documentation contains the following information:

- Connector overview
- Prerequisites
- Detailed installation and configuration procedures
- Details about the data imported from the connector and other connector capabilities

Access connector-specific documentation from the Documentation directory of the connector package or from the SOI_HOME\Documentation directory after you install the connector.

Connector Installation

This section contains information about installing connectors. Connectors are provided in one of the following ways:

- Directly on the CA SOI image
- Separately downloadable as published solutions on CA Support Online

Note: CA Catalyst r3.2 connectors use the CA Catalyst r3.2 framework, which is not embedded in CA SOI. For information about CA Catalyst r3.2 connectors, see How to Install the CA Catalyst r3.2 Connectors (see page 24).

Connector Installation Considerations

Consider the following before you install connectors in your CA SOI environment:

- Verify that the SA Manager is already installed before you install connectors. The following SA Manager information is required during connector installation:
  - Host name
  - ActiveMQ Server port number (61616 by default)
  - Administrator user credentials
- Install all connectors from a local image or DVD or from a network share. Installers for the connectors do not support the use of UNC shares.
- Multiple connector installations are supported on one system. If you plan to install connectors on a system with other CA SOI components or multiple connectors on the same system, see the hardware requirements in the *Release Notes* to verify that the system has the appropriate resources to support the installations.

- As a best practice to avoid memory issues, connectors on one system must manage no more than 200,000 CIs combined. For example, if you have four CA eHealth installations that manage a total of 300,000 CIs, you must distribute the four remote CA eHealth connectors across at least two systems. Also consider the number of events that each connector will manage if your overall deployment represents a significant event stream. In this case, try to distribute the event stream evenly across remote connector systems and (when possible) group connectors that require the same type of processing.
- Some connectors require installation directly on their domain manager and others support a remote installation. The installer blocks installation of any connector that requires a local installation when the domain manager is not present on the system. For more information about whether a connector supports remote installation and other prerequisites and considerations, see the specific *Connector Guide* for that connector.

Note: For a list of provided product connectors and the domain manager versions they support, see the CA SOI product page on CA Support Online.

Install Universal, Event, Sample, and Mid-tier Connectors

CA SOI provides the following connectors either on the installation image or as related downloads:

- Universal connector (see page 67) (integrated into the main CA SOI installer)
- Event connector (see page 97) (available on a separate disk with CA Event Integration)
- Sample connector (see page 125)
- Mid-tier connector (see page 48) (installed when you install the SA Manager)
- CA SOI Domain connector

Follow the provided links for information about installing and working with these connectors. For information about working with the Mid-Tier connector, see the *Event and Alert Management Best Practices Guide*.

Consider the following when installing the Sample and Domain connectors provided on the CA SOI image:

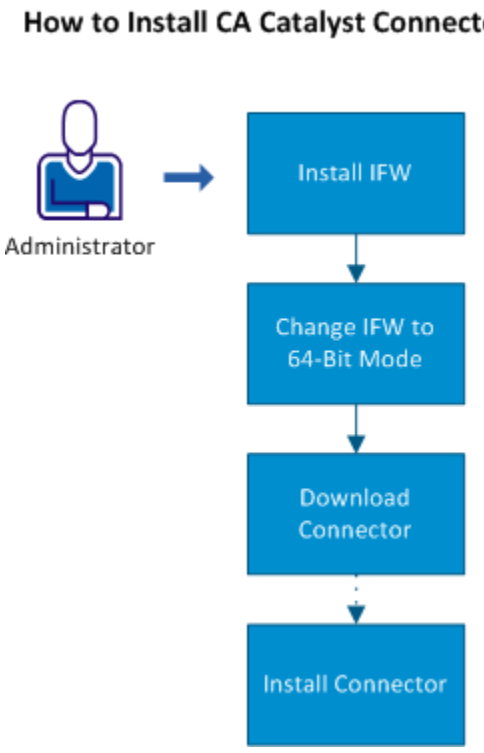
- If you are copying the installation program from the installation image to another system, verify that the merge modules file (*component*.iam.zip file) is in the same folder as the installation executable.
- For specific installation requirements, see the section for each connector in this guide. For instructions and requirements for the Domain connector, see the *Implementation Guide*.

How to Install CA Catalyst Connectors (Pre-r3.2)

CA Catalyst connectors prior to CA Catalyst r3.2 integrate with a specific domain manager or generic data source (such as SNMP traps) and are provided as downloadable published solutions.

For the most recent list of available connectors, see the connector download page on CA Support.

Use this scenario to guide you through the process:



Complete the following process to install CA Catalyst connectors:

1. Install the IFW (see page 20) from the CA SOI image on the connector system.

   Note: If you are installing connectors with the SA Manager, which already has the latest IFW installed, skip to Step 2.

2. (Optional) If your connector supports 64-bit, change the IFW to 64-bit mode (see page 21).
3. Download the connector and prepare for installation (see page 22).
4. Install the connector (see page 23).

Install the IFW

Install the latest version of the IFW on any system where you want to install CA Catalyst connectors. This procedure is not required if you are installing connectors with the SA Manager, which already has the latest IFW installed.

Consider the following items:

- Connectors that are provided on the CA SOI installation image do not require a separate IFW installation.
- CA Catalyst r3.2 connectors require an IFW Proxy installation instead of an IFW installation. For more information about CA Catalyst r3.2 connectors and the IFW Proxy, see How to Install CA Catalyst r3.2 Connectors (see page 24).
- Because some connectors do not work with a 64-bit IFW, the IFW does not install as a 64-bit application by default. However, you can change the IFW to operate as 64-bit (see page 21).

Follow these steps:

1. Run IntegrationServices.exe from the Disk1\SOI folder of the CA SOI installation image, and click CA Service Operations Insight - Integration Services.
2. Click Next.
3. Accept the license agreement, then accept the third party license agreements, and click Next.

   The Choose Install Folder page opens.
4. Accept, enter, or choose the installation folder.

   Note: The maximum installation path length is 150 characters. The installation blocks paths with more than 150 characters.
5. Complete the fields to connect to the SA Manager and configure connector preferences, and click Next. Refer to your Installation Worksheet in the SA Manager section for these values.
6. Specify whether to start the product services after installation, and click Next.

   The Pre-Installation Summary page opens.
7. Click Install.

   The IFW installs on the system and integrates with the specified SA Manager. An installation summary page opens when the installation finishes.
8. (Optional) Review the installation log file (IntegrationServices_Install_*releasenumber*) that is in the SOI_HOME\log folder to check for installation errors. This folder also provides the installation log files for the other installed components.

Change the IFW to 64-Bit Mode

Because some connectors do not work with a 64-bit IFW, the IFW installs as a 32-bit application by default. However, you can manually change the IFW to run on 64-bit systems. For more information about your connector, refer to the guide that is provided with your connector.

To avoid performance issues, if the connector server you monitor receives more than 100,000 CIs daily, we recommend enabling the 64-bit IFW.

Follow these steps:

1. Stop the CA SAM Integration Services service on the server where the IFW is installed.
2. Navigate to the SOI_HOME\jsw\conf folder.
3. Open the SAM-IntegrationServices.conf file and find the following lines:

   ```
   # Java Application
   set.JAVA_HOME=C:\Program Files (x86)\CA\SOI\jre-32
   ```

4. Verify that JAVA_HOME is pointing to the correct Java library location and update if necessary.
5. Locate the following line:

   ```
   # #include ../conf/IFW-wrapper-jvm-64.conf
   ```

6. Uncomment the line by removing the first pound sign and the space before the second pound sign:

   ```
   #include ../conf/IFW-wrapper-jvm-64.conf
   ```

7. Restart the CA SAM Integration Services service.
8. Repeat Steps 1-7 on other servers where you want to change the IFW to 64-bit mode.

Download Connectors and Prepare for Installation

The base CA SOI image does not provide CA Catalyst connectors. Download CA Catalyst connectors from the CA SOI product page on CA Support Online and install them separately. For the connector installation to work, you must extract downloaded connector files so that an exact folder structure is created.

Follow these steps:

1. Access CA Support Online and log in.
2. Click the Support by Product link and select CA Service Operations Insight from the Select a Product page drop-down list.
3. Locate the Recommended Reading section and click CA Service Operations Insight - Connectors.
4. Download the patch for the CA Catalyst connector that you want to install, and copy it to the system where you have the CA SOI installation image.
5. Extract Connector_*ProductName*.zip to the Disk1\SOI folder of the installation image.
6. Create the Merge_Modules folder in the Connector installer and copy IntegrationServices.iam.zip from the installation image in the Disk1\SOI folder.

   The connector installation fails unless the following folder structure exists in the folder where you extracted the zip files:

   - SOI
     - Merge_Modules (contains IntegrationServices.iam.zip)
     - Connector_*ProductName*.exe
   - Documentation
     - PDFs
     - Readme

   When you extract the zip files from the same location, the correct folder structure is automatically created.

   Multiple Connector_*ProductName*.exe files can use the same Merge_Modules folder, if you want to install multiple connectors on the same system.
7. Check the *ProductName Connector Guide* and Connector Readme provided with the connector package (in the Documentation folder) before you start the installation.

   Many connectors have required preinstallation steps.

Install CA Catalyst Connectors

Install CA Catalyst connectors to establish integrations with specific domain managers and data sources.

Note: CA Catalyst connectors reference the old product name CA Spectrum Service Assurance and the previous version number r2.5 in several places, such as the installer, connector-specific documentation, and the installed Start menu shortcut. However, these connectors do work with the current release of CA SOI.

Follow these steps:

1. Double-click the Connector_*ProductName*.exe file located in the SAM folder at the location where you extracted the connector.

   Note: The zip files must be in the Merge_Modules folder.

   The Introduction page of the connector installation wizard opens.

2. Click Next.

   The License Agreement page opens.

3. Scroll to the bottom of the agreement, select "I accept the terms of the License Agreement" and click Next.

   The *Product Name* Connector Configuration page opens.

4. Enter the required information for connecting to the domain manager, and click Next:

   Note: The *ProductName Connector Guide* provided in the Documentation folder of the connector package contains descriptions for all required domain manager information.

   The Service Startup page opens.

5. Specify whether to start the product services automatically after installation, and click Next.

   The Pre-Installation Summary page opens.

6. Review your selections, and click Install.

   The connector installs on the system and integrates with the appropriate domain manager and CA SOI instances. The Install Complete page opens when the installation finishes.

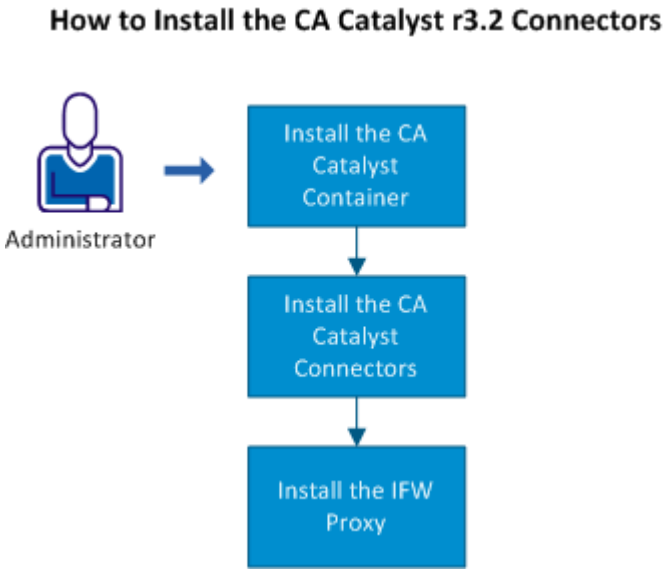
Most connectors install a log file for troubleshooting installation errors. For more information, see the appropriate *ProductName Connector Guide*. To verify that the connector installed and initialized correctly, view the connector status in the Administration UI.

How to Install CA Catalyst r3.2 Connectors

As an administrator, you install the CA Catalyst r3.2 Container and the IFW Proxy to facilitate connector communication with the IFW.

CA Catalyst r3.2 connectors support the CA Catalyst r3.2 infrastructure that is not embedded in the SA Manager. Once the communication is established between the IFW and the CA Catalyst r3.2 connectors, the connectors operate the same as traditional CA Catalyst connectors.

Use this scenario to guide you through the process:



1. Install the CA Catalyst r3.2 Container (see page 25).
2. Install the CA Catalyst r3.2 connectors (see page 27).
3. Install the IFW Proxy (see page 27).

Download the IFW Proxy Patch from CA Technologies Support

This patch provides the installations for the CA Catalyst r3.2 Container and the IFW Proxy.

Follow these steps:

1. Log in to http://support.ca.com.
2. Access the CA SOI product page, and click CA Service Operations Insight Solutions and Patches in the Recommended Reading section.
3. Download the latest IFW Proxy patch.

   For current information about IFW Proxy updates, see the *Release Notes* on the Bookshelf.
4. Follow the download instructions on the Support page.

Install the CA Catalyst Container

Install a CA Catalyst Container (r3.2 Build 535 or higher) on the systems where you plan to install CA Catalyst r3.2 connectors. You can install multiple CA Catalyst r3.2 connectors on the same system into the same CA Catalyst Container.

For an IFW Proxy implementation, we recommend a Container-only CA Catalyst deployment. This procedure reflects the recommended selections to make in a Container-only environment. The CA Catalyst Server is not required.

Follow these steps:

1. Run catalyst_install.bat from the IFW Proxy Patch (see page 25).

   The CA Catalyst Container installation wizard opens.

   Note: For all installation pages not mentioned in this procedure, leave the default selections unless the default port numbers are in use. For the simplest deployment experience, complete the Installation Worksheet for the CA Catalyst Container for CA SOI with the values that you enter during installation.
2. Proceed through the wizard, and perform the following actions to prepare for the CA SOI integration:

   New Catalyst Container

   Leave the default selection of Default Connector Container (CatalystConnector) unless you are installing multiple Containers on the same system.

   Choose Install Folder

   Retain the default installation location.

Configure High Availability

(Optional) Select Configure for High Availability and enter the necessary information if you are deploying the CA Catalyst Container in a cluster environment.

CA EEM Server Configuration

Enter information about the CA EEM server to use for CA Catalyst security. This can be the same CA EEM server that you use for CA SOI. Use the default application name for CA Catalyst; all other values can be the same.

Administrator Configuration

Enter CA Catalyst administrator credentials.

Registry Location

Select File Registry for a Container-only implementation.

Catalyst Connector Container Configuration

Accept the default values unless the ports are already in use.

HTTP Port

Retain the default port. If this is the second Container on the same system, the defaults automatically change to avoid conflicts. If this is anything past the second Container on the same system, you must manually change to unused ports.

Default: 8080, or 8180 on a multiple Container system

HTTP Service Port

Retain the default port. If this is the second Container on the same system, the defaults automatically change to avoid conflicts. If this is anything past the second Container on the same system, you must manually change to unused ports.

Default: 7000, or 7100 on a multiple Container system

HTTPS Service Port

Retain the default port. If this is the second Container on the same system, the defaults automatically change to avoid conflicts. If this is anything past the second Container on the same system, manually change to unused ports.

Default: 7443, or 7543 on a multiple Container system

Catalyst Services Startup

Verify that the check box is selected to start the services.

Specify 64 Bit Support

If you are installing on a 64-bit system, select Use 64 Bit JRE to install as a 64-bit application.

3. Click Install on the Install Summary page.

   The CA Catalyst Container installs. The Install Complete page displays the results of the installation. To troubleshoot installation errors, see the CATALYST_HOME*containerName*\logs\CA_Container_InstallLog.log file, or see the *CA Catalyst Installation Guide*.

4. Repeat this process on all systems where CA Catalyst Container installation is required for hosting CA Catalyst r3.2 connectors.

Install CA Catalyst r3.2 Connectors

Install CA Catalyst r3.2 connectors on the same system where you installed the CA Catalyst r3.2 Container. CA Catalyst r3.2 connectors are available as patches on CA Support.

Consider the following items before installing CA Catalyst r3.2 connectors:

- Each CA Catalyst connector comes with a product-specific *Connector Guide* and *Readme*. Reference these documents for information about the connector functionality, installation, and configuration.
- You can install multiple connectors into the same CA Catalyst Container. Alternatively, you can install connectors on the same system into multiple CA Catalyst Containers with different Container IDs. Using dedicated CA Catalyst Containers on the same system lets each connector run under its own process.
- Refer to your Installation Worksheet in the CA Catalyst r3.2 Container section when the connector installer asks you for CA Catalyst information.

Install the IFW Proxy

The IFW Proxy enables CA SOI to visualize and manage data that the CA Catalyst r3.2 connectors provide. Install the IFW Proxy on each CA Catalyst Container system that hosts connectors that you want to manage in CA SOI. A CA Catalyst Container is required on the system on which you install the IFW Proxy. If you have multiple CA Catalyst Containers hosting connectors on a single system, install a separate IFW Proxy to communicate with each Container.

We recommend that all connectors are installed on the CA Catalyst Container system before you install the IFW Proxy. However, you can install a connector on a Container system after installing the IFW Proxy.

Follow these steps:

1. Run soi.ifwproxy.exe from the IFW Proxy Patch (see page 25) on the CA Catalyst Container system that hosts connectors that you want to manage in CA SOI.

   Note: If multiple CA Catalyst Containers are available on the system, the IFW Proxy installer prompts you to select the appropriate Container.

   The IFW Proxy installer opens to the Introduction page.

2. Accept both license agreements and click Next.

3. Enter SA Manager information as follows, and click Next.

   Note: Refer to your Installation Worksheet in the SA Manager section for these values.

   SA Admin

   Defines the administrator user name that is created during the CA SOI installation.

   Default: samuser

   Password

   Defines the password for the administrator user.

   Manager host

   Defines the host name where the SA Manager is installed.

   ActiveMQ port

   Defines the ActiveMQ Server port through which connector communication occurs.

   Default: 61616

   UCF broker port

   Defines a valid port for the UCF Broker to send information that is created or updated in CA SOI to the CA Catalyst r3.2 connector.

   Default: 8020

   Use DNS for name resolution

   The IFW Proxy does not use this value.

   Connector name

   Defines the name that is displayed for all connectors from this source. If you are implementing a high availability solution, change this value to the virtual host name.

4. Click Install.

   The IFW Proxy installs, and the Install Complete page opens. If errors display, check the CA_IFWProxy_InstallLog.log file installed to CATALYST_HOME*containerName*\logs for details.

   Note: CATALYST_HOME denotes the root CA Catalyst installation directory, which is C:\Program Files\CA\Catalyst by default.

5. (Optional) Repeat Steps 1-4 for all local or remote CA Catalyst Containers that host connectors that you want to manage in CA SOI.

Interacting with CA Catalyst r3.2 Connectors

This section lists methods of interaction with CA Catalyst r3.2 connectors that are different than those for traditional connectors.

Configure Connector Administration Settings

By default, you can start or stop a CA Catalyst r3.2 connector from the CA SOI Administration UI using the Start and Stop buttons. You can configure whether these buttons actually start or stop the connector or simply stop the flow of connector data into CA SOI.

Follow these steps:

1. Access the ifw.properties file located at CATALYST_HOME*containerName*\registry\topology\physical*connectorserver*\ifw on the CA Catalyst connector system.

   Note: If your CA Catalyst deployment includes a CA Catalyst Server, log in to the Registry Administration UI to access and modify this file.

2. Set the connectorAdmin value to one of the following, and save and close the file:

   true (default value)

   Lets you actually start and stop the connectors in the CA SOI Administration UI. The Start and Stop buttons in the Administration UI start or stop the connector in CA Catalyst.

   false

   Lets you start or stop the data flow from connectors in the CA SOI Administration UI. The Start and Stop buttons do not actually control the connector status in CA Catalyst, only whether connector data makes it into CA SOI. If you set this value, you can start and stop connectors only from the CA Catalyst Registry files.

3. Restart the CA Catalyst Container *ContainerName* service on the connector system if you made a change to the file.

Change Encrypted Connector Property Values from Dashboard Administration Tab

The CA SOI Administration tab supports modifying CA Catalyst connector properties. However, if you change the value of an encrypted property, use CA Catalyst encryption so that CA Catalyst can process the new encrypted value.

Follow these steps:

1. Open a command prompt on a system with CA Catalyst r3.2 components installed.
2. Navigate to CATALYST_HOME*containerName*\tools\encrypt and run the following command:

   `encrypter` *`password`*

   password

   Defines the new value of the encrypted property.

   An encrypted string appears.
3. Copy the encrypted string, and paste it into the property field on the CA SOI Administration UI.
4. Save the changes, and restart the connector.

   The encrypted value change takes effect.

Change the CA SOI Password on a CA Catalyst Connector System

If you change the CA SOI administrator password, you also change the password in the IFW Proxy on the CA Catalyst Container. Keep this password synchronized in CA SOI and the IFW Proxy to avoid connection issues.

Follow these steps:

1. Open a command prompt on the connector system, navigate to CATALYST_HOME*containerName*\ifw, and run the following command:

   `EncryptSAMCreds` *`newpassword`*

   newpassword

   Defines the new password for the administrator user.

   The command generates an encrypted password.

2. Copy the password.

3. Open the CATALYST_HOME*containerName*\ifw\resources\configurations\SSA_IFW_*servername*.xml file, paste the new encrypted password into the password property, and save and close the file.

4. Open the CATALYST_HOME*containerName*\registry\topology\physical*connectorserver*\ifw\eventManagementServer.properties file, paste the new encrypted password into the password property, and save and close the file.

   Note: If your CA Catalyst deployment includes a CA Catalyst Server, log in to the Registry Administration UI to access and modify this file.

5. Restart the CA Catalyst Container *ContainerName* service.

   The password change takes effect.

Remove the Connector and Connector Data from CA SOI

Removing CA Catalyst r3.2 connectors and their data from CA SOI does not require uninstalling the connector. Uninstalling the IFW Proxy deletes the connection between CA SOI and CA Catalyst, and you can remove existing connector data from CA SOI.

Follow these steps:

1. Uninstall the IFW Proxy on the CA Catalyst connector system by selecting Uninstall CA Service Operations Insight IFW Proxy from the Start menu.

   Uninstalling the IFW Proxy removes the connection between CA SOI and the CA Catalyst connector.

2. Find the connector entry in the CA SOI Administration UI and click Remove Connector.

   The connector data is deleted from the SA Store database, and the entry disappears from the tree when the operation is complete.

3. (Optional) Uninstall the connector if it is not required for other CA Catalyst solutions.

If you need to maintain a connection between CA SOI and other connectors on the same system, retain the IFW Proxy. Clicking Remove Connector on the CA SOI Administration UI still removes the CA Catalyst r3.2 connector data and updates the connector configuration to prevent subsequent use of the connector in CA SOI.

To enable a disabled connector, set the SOIState property to Enabled or remove the property altogether in the *connectorname*.conf file located at CATALYST_HOME*containerName*\registry\topology\physical*nodename*\modules\configuration.

Note: If your CA Catalyst deployment includes a CA Catalyst Server, log in to the Registry Administration UI to access and modify this file.

High Availability Connector Considerations

High availability (also known as fault tolerance or failover) is a common architectural requirement based on Microsoft Cluster Server (MSCS) that focuses on ensuring business continuity in the event of an interruption of IT resource availability. The main objective of implementing a high availability solution is zero downtime for IT resources.

You can install connectors in a high availability environment. High availability support provides failover capabilities and a solution for applying maintenance while avoiding service management downtime. For more information about installing connectors to work in a high availability CA SOI environment, see the *Implementation Guide*.

To install connectors to operate with a high availability domain manager, you must follow the process described in How to Implement CA SOI in a MSCS Environment in the *CA SOI Implementation Guide*, with the following differences:

- Install only the IFW and the connector on the domain manager nodes.
- Only connector-specific CA SOI resources are added to the cluster group after you run the resource kit on each node. No data files are moved to the shared disk when running the resource kit on a connector-only system.

Note: This procedure does not apply for installing the CA Spectrum connector when CA Spectrum is installed with Primary and Secondary SpectroSERVERSs (Distributed SpectroSERVER environment).

Verify the status of all connectors and remove any connectors reporting to the real node.

Upgrade CA Catalyst Connectors

CA Catalyst connectors do not require an upgrade to work with the current release of CA SOI. However, upgrade the IFW on the system to ensure that they can interact with the latest version of the product. Upgrading the IFW is not required in the following situations:

- The connectors exist on the SA Manager. Upgrading the SA Manager to the current release of CA SOI already upgraded the IFW, and the connectors come online and work immediately after the upgrade.
- Connectors provided with the CA SOI image may require a full upgrade.

To upgrade CA Catalyst connectors, install the IFW (see page 20) on the connector system.

The IFW upgrades, and the connectors are configured to work with the current release of CA SOI.

Consider the following information:

- References to the old product name and version number are not updated in the connector documentation and Start menu shortcuts.
- If you are using CA Service Desk with the auto clear alerts option enabled, then shut down a connector, CA SOI automatically changes the cleared alerts to the selected status. You cannot undo this operation. For more information, see the *Implementation Guide*.
- If the CA Catalyst container with the IFW proxy installed is upgraded from CA Catalyst 3.2 to 3.3 or 3.4, then the IFW proxy must also be upgraded to CA SOI 3.2.

Uninstall Connectors

Uninstall connectors to remove the connector software. After uninstallation, you must perform the operations described in Connector Removal to fully remove references to the uninstalled connector in the product.

Follow these steps:

1. Select one of the following from the Start menu on the connector system:
   - Connectors provided with the CA SOI image: Start, Programs, CA, Service Operations Insight, Uninstall *Connector Name*.
   - CA Catalyst connectors: Start, Programs, CA, Service Assurance Manager, Uninstall *Connector Name*.

   The Uninstall *Connector Name* dialog opens.
2. Click Uninstall.

   The connector uninstalls. If the connector is the only one installed on the system (and no other CA SOI components are also installed), the IFW is uninstalled. If other connectors exist on the system, the IFW remains.

Connector Configuration

The Connector Configuration option in the Administration tab on the Dashboard lets you view (see page 36), edit (see page 38), and configure (see page 41) and remove (see page 42) the connectors that are available to CA SOI.

View Connector Status

As an administrator, you can view the Connector Status table, which lists all the connectors available to CA SOI. From this table, you can monitor the IFW and connector status.

Follow these steps:

1. Click the Administration tab, and select Connector Configuration.

   The Connector Status table displays with the following columns:

   Connector Service

   Displays the name of the server where the IFW is installed. The IFW exists on every system that has at least one connector installed, and it connects to the domain manager that CA SOI is monitoring.

   Status

   Displays one of the following states for the IFW on the connector server:

   Online/Offline

   Indicates that the IFW is running or not running.

   Initializing

   Indicates that the IFW is starting up.

   Handshake

   Indicates that the IFW was started and is waiting for CA SOI to contact it.

   Closing

   Indicates that the IFW is shutting down.

   Connector

   Displays the connector identifier using the following format:

   connector_mdrproductvalue@product_host. The connector runs on its installed system, communicates with the individual domain manager, collects data from it, and lets the IFW process this data. The connector identifier typically includes a unique ID number for each connector and the system on which the integrated domain manager is installed.

Status

Displays one of the following states:

Online\Offline

Indicates that the connector is running or is not running.

Initializing

Indicates that the connector is starting. This state is short-lived during which connectors initialize their connections to domain managers and build their caches before switching to the Online status.

Handshake

Indicates that the connector was started and is waiting for CA SOI to contact it.

Closing

Indicates that the connector is shutting down.

Source Description

Contains information about the source domain manager with which the connector integrates, specifically on which system the domain manager is running.

Connector Description

Contains the connector identifier and version number.

2. Click one of the server names in the Connector Service column.

The read-only details page opens and displays the following details about the IFW on each connector server:

Note: The tree view in the Administration Pages pane is expanded, and the selected server is highlighted under the Connector Configuration node. You can also navigate to this details page by clicking the tree nodes.

CA Service Operations Insight Integration Framework Status

Indicates the status of the IFW on the system.

Messaging Service Properties

Displays information regarding the ActiveMQ messaging service that controls the exchange of information between the connector, the IFW, and manager components.

Connector Status

Contains status information about the connectors that are installed on this system.

UCF Broker 1/UCF Binding 1

Contains information about UCF, or the common CA Catalyst connector framework.

3. Select Connector Configuration in the tree to return to the Connector Configuration page.
4. Select one of the connector identifiers in the Connector column.

   Note: The tree view in the Administration Pages pane is expanded and the selected server is highlighted under the Connector Service node. Also, you can navigate to this details page by clicking the tree nodes.

   For more information about editing the connector, see Edit Connector Configuration (see page 38).

Edit Connector Configuration

As an administrator, you can change the configuration settings, navigate to the Connector Configuration page on the Administration tab in the CA SOI Dashboard. Connectors are initially configured when you install them.

Follow these steps:

1. Click the Administration tab and the Connector Configuration option.

   The Connector Configuration page opens and displays the Connector Status table.
2. Click one of the connector names in the Connector column.

   The connector details page opens and displays a number of tables with additional details about the connector and the server where it is installed. Many connectors also have a read-only Connector Type Data table that displays information such as the supported CI types.

   Note: If your connector supports Scheduler Configuration and has the scheduler property enabled, then the button is available. For more information about Scheduler Configuration availability, see the connector documentation provided with your connector.

   Important! The CA:00056_service-discovery@*servername* entry on the SA Manager server represents the Service Discovery component. This entry always has blank Connection Details. Do not edit any of the connector controls for this entry.

3. (Optional) Edit the following settings in the Connector Controls column as necessary, and click Save:

dns_resolution

Specifies whether to use DNS resolution to resolve device names. If a reliable DNS mechanism is not in place (for example, no DNS server on the network, or configuration items (CIs) not defined to the DNS), disable DNS lookups to prevent CI resolution and normalization failure.

Default: on

useAlertFilter

Specifies whether to filter alerts based on their existence in a managed service in CA SOI. If the control is turned on, the connector only sends domain manager alerts that are associated with a CI that is part of an existing managed service in CA SOI. If the control is turn*ed* off, the connector forwards all alerts from the domain manager, regardless of whether they relate to a service.

Default: on

Note: For CA SOI r3.1 and later, the IFW no longer requires or uses EVENT_FILTER requests. The IFW now sends the alerts (managed or unmanaged) that the connectors report to the SA Manager. CA SOI ignores the global setting sendAllAlerts (in the IFW configuration file) and the connector-specific setting useAlertFilter (on the Administration tab Connector Configuration page). CA SOI now behaves as if sendAllAlerts=1 and useAlertFilter=0, regardless of the actual settings.

getCIsAtStartUp

Specifies whether to rediscover CIs every time the connector starts. This control is enabled by default so that connectors always provide a current record of all CIs from their domain managers. You can turn this control off if the connector does not support collecting CIs at startup, such as the Universal or SNMP connector.

Default: on

isRemotable

Specifies whether to allow the connector framework to access the connector remotely for create, update, and delete operations on the source domain manager.

Default: on

useServiceFilter

Specifies whether to send all relationships to CA SOI or only the ones associated with modeled services. Set the control to true to run relationships through a service filter and receive only the relationships associated with modeled services.

Default: on

getRelationshipsAtStartup

Specifies whether to rediscover relationships every time the connector starts. This control is turned off by default so that relationships are only obtained and imported as a part of service model imports. You should only enable this control if you require relationship CIs outside of imported service models, for example, if you define an Unmanaged Relationship Service Discovery rule.

Default: off

performDeltaProcessing

Specifies whether to process and publish deltas on CIs between the time the connector or SA Manager was last stopped or restarted. When enabled, this setting also performs delta processing on relationships if the getRelationshipsAtStartup property is enabled.

Default: on

4. Select any field in the Connection Details, Connector Instance Data, and Launch in Context Details tables, edit the configuration setting, and click Save in the same table.

   The changes are saved. Verify that any changes you make the connection settings do not break the connection. For example, if you modify a Host field in the Connection Details, first confirm that the domain manager is installed and configured on the new host.

   Note: For information about the connector parameters for each connector, see the *CA Catalyst Connector Guide* provided with the specific connector package.

5. Click Stop and wait until the connector status changes to Offline.
6. Click Start and wait until the connector status changes to Online.

   The connector is restarted. Depending on the type of connector, it can take a few minutes before the connector Status displays Online.

Important! Do not perform rapid start and stop operations on the connector. Each stop and start sends the corresponding command to the connector. Rapid start and stop operations from the interface can cause these commands to queue on the connector and cause the connector to start and stop repeatedly until all commands in the queue are processed.

Configure the IFW Configuration File

As an administrator, you can change the connector-related settings that are saved in the IFW configuration file (SOI_HOME\resources\Configurations\SSA_IFW_HostName.xml). The settings are applicable to all connectors running under that IFW. You can configure the IFW configuration file to change these settings globally for all connectors on the system. For example, changing the alert filter setting at the IFW level automatically overrides the individual alert filter settings for all the connectors that are installed on that IFW system.

Follow these steps:

1. Open the SOI_HOME\resources\Configurations\SSA_IFW_HostName.xml file on a system with the IFW installed (any system with connectors or the SA Manager).
2. Change the value for the appropriate parameters in the ConnectorConfig section as follows, and save and close the file when finished:

   Important! Use caution when changing these settings, and change documented settings *only*.

   retryCount

   Defines how many times to retry connecting to the ActiveMQ Server component on the SA Manager when the connection fails. Before changing the default retry settings, consider that the amount of time a connector waits for the ActiveMQ Server connection reflects the amount of queued data that will consume memory in the IFW until the connection is reestablished.

   Default: 5

   retryInterval

   Defines the number of seconds between retrying to connect to the ActiveMQ Server.

   Default: 30

   dns_resolution

   Defines whether to use the DNS lookups for the CI name resolution. The default value of 1 turns on DNS lookups for all connectors on the system. When set to 1, you can also manage DNS lookup settings by connector if you want to use different settings for each connector. Change this value to 0 to disable the DNS lookups for all connectors. When you set this value to 0, you cannot manage DNS lookup settings by connector; DNS is always disabled for all connectors.

   Default: 1

   Note: For more information about managing DNS lookup settings by connector, see Edit Connector Configuration and the *Implementation Guide*.

sendAllAlerts

Defines whether to filter or send alerts that are not associated with a managed service. The default value of 1 sends all alerts from all connectors. Change the value to 0 to let you specify this behavior by connector.

Default: 1

Note: For CA SOI r3.1 and later, the IFW no longer requires or uses EVENT_FILTER requests. The IFW now sends the alerts (managed or unmanaged) that the connectors report to the SA Manager. CA SOI ignores the global setting sendAllAlerts (in the IFW configuration file) and the connector-specific setting useAlertFilter (on the Administration tab Connector Configuration page). CA SOI now behaves as if sendAllAlerts=1 and useAlertFilter=0, regardless of the actual settings.

throttleConnectorStartup

Defines whether to initialize connectors on the system simultaneously or one after another. The default setting of 0 initializes all connectors simultaneously. You can change this setting to 1 for connectors to initialize sequentially, which can improve the performance. The setting can also prevent memory overuse if there are multiple connectors on the system that manage large amounts of CIs.

Default: 0

3. Restart the CA SAM Integration Services service.

Connector Removal

You can either permanently or temporarily remove a connector from the CA SOI database and all interfaces:

- Permanently remove an uninstalled connector (see page 43).
- Temporarily disable an installed connector (see page 43).

Remove an Uninstalled Connector

After you uninstall a connector, remove the database references using the Administration tab. This also removes the connector database entry and the connector name from the list of connectors in the tree view.

If the connector comes back online after the operation is completed, the connector reregisters with CA SOI.

Follow these steps:

1. Select the Administration tab and the Connector Configuration option.
2. Select the name of an offline connector in the Connector column.

   Note: You cannot remove an online connector.
3. Click Remove Connector and confirm the deletion.

   The selected connector registration is permanently removed from the CA SOI database, and the connector name is removed from the tree on the Administration tab.

Disable an Installed Connector

You can disable an installed connector if you want to remove connector records temporarily, but keep the connector installed for potential future use.

Important! To ensure data integrity, perform this operation after business hours for connectors with a large amount of CIs and relationships. Removing the database information for these CIs and relationships locks the system for approximately ten minutes for every 100,000 CIs managed by the connector.

Note: If you are using CA Service Desk with the auto clear alerts option enabled, then shut down or remove a connector, CA SOI automatically changes the cleared alerts to the selected status. You cannot undo this operation. For more information, see the *Implementation Guide*.

Follow these steps:

1. Select the Administration tab and the Connector Configuration option.
2. Verify that the Connector Service status is Online for the connector to disable.

   The IFW for the connector must be running for the disable operation to work. If it is not running, start the CA SAM Integration Services service on the connector system.
3. Select the connector to disable.
4. Click Stop if the connector appears as Online.

   Note: You cannot disable an online connector.

5. Click Remove Connector after the connector appears as Offline and confirm the disable.

   The selected connector's registration is removed from the CA SOI database and the connector name is removed from the tree on the Administration tab. However, if the connector is still installed, all of its required files still exist in case you want to re-enable the connector.

Enable a Disabled Connector

You can enable a connector that you previously disabled to bring it back online and collect data from its domain manager again.

Follow these steps:

1. Open the connector configuration file that in the SOI_HOME\resources\Configurations folder on the connector system.

   The format of the connector configuration is typically the connector name followed by the connector system name (for example, sampleConnector_server1.xml).

2. Change the State property value from not Enabled to Enabled and save the file.

3. Restart the CA SAM Integration Services service on the connector system.

   The connector entries reappear in the Connector Configuration tree and in all other interfaces, and the connector begins collecting data from its domain manager.

Connector Removal Success and Failure Messaging

CA SOI provides an Administration tab confirmation message that indicates success or failure upon connector removal.

Note: This procedure assumes that you have already uninstalled the connector and are trying to remove its data from the CA SOI interfaces.

Follow these steps:

1. On the Dashboard, click the Administration tab.
2. Click the plus sign (+) next to the Connector Configuration.
3. Click the plus sign (+) next to a connector.
4. Click the connector.
5. If the connector is still online, click Stop.
6. Click Remove Connector.

   A confirmation appears at the top of the page that provides the connector removal success or failure.

Improved Connector Lifecycle Management

CA SOI provides an improved connector lifecycle management. The improved process enables you to reduce the time that you spend on diagnosing any connector-related connection issues. For example, when the domain manager of a connector goes offline, the IFW no longer stops trying to start the connector even after the completion of the configured retry count. The connector goes into an extended retry mode, if enabled. And, whenever the domain manager comes online, the connector establishes the connection and starts the processing, thereby reducing the downtime duration. Additionally, an email notification about the connection status is also sent to the administrator so that the administrator is aware of the issue and can take appropriate steps. This mechanism helps you proactively manage unexpected connection conditions and take correct and quick decisions.

Note: This feature is only available for connectors that are installed and run in CA SOI Integration Service. The lifecycle management for CA Catalyst r3.x connectors that run in CA Catalyst is outside the control of CA SOI, and is handled by CA Catalyst through connector configuration.

CA SOI has improved the connector reconnection process as follows:

- A new connector control *retryUntilMdrAvailable* is now available. This connector control specifies whether a connector should retry indefinitely to connect to its domain manager after the completion of the already configured connection retry count. You can configure this connector control to enable or disable the extended retries. To add this configuration, in the connector configuration file (SOI_HOME\resources\Configuration*ConnectorName*.xml), add the following property to the <ConnectorControls> element:

  ```
  retryUntilMdrAvailable="1"
  ```

  Note: For more information about how to configure a connector control, see the *Connector Guide*.

  When the configured retries to connect to the domain manager are exhausted, the retryUntilMdrAvailable connector control determines whether the connector should continue its retries. If the value of the control is set to 1 (enabled), the connector goes into an extended retry mode. In this mode, the connector tries to connect to its domain manager periodically until that connection is successful, or the user manually shuts down the connector.

  Note: The retry period for this mode is specified in the SOI_HOME\resources\Configurations\SSA_IFW_*HostName*.xml configuration file using the following property in the <ConnectorConfig> element:

  ```
  extendedWaitBeforeRetry="<seconds>"  - the default is 600 seconds
  ```

If the value of retryUntilMdrAvailable is set to 0 (disabled), the connector does not go into the extended retry mode. The connector does not try infinitely to connect to the domain manager after the configured retry count is exhausted.

If retryUntilMdrAvailable is not defined within the connector configuration, it will default to 1 (enabled). It is only necessary to add the control parameter to disable this feature.

- If the extended retry mode is enabled, the IFW sets the connector status as OFFLINE (RETRYING) after the IFW is unable to start the connector within the configured retry count. The IFW also sends a handshake message to the SA Manager. The SA Manager reads and stores the new status. The message includes detailed information that the configured retry count is exhausted, the IFW is unable to start the connector within the specified retry count, and it is marking it as OFFLINE (RETRYING). The SA Manager also sends an email notification that includes the handshake message information (received from the IFW) to the administrator.

  Note: For more information about how to configure email notifications, see the *Troubleshooting Guide* that is provided on the CA SOI online bookshelf.

- When the connector is in the extended retry mode, the status of the connector is displayed as OFFLINE (RETRYING) in the Administration UI and Connection dialog. In this state, the Stop button is available from the Administration UI. Users can use this button to stop the extended retries.

- Each retry logs a message in the Info dialog on the Console, indicating that the IFW is attempting to restart the connector. The message is also displayed in the Administration UI.

Chapter 3: Connector Infrastructure

This section explains the connector infrastructure, which includes the Integration Framework, ActiveMQ Server, UCF Broker, Event Management, and USM.

This section contains the following topics:

Integration Framework

The *integration framework (IFW)* is the mechanism that CA SOI uses to connect to domain managers and gather CI, service, topology, and state information. The IFW exists on any system with a connector or the SA Manager, and it interfaces with the connector framework to prepare connector data for transmission to the manager components. The IFW contains a transformation engine that uses connector policy to transform connector data to the USM format. The IFW also includes the infrastructure of the Event Management component, which provides the mechanism for storing events from connectors for exposure to event policy and eventual display as alerts after event processing completes.

The IFW uses the Apache ActiveMQ message broker, which fully implements the Java Message Service (JMS) as its protocol.

Apache Active MQ JMS Server

Apache ActiveMQ is an open source (Apache 2.0 licensed) message broker that implements the Java Message Service 1.1. The ActiveMQ Server controls all messaging and communication from external sources. The server also receives alerts and CI information from connectors and sends this information to various components for storage and analysis.

Event Management

Event Management provides a way to manage the event (USM type Alert) data from connectors more granularly and to control what makes into the Operations Console as alerts. It provides a processing layer between raw connector USM alert data and alert management. With Event Management, you can control the event stream so that a consolidated, high quality, and actionable set of alert conditions appear in the Operations Console as alerts.

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

Unified Service Model

The *Unified Service Model (USM)* is the schema that defines the internal format for all CA Catalyst data. Connectors transform all data collected from domain managers to the USM format before they send the data through CA Catalyst. The USM schema is stored in the CA Catalyst Registry.

Note: For more information, see the "Unified Service Model (USM)" chapter.

UCF Broker

The UCF Broker is a communication layer that controls access to the enabled bidirectional connectors, which can invoke inbound to connector operations on source domain managers. The Logic Server communicates synchronization changes to bidirectional connectors through the UCF Broker.

How Connectors Fit into CA SOI and CA Catalyst

Connectors integrate through a common web services framework called the Unified Service Model (USM) to expose data from a wide range of CA Technologies and third-party products for management by a set of consuming products. The *Unified Service Model (USM)* is the semantic schema that is used as the CA Catalyst and CA SOI infrastructure.

Connectors interact with consuming products and integrating products as follows:

Consuming Products

Consuming products leverage the infrastructure by consuming information that the connectors retrieve from integrated products. Consuming products provide a consolidated view of connector data for management in a unique, broader context. Consuming products and solutions include the following:

- CA SOI

  CA SOI consolidates data retrieved by connectors and lets you model services to represent cross-product data based on logical business functions. CA SOI is a layer of management that brings together the information from domain management products.

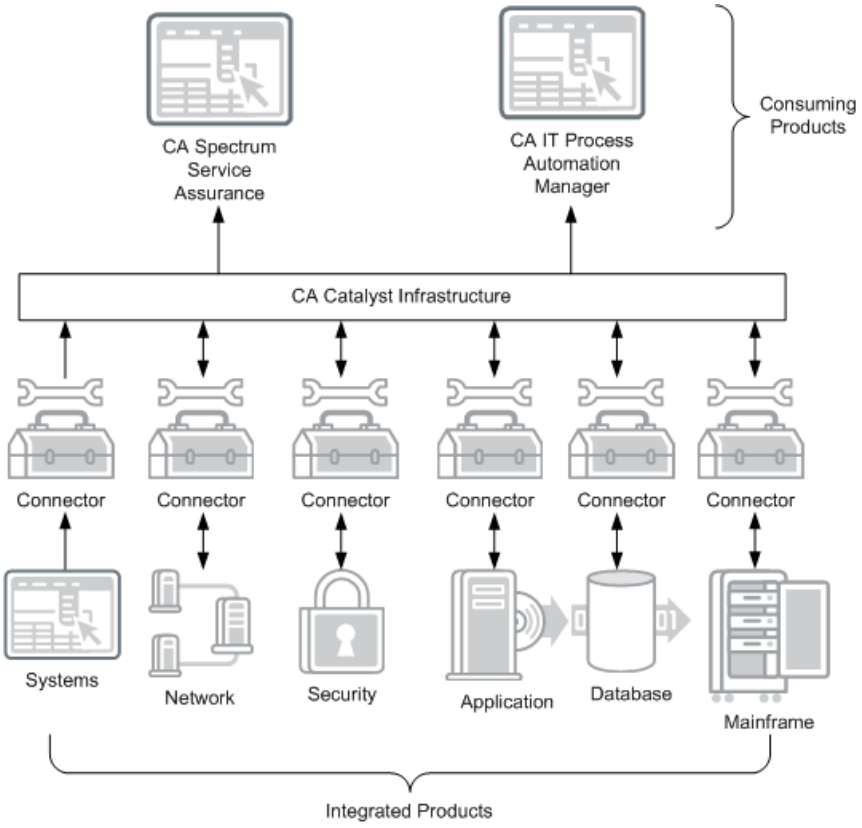
- CA Process Automation

  CA Process Automation lets you automate cross-product workflows by modeling processes that use functionality available from integrated products. CA Process Automation uses connectors to retrieve data from integrated products for use in process modeling.

Integrated Products

Integrated products are the products where connectors retrieve data. Each integrated product has a separate connector developed specifically to integrate with that product.

The following illustration shows how connectors retrieve data from integrated products, transform data into a common format using the infrastructure, and expose the data for consuming products to manage:



Chapter 4: Unified Service Model

This section explains the basic concepts of the Unified Service Model (USM).

This section contains the following topics:

What is USM?

The *Unified Service Model (USM)* is the semantic schema that is used as the CA Catalyst and CA SOI infrastructure. USM was developed to abstract and integrate information across many management products and domains and provide a single point for data federation, interoperability, and access to management data across an enterprise.

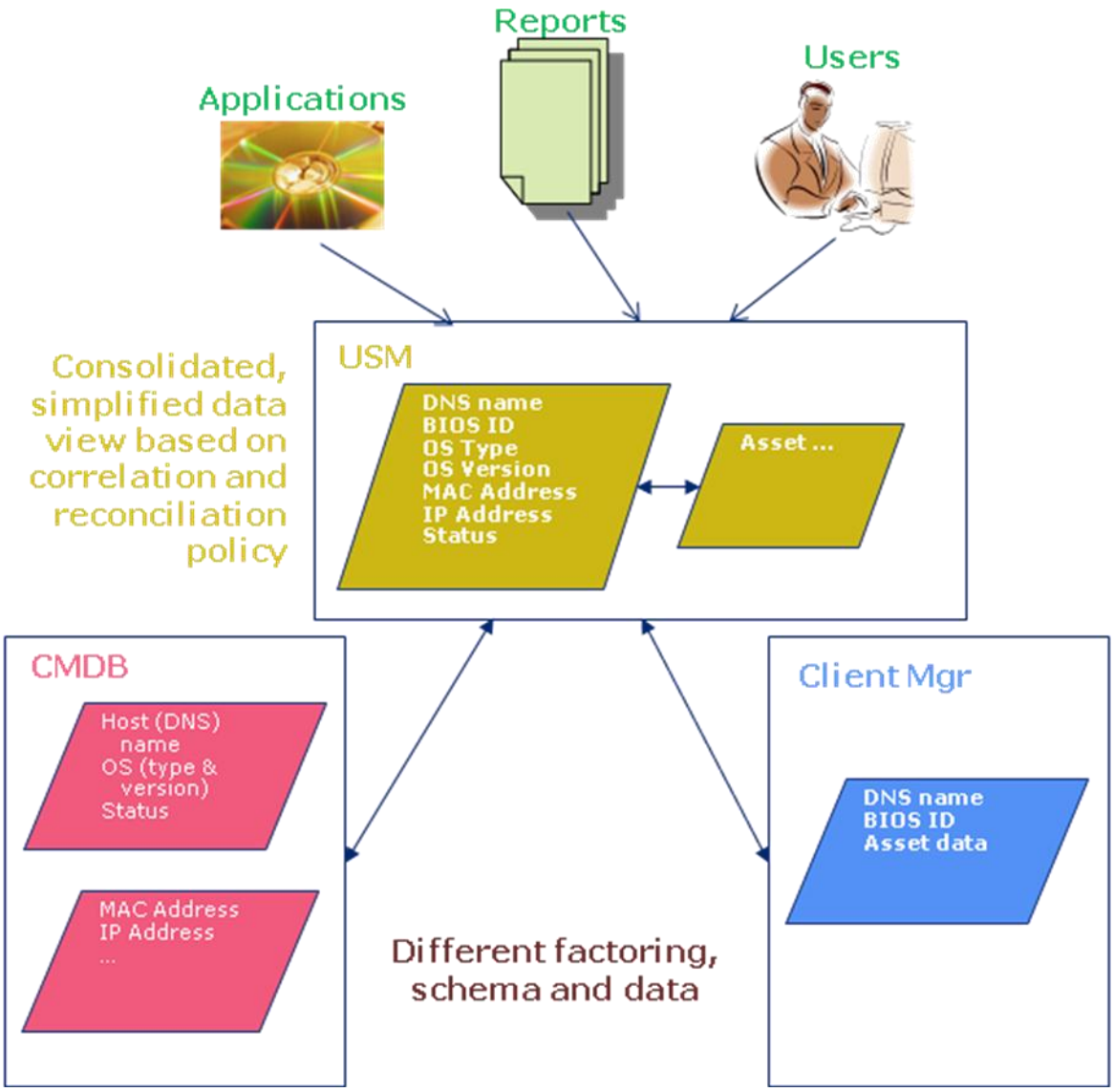
The USM schema defines the internal format for all CA Catalyst data, which is transmitted for display to CA SOI. Connectors transform all data collected from domain managers to the USM format before sending the data through CA Catalyst to CA SOI.

In IT management, specific domains exist such as networks, systems, fault, or security management systems and also specific standards exist for the domains. No mediating set of abstractions can discern the core concepts from the details and enable the merging of the specific domain semantics and data. USM addresses the mediation by defining logical, coherent, high-level, and common representations of general resource types, their data, and operations.

Specific data properties for USM types enable the following:

- High-level state management (available or unavailable)
- General alert management (informational to fatal alerts)
- CI identification and correlation
- Simple query (discover where to get more information)
- Facilitation of data manipulation, interoperability, and federation across a variety of IT management products

Data in different product-specific formats normalize to a single, consistent format that is correlated, reconciled, and accessed by users, applications, and reports.



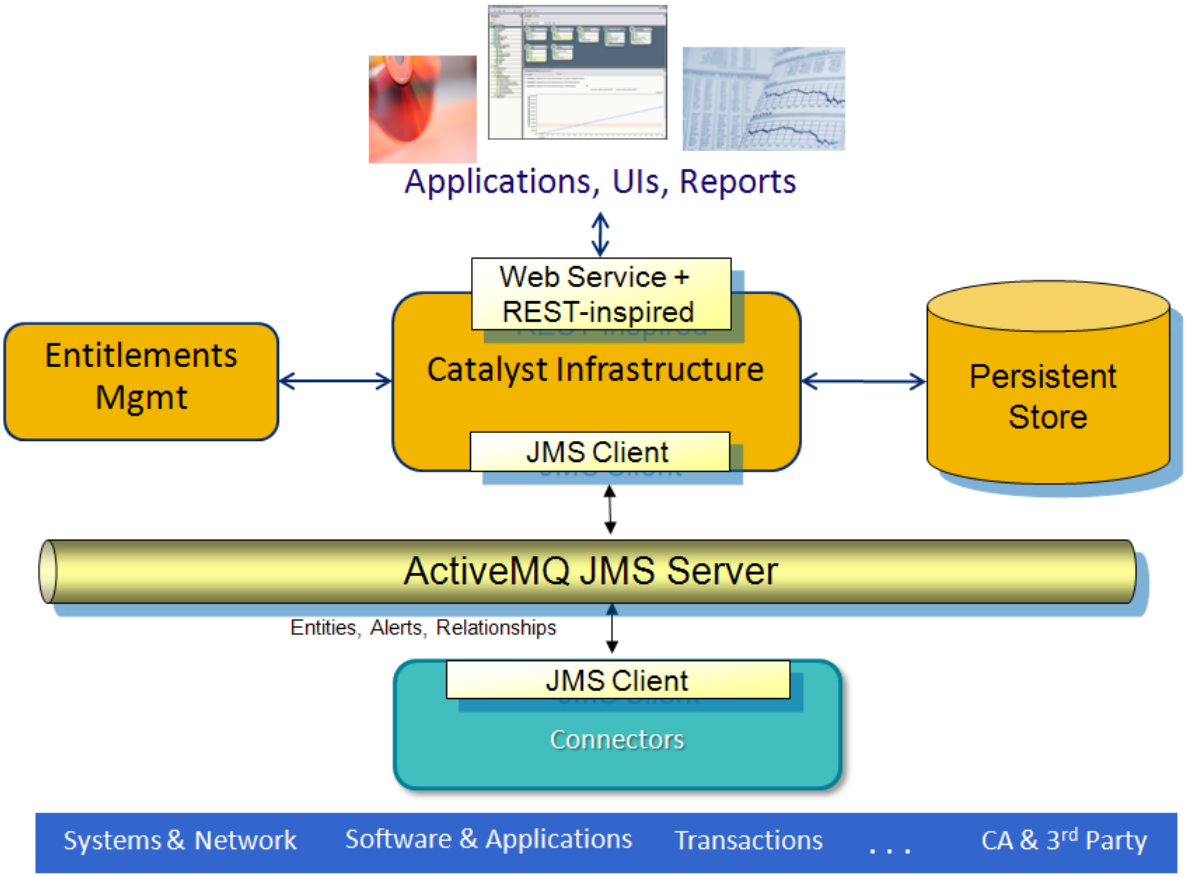
CA Catalyst is the integration and application platform that implements the USM schema, and CA SOI displays CA Catalyst data in USM format.

USM Schema

The USM schema provides facilities for CA Technologies and third-party management applications to do the following:

- Publish USM data and receive notifications of changes
- Persist and reconcile USM data
- Access data through web services and application interfaces

The following illustration shows the CA Catalyst infrastructure and how data from the infrastructure displays in CA SOI:



This infrastructure is critical to fully using USM semantics. Because each management product supports a limited set of management domains, none can provide a complete view of the IT and business environment. CA Catalyst facilitates the transformation of product- and domain-specific data to the USM syntax and semantics and keeps products notified of changes, additional information, and alerts published by other applications. The complete data set in the CA Catalyst store is correlated and reconciled, which creates a more inclusive and accurate representation of the environment.

Correlation

Correlation compares the key property values of a CI from one domain manager to another domain manager. CA Catalyst performs correlation and CA SOI, as a CA Catalyst consumer, can receive the correlated data from CA Catalyst. In CA Catalyst, correlation is based on one or more algorithms to achieve this mapping. You can adjust certain correlation criteria using additional semantics such as AND, OR, ANDIFEXIST, and NOT to modify the default rule. You can use the default rule, also named as PriorityListUSM, that better suits a solution.

Also, you can configure correlation to correlate across types that share a common base type. This results in *one* of the following:

- In a derived CI type correlating to a base type, which indicates both are identical
- In peers (children of the same base type) correlating, which is a multifunctional device (for example, a computer system acting as a router)

CA Catalyst supports relationship correlation that involves comparing source, target, scope, and semantics of a relationship with another CI from a different domain.

To correlate instances, the CA Catalyst infrastructure compares the following information from the USM schema:

- Type (root element) of the instance
- Correlation rule
- Multi CI correlation inclusion rule
- Equivalent element across types
- Identifying name for the instance (the element, InstanceName)

Note: For more information about correlation, see the CA Catalyst documentation.

Reconciliation

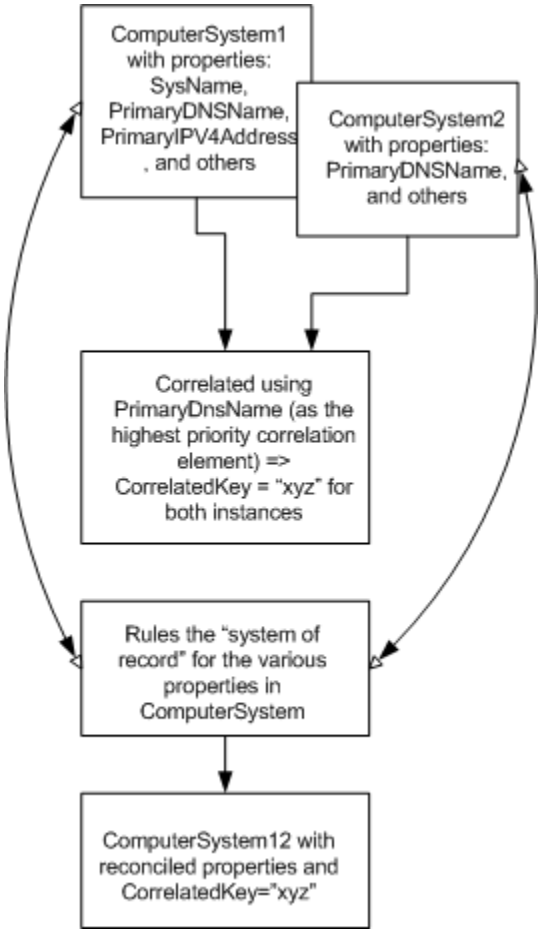
Reconciliation is the act of resolving differing property values across multiple CI projections into a single reconciled CI. Reconciliation operates across correlated CIs by using reconciliation formulas to determine property values that take precedence when discrepancies occur.

You can set various reconciliation formulas, including the following:

- First non-null wins
- Majority wins
- Data from specific domain manager wins
- Last in wins (default formula)

Note: For more information about reconciliation, see the *Administration Guide*.

The following illustration shows the correlation and reconciliation of two ComputerSystem CIs:



This illustration shows three instances of ComputerSystem—one from CA CMDB, one from CA ITCM, and a reconciled instance that CA Catalyst created. Some of the correlation elements overlap with each other and have the same CorrelatedKey value.

Advanced Features

The following are the advanced USM features:

- Custom Operations (see page 56)
- Metrics (see page 57)
- Profiles (see page 57)
- Filters (see page 57)

Custom Operations

Custom operations are operations that a connector can invoke to implement domain manager functionality that may be beneficial to a wider range of domain managers integrated through CA Catalyst. For example, a connector for an incident management product could implement a custom operation to create an incident for an alert on a CI managed by multiple domain managers.

Important! CA Catalyst consumers directly invoke custom operations. While developing custom connectors, you can build custom operations into those connectors, and consuming products that support the custom operations can then use them. However, CA SOI currently does not support invoking custom operations from its user interfaces.

Note: For more information about supported custom operations, see the "USM Specifics" appendix.

Metrics

Metric represents a measurement or reported value that characterizes the operation of a CI, such as a counter of incoming messages or a gauge reporting CPU utilization. The abilities to report the metrics available for an entity and to obtain the metric values are accomplished through a custom WSDL. A metric changes in response to external stimuli, as opposed to being configured.

Metrics data is gathered using the two USM custom operations, GetAvailableMetrics and GetMetricsValues. Connectors derive metrics from imported domain manager data and forward the same metrics information to the consuming product for viewing and analysis.

Important! CA Catalyst consumers directly support metrics. While developing custom connectors, you can build metrics information into those connectors, and consuming products that support metrics can then use them. However, CA SOI currently does not support viewing of collected metrics in its user interface.

Note: For more information about supported metrics, see the "USM Specifics" appendix.

Profiles

Profiles help you define a minimum standard set of capabilities and behavior across connectors. Usage of a profile facilitates interoperability and integrations. The Incident Profile is an example of the USM profile. Various CA Catalyst connectors such as BMC Atrium/Remedy and IBM Tivoli Service Request Manager use the Incident Profile.

Note: For more information about the Incident Profile, see the "USM Specifics" appendix.

Filters

Filters in the context of CA SOI are provisions that help connectors understand what type of data is being requested and retrieved from the domain manager. The get() (see page 143) connector method uses filters to retrieve information based on the predefined conditions.

SiloDataFilter (see page 145) is the primary filter that CA SOI uses when calling interfaces that the connector implements. SiloDataFilter supports various filter elements (such as entitytype, itemtype, recursive, and so on). Using combination of these filter elements, you can retrieve specific information from the domain manager.

For example, if the entitytype filter element is set to Item and itemtype to ComputerSystem, the connector would search for and display all entities of type ComputerSystem; it would not display any other entity. This way, you get to view and analyze only the required data in which you are interested.

Chapter 5: Connector Integrations

This section contains the following topics:

Connector Integration Types

The CA SOI infrastructure provides different types of connector integrations to accommodate different levels of detail. The following are the various levels and types of connector integrations that exist in provided connectors and that you can use to develop custom integrations:

- Level 1 connectors of type *CLI Based*
- Level 2 connectors of type *Generic Collection*
- Level 3 connectors of type *Process/Workflow*
- Level 4 connectors of type *Outbound without Service Import*
- Level 5 connectors of type *Outbound with Service Import*
- Level 6 connectors of type *Outbound and Inbound*

Level 1 Connectors

Level 1 connectors are *CLI-based* that can manually publish information to CA SOI. The provided Universal connector (see page 67) is a level 1 connector that lets you establish level 1 integrations. A Level 1 connector provides the following capabilities:

- Supports publishing of data to the consuming product (Publish)
- Supports importing of service representations (Service Import)
- Supports CIs from a domain manager
- Supports alerts from a domain manager

Level 2 Connectors

Level 2 connectors are of type *Generic Collection*. They can dynamically collect data from a generic source but cannot subscribe for changes to collected data. A Level 2 connector provides the following capabilities:

- Supports publishing of data to the consuming product (Publish)
- Supports CIs from a domain manager
- Supports alerts from a domain manager

Examples of Level 2 connectors include generic CA Catalyst connectors (such as SNMP) and the Event connector provided with CA SOI. You can use these connectors to establish custom Level 2 integrations based on generic data such as SNMP traps, text logs, and so on.

Level 3 Connectors

Level 3 connectors are of type *Process/Workflow*. They are responsible for complex process orchestration (for example, CA Process Automation workflow-based connectors). They include the ability to publish results for consumption by other management domains, subscribe to events from other connectors, and take actions based on a filtered set of events, CIs, or both to accomplish some form of business function (for example, provisioning a computer system or enabling the user account). A Level 3 connector provides the following capabilities:

- Supports publishing of data to the consuming product (Publish)
- Supports subscription to the domain manager changes (Subscribe)
- Supports inbound-to-connector operations: create, retrieve, update, and delete
- Supports advanced features such as metrics and custom operations.

Note: No Level 3 connectors are currently available that CA SOI can use.

Level 4 Connectors

Level 4 connectors are of type *Outbound without Service Import*. They can publish data and subscribe to domain manager changes but cannot import a service representation. A Level 4 connector provides the following capabilities:

- Supports publishing of data to the consuming product (Publish)
- Supports subscription to the domain manager changes (Subscribe)

- Supports advanced features such as metrics and custom operations, if applicable
- Supports CIs from a domain manager
- Supports alerts from a domain manager

Examples of Level 4 connectors include the NetAPP SANscreen connector and IBM Tivoli Monitoring connector.

You can build a custom Level 4 connector using the connector SDK provided within the Sample connector (see page 125).

Level 5 Connectors

Level 5 connectors are of type *Outbound with Service Import*. A Level 5 connector provides the following capabilities:

- Supports publishing of data to the consuming product (Publish)
- Supports subscription to the domain manager changes (Subscribe)
- Supports advanced features such as metrics and custom operations, if applicable
- Supports CIs from a domain manager
- Supports alerts from a domain manager
- Supports importing of service representations

Level 6 Connectors

Level 6 connectors are of type *Outbound and Inbound*. These connectors can perform both inbound-to-connector and outbound-to-connector operations. A Level 6 connector provides the following capabilities:

- Supports publishing of data to the consuming product (Publish)
- Supports subscription to the domain manager changes (Subscribe)
- Supports advanced features such as metrics and custom operations, if applicable
- Supports CIs from a domain manager
- Supports alerts from a domain manager
- Supports importing of service representations (Service Import)
- Supports inbound-to-connector operations: create, retrieve, update, and delete (if applicable)

Examples of Level 6 connectors include the IBM Tivoli Service Request Manager connector and BMC Atrium/Remedy connector.

Summary of Connector Integration Levels and Types

The following table summarizes the connector levels, types of connectors, and supported functionality:

Level	Type	Publish	Subscribe to Domain Manager Changes	Service Import	CIs	Alerts	Advanced Features (Metrics, Custom Operations)	Inbound-to-connector Operations
1	CLI Based	Yes	No	Yes	Yes	Yes	No	No
2	Generic Collection	Yes	No	No	Yes	Yes	No	No
3	Process/Work flow	Yes	Yes	No	No	No	Yes	Yes
4	Outbound without Service Import	Yes	Yes	No	Yes	Yes	Yes (if applicable)	No
5	Outbound with Service Import	Yes	Yes	Yes	Yes	Yes	Yes (if applicable)	No
6	Outbound and Inbound	Yes	Yes	Yes (if applicable)	Yes	Yes	Yes (if applicable)	Yes

Connector Integration Example Scenarios

This section includes connector integration example scenarios for the Universal connector, Generic connector, Event connector, and CA SOI connector.

Universal Connector Integration

The Universal connector provides command line and web services programming interfaces that let the connector forward CIs, events, and services to CA SOI. The Universal connector is appropriate for Level 1 integrations with CA SOI provided any of the following conditions are met:

- The domain manager has a script interface that calls the Universal connector command line interface to create CIs and alerts in CA SOI.
- The domain manager has a programming interface that the Universal connector web services interface can use to create CIs and alerts in CA SOI.

- Custom events are forwarded to CA SOI.

  Note: The CA NSM event management message action can also use the Universal connector if custom events require forwarding to CA SOI.

The Universal connector only forwards events to the SA Manager, which indicates the following:

- The integrated application must call the Universal connector interface.
- It is not possible to get information from the integrated application. For example, you cannot import services from the Universal connector in the Operations Console.
- Because CA SOI cannot get the current status for a CI, the status for the Universal connector CIs might not be current after a CA SOI server restart or in newly created service models.

Generic Connector Integration

Generic data collection connectors can dynamically collect data from various generic sources. They process and convert the data to the USM format using a connector policy and dispatch it to the consuming product. However, unlike product-specific connectors, generic connectors do not include any product-specific policy. One fixed default policy for the generic connector is not possible, because the type and format of the received information varies from one domain manager to another. Therefore, for Level 2 integrations using a generic connector, you must customize a template policy file so that the policy appropriately processes data from the data source and displays meaningful information in CA SOI.

The CA Catalyst connector for SNMP is a good example of Level 2 integration. This SNMP connector acts as an SNMP trap destination for any product that supports SNMP, which helps ensure that you get access to the SNMP trap data from a wide range of products. The ability to collect, transform, analyze, and manage SNMP trap data in CA SOI lets you determine trends and take appropriate measures for SNMP-capable devices in your enterprise. For example, you can analyze an SNMP trap in CA SOI to identify the root cause of the trap. You can determine whether the trap is because of an extraordinary event (indicating an issue or an error has occurred) or a confirmed event providing status information, such as a process ending normally or a printer coming online. Based on your analysis, you can take corrective actions to fine-tune or rectify the situation.

While developing a generic connector (Level 2) integration, consider the following:

- You can monitor additions and modifications to events (such as SNMP traps) coming from a domain manager by using a generic connector.
- For each new entity that generates an event, the first event is considered as a CI and an alert, and any subsequent event from the same source is considered only as an alert.

- You cannot subscribe for changes to collected domain manager data using a generic connector.
- You cannot support importing of service representations, advanced features (such as metrics and custom operations), and inbound-to-connector operations using a generic connector.
- You can map object types of the integrating domain manager as appropriate. For example, in case of the SNMP connector, any CA Workload Automation job scheduled on the specific agent host is a CI and is mapped to the USM type ITActivity.
- You must write a policy specific to the integrating domain manager.
- You must formulate, understand, and follow the event mapping guidelines for writing the policy. For example, in case of the SNMP connector, all system-level traps from an SNMP product are mapped to the ComputerSystem USM type; similarly, all CPU-related traps are mapped to the Processor USM type, and so on.

  Note: For more information about how to download the connector and accompanying documentation, see the Download Connectors and Prepare for Installation (see page 22) section.

Note: In those scenarios where both the connectors—generic connector and the Event connector—are applicable, we recommend that you use the generic connector.

Event Connector Integration

The Event connector (see page 97) uses the integration adaptors provided with the CA Event Integration installation, which is deployed with CA SOI. The adaptors support a variety of sources and are driven by CA Event Integration source policy, which you can tailor based on your specific needs. The following are the integration adaptors:

LogReader

Scans file system log files local to the application system.

Policy File: Applog-src

WinSysLog

Scans Windows system, application, or security event logs.

Policy File: Syslog-src

SNMPplugin

Receives SNMP traps on the configured port.

Policy Files: Snmp-src, hpbac-src, mfopsmvs-src, and mfsysview-src

UniEvent

Reads CA NSM events. It is local to the CA NSM Event Manager.

Policy File: Nsmevent-src

WSEventing

Defines an event publisher from which to collect web service notifications and the event details of the publisher.

Policy File: Wsevent-src

You can develop a policy-based Event connector integration if the integrated application already writes events to log files or uses another of the provided adaptor method (such as SNMP trap communication). The Event connector processes this information with the following considerations:

- You create or modify a CA Event Integration source policy that instruments the appropriate adaptor.
- Depending on the adaptor type, you might need an Event connector installation that is local to the integrated application. You can use central event receivers on existing CA Event Integration manager installations.
- The connector retrieves events from the source, uses the policy to transform the events, and forwards them to CA SOI.
- Depending on the configured reconciliation level, the connector implicitly creates event-related, device-level, and instance-level CIs.
- You cannot create services and relationships.
- Full synchronization is not provided; therefore, the status for Event connector CIs might not be current after a CA SOI server restart or in newly created service models

Note: For more information about developing CA Event Integration source policy, see the *CA Event Integration Product* Guide.

CA SOI Connector Integration

You can achieve the deepest and most robust level of integration when building a custom Level 4, 5, or 6 (see page 59) CA SOI connector. You develop a custom connector in Java using the connector SDK provided with the Sample connector. A custom connector has the following characteristics:

- Receives requests from the SA Manager
- Synchronizes with the SA Manager

  A CA SOI server restart or creation of a new service model causes a request to the connector for the current CI alerts and verifies that the CI status in CA SOI is up-to-date

- Queries the third-party manager native interface (API or files) for CIs and alerts
- Implements an event listener for forwarding to CA SOI
- Supports service import requests from the Operations Console (if applicable)
- Embedded in the Windows service CA SAM Integration Services

You can also build advanced features into custom connectors, such as the following:

- Metric collection
- Custom operations
- Inbound to connector operations

Note: For more information, see the "Building a Custom Connector Using the Sample Connector" section.

Chapter 6: Using the Universal Connector

This section introduces the Universal connector and describes various components of the Universal connector, Universal connector web service methods, and installation information for the connector. The section also explains the command line interface and programming interface (see page 92) to manage the Universal connector.

This section contains the following topics:

Universal Connector

CA SOI provides a Universal connector in the IFW that can retrieve services, CIs, and events from various CA Technologies and third-party products. The Universal connector provides command line and web services programming interfaces that third-party products can use to publish new services, CIs, and events to the web service, which is polled periodically for these events. These events are normalized to the USM format and made available to the SA Manager.

The Universal connector is appropriate for Level 1 integrations with CA SOI provided any of the following conditions are met:

- The domain manager has a script interface that calls the Universal connector command-line interface to create CIs and alerts in CA SOI.
- The domain manager has a programming interface that the Universal connector web services interface can use to create CIs and alerts in CA SOI.
- Custom events are forwarded to CA SOI.

  Note: The CA NSM event management message action can also use the Universal connector if custom events require forwarding to CA SOI.

The Universal connector only forwards events to the SA Manager, which indicates the following:

- The integrated application must call the Universal connector interface.
- It is not possible to get information from the integrated application. For example, you cannot import services from the Universal connector in the Operations Console.
- Because CA SOI cannot get the current status for a CI, the status for the Universal connector CIs might not be current after a CA SOI server restart or in newly created service models.

Universal Connector Components

The Universal connector consists of the following main components:

Connector

Polls the web service periodically for services, CIs, and events; normalizes these items to the USM format; and makes them available to the SA Manager. The connector is part of the IFW.

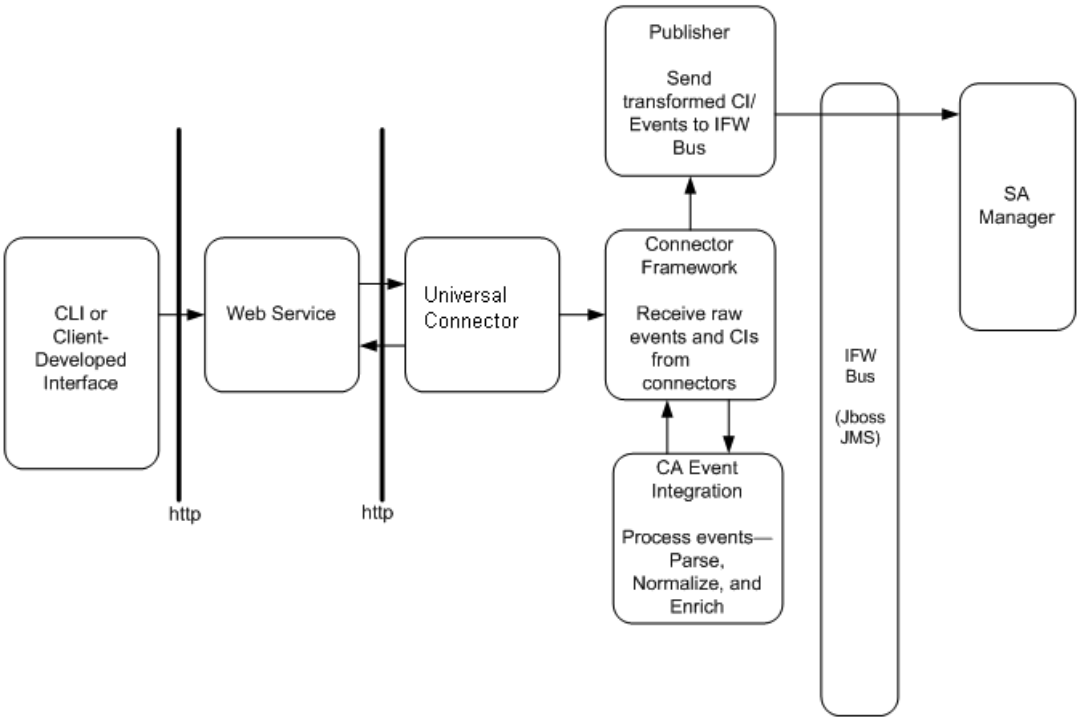
Web Service

Lets various CA Technologies and third-party products send services, CIs, and events for the Universal connector to collect and publish to CA SOI. The web service runs on a Tomcat server that by default is on the SA Manager, is named GenericConnectorService, and is located at *http://SA-Manager:7090/axis2/services/GenericConnectorService?wsdl*. The server polls for the events that the web service client forwards.

Web Service Client

Communicates with the web service using the web service interface. It provides a command-line interface (GCEventAddCmd.bat) that implements various web services methods to load services, CIs, and events into the web service. The web service client also provides supporting libraries for user programs to use the web services interface. This component is installed when you select the Universal connector installation option in the CA SOI installation wizard.

The following illustration shows the Universal connector process flow:



Install the Universal Connector Client

The Universal Connector Client installs the framework necessary to run the Universal connector. The Universal connector lets you add CIs, relationships, services, and alerts through a command line or programmatic interface.

A fully functional Universal connector installs automatically with the SA Manager. You can also install the Universal connector client on a separate system to enable use of the Universal connector command line interface on that system.

Note: The merge modules that are included in the same Installers folder of the installation image must remain in the same directory as the installer for the installation to work.

Follow these steps:

1. Run soi-installer.exe from the Disk1\SOI folder of the CA SOI installation image.

   The License Agreement page opens.

2. Accept the license agreement, then accept the third-party license agreements, and click Next.

   The Choose Install Folder screen opens.

   Note: If CA SOI components are already installed on the system, the Choose Install Folder page does not appear. Any new CA SOI components that you install are installed to the existing CA SOI directory, because all components must reside in the same directory.

3. Accept, enter, or choose the installation folder.

   Note: The maximum installation path length is 150 characters. The installation blocks paths with more than 150 characters.

   The Choose Install Set screen opens.

4. Select Universal (client utility), and click Next.

   The Pre-Installation Summary page opens.

5. Click Install.

   The Universal Client functionality installs on the system. The Install Complete page opens when the installation finishes.

6. Click Done.

7. (Optional) Review the installation log file (CA_Service_Operations_Insight_Install_*releasenumber*) that is in the SOI_HOME\log folder to check for installation errors. This folder also provides the installation log files for the other installed components.

Configure the Universal Connector

When you install the Universal connector, it is automatically enabled with default settings. You can change the default settings and controls.

Follow these steps:

1. Access the Dashboard, select the Administration tab, and select the Connector Configuration option.
2. Select the Universal connector in the Connector column.

   The Universal connector details page opens and displays a number of tables with more details about the connector and the server where it is installed.
3. (Optional) Change any of the connector controls in the Connector Controls table and click Save.
4. Click any of the following Value fields in the Connector Details table and click Save:

   host

   Indicates the host name for the Tomcat application server where the web service is running.

   port

   Indicates the port number for the Tomcat application server where the web service is running.

   poll interval

   Specifies the number of seconds to wait between polling for events.
5. Click Stop and wait until the connector status changes to Offline.
6. Click Start and wait until the connector status changes to Online.

   Note: It may take a few minutes before the connector status displays Online.

Important! Do not perform rapid start and stop operations on the connector. Each stop and start sends the corresponding command to the connector. Rapid start and stop operations from the interface can cause these commands to queue on the connector and cause the connector to start and stop until all commands in the queue are processed.

Command Line Interface

Use the GCEventAddCmd batch program (GCEventAddCmd.bat) to manage the Universal connector from a command line. All appropriate files (such as GCEventAddCmd batch file, sample XML files) are installed in the CA SOI root directory, for example C:\Program Files\CA\SOI. The batch program lets you create services, CIs, relationships, and alerts in CA SOI from any program that is able to run commands.

Note: For information about the programming interface, see the Programming Interface section.

You can run the GCEventAddCmd batch program based on the following scenarios:

- For publishing a single event, we recommend that you run the batch program using the appropriate command line parameters. However, if you want, you can also use the XML file as described in the next point.
- For publishing multiple events, CIs, and services, run the batch program using the appropriate XML file. In this case, parameters are stored in the appropriate XML files.

Note: Typically, the CA SOI root directory is not in the system search path (Windows %PATH% environment). Therefore, prefix the CA SOI directory, such as C:\Program Files\CA\SOI, while running GCEventAddCmd.bat.

GCEventAddCmd.bat—Run the Universal Connector

The batch program GCEventAddCmd.bat wraps a web services client that can publish services, CIs, and events from the Command Prompt window using USM properties. The batch program is located in the root directory of the CA SOI installation.

Publish a Single Status Event

The batch program can publish a single event to CA SOI when specified with the appropriate command line parameters. For a single event, you can pass most of the parameters in the argument list. In this case, run the batch program as follows:

```
GCEventAddCmd -h<wsHostName:wsPort> -a<MdrElementID> -i<AlertedMdrElementID>
-s<severity> -t<AlertType>  -m<summary>
```

-h*<wsHostName:wsPort>*

Specifies the web services host name and port number, which is the same as the SA Manager host name and port number. By default, the SA Manager port is 7090.

-a*<MdrElementID>*

Specifies the unique alert ID.

-i*<AlertedMdrElementID>*

Specifies the AlertedMdrElementID, which is the MdrElementID of the CI that the alert is associated with.

-s*<severity>*

Specifies the alert severity. The valid values are Normal, Minor, Major, Critical, Down.

-t<*AlertType*>

Specifies the type of alert. The valid values are Risk and Quality.

-m<*summary*>

Specifies the appropriate message text that you want to display.

Publish Multiple Status Events, CIs, and Services

When the batch program is used with an XML file, the batch program can publish multiple events, CIs, and services. The parameters are stored in the XML file. The XML file acts as a source for the Universal connector.

In this case, run the batch program as follows:

```
GCEventAddCmd -h<wsHostName:wsPort> -f<XML file>
```

-h<wsHostName:wsPort>

Specifies the web services host name and port number, which is the same as the SA Manager host name and port number. By default, the SA Manager port is 7090.

-f<XML file>

Specifies the XML file name.

The following example shows how you can use an XML file (for example, Add_alerts.xml) to run the batch program:

```
GCEventAddCmd –hSOIserver.ca.com:7090 –fAdd_alerts.xml
```

eventType, MdrElementID, and className are the required USM properties for any XML file that you want to use. All other USM properties depend on the type of CI you are adding. Refer the USM schema documentation link on the bookshelf to determine which USM properties are required for each class of CI. For example, the Service CI requires ServiceName and ServiceVersion.

Note: For more information about how to access the USM schema or how to use the USM Web View, see the How to Find USM Properties for a CI section.

To understand how to use and create your XML files, you can review the example scenarios provided in the Integration Scenario Examples section. You can also view the sample XML files available in the Sample XML Files section.

Integration Scenario Examples

The following are example scenarios where you can use the GCEventAddCmd batch program to achieve specific tasks:

- Create a service
- Create a CI
- Add an alert
- Add a CI to a service
- Update a CI
- Delete a CI

Note: If you want to know the required USM properties to include in a certain USM type (such as service, computer_system, application, and alert), check the USM schema documentation.

Create a Service

To create a service, run the GCEventAddCmd batch program with an XML file that contains the <Service> tag and appropriate property tags with corresponding values. The template tag structure is as follows:

```
<Services>
        <Service>
          ...
          ...
        </Service>
</Services>
```

Note: For a complete list of USM property tags, see the USM schema to determine which USM properties are required for this class of CI.

The following example lets you create a service called MyService by running the batch program as follows:

```
GCEventAddCmd –hSOIserver.ca.com:7090 –fMyService.xml
```

The example uses the MyService.xml file that includes the following information:

```
<Services>
        <Service>
          <property tag="eventType" value="AddService" />
          <property tag="MdrElementID" value="MyService" />
          <property tag="className" value="Service" />
```

```
          <property tag="ServiceName" value="MyService" />
          <property tag="ServiceVersion" value="0" />
          <property tag="Description" value="Adding a new Service" />
          <property tag="AdministrativeStatus" value="Managed" />
        </Service>
</Services>
```

The USM properties used in this example XML file are as follows:

eventType

Specifies the type of event. For this example, the value is AddService.

MdrElementID

Specifies a unique identifier for the service. For this example, the value is MyService.

className

Specifies the USM class name. For this example, the value is Service.

ServiceName

Specifies the name of the service that distinguishes it from others. For this example, the value is MyService.

ServiceVersion

Specifies the version level of the service. For this example, the value is 0.

Description

Specifies a detailed description of the service. For this example, the value is Adding a new Service.

AdministrativeStatus

Specifies the high-level status of the service. For this example, the value is Managed.

Create a CI

You can manage CIs in two different XML tag contexts—<Events> tag and <Services> tag. The combination with other actions determines the XML tag context of your choice. For example, if you want to manage a CI along with a service, use the <Services>...</Services> tags. If you want to create an event along with the CI creation, use the <Events>...</Events> tags.

- For Events tag, the structure to create a CI is as follows:

```
<Events>
    <Event>
      eventType = AddCIEvent
```

- For Services tag, the structure to create a CI is as follows:

```
<Services>
    <Service>
      eventType = AddCI
```

Note: For a complete list of USM property tags, see the USM schema to determine which USM properties are required for the specific class of CI.

The following example lets you create a CI called MyMachine of the USM type ComputerSystem by running the batch program as follows:

```
GCEventAddCmd –hSOIserver.ca.com:7090 –fCreateCI.xml
```

The example uses the CreateCI.xml file that includes the following information about the CI:

```
<Events>
        <Event>
          <property tag="eventType" value="AddCIEvent" />
          <property tag="MdrElementID" value="MyMachine" />
          <property tag="className" value="ComputerSystem" />
          <property tag="Label" value="MyMachine" />
          <property tag="Description" value="Description of a Universal Connector
Entity" />
          <property tag="AdministrativeStatus" value="Managed" />
          <property tag="Vendor" value="Dell" />
          <property tag="PrimaryDnsName" value="MyMachine.ca.com" />
          <property tag="ComputerName" value="MyMachine" />
          <property tag="MemoryInGB" value="1024" />
        </Event>
</Events>
```

The parameters used in this example XML file are as follows:

eventType

Specifies the type of event. For this example, the value is AddCIEvent.

MdrElementID

Specifies a unique identifier for the CI. For this example, the value is MyMachine.

className

Specifies the USM class name. For this example, the value is ComputerSystem.

Label

Specifies a short description for identification of the CI. It is important that the value of the label uniquely distinguishes entities in the interface. For this example, the value is MyMachine.

Description

Specifies a description of the CI. For this example, the value is "Description of a Universal Connector Entity".

AdministrativeStatus

Specifies the high-level administrative status of the CI. For this example, the value is Managed.

Vendor

Specifies the hardware vendor name. For this example, the value is Dell.

PrimaryDnsName

Specifies the fully qualified DNS name of the CI. For this example, the value is MyMachine.ca.com.

ComputerName

Specifies the name of the computer. For this example, the value is MyMachine.

MemoryInGB

Specifies the amount of system memory in GB. For this example, the value is 1024.

Add an Alert

To add alerts using an XML file, run the GCEventAddCmd batch program with the XML file that contains the <Event> tag and appropriate property tags with corresponding values. The template tag structure is as follows:

```
<Events>
        <Event>
          …
          …
        </Event>
</Events>
```

Note: For a complete list of USM property tags, see the USM schema to determine which USM properties are required for the specific class of CI.

The following example lets you add an alert of Minor severity by running the batch program as follows:

```
GCEventAddCmd –hSOIserver.ca.com:7090 –fAddAlert.xml
```

The example uses the AddAlert.xml file that includes the following information about the alert:

```
<Events>
        <Event>
          <property tag="eventType" value="StatusEvent" />
          <property tag="MdrElementID" value="A00001" />
          <property tag="className" value="Alert" />
          <property tag="AlertedMdrElementID" value="UCServer" />
          <property tag="AlertType" value="Risk" />
          <property tag="Severity" value="Minor" />
          <property tag="Summary" value="UC_Server has an infrastructure alarm.." />
          <property tag="Message" value="The Detailed message associated with this
alert.." />
        </Event>
</Events>
```

The parameters used in this example XML file are as follows:

eventType

Specifies the type of event. In the case of alert, the value is StatusEvent.

MdrElementID

Specifies a unique alert ID. For this example, the value is A00001.

className

Specifies the USM class name. For this example, the value is Alert.

AlertedMdrElementID

Specifies the AlertedMdrElementID, which is the MdrElementID of the CI that the alert is associated with. For this example, the value is UCServer.

AlertType

Specifies the type of alert: Risk, Quality, or Health. For this example, the value is Risk.

Severity

Specifies the alert severity. For this example, the value is Minor.

Summary

Specifies the appropriate summary text that you want to display. For this example, the value is "UC_Server has an infrastructure alarm..".

Message

Specifies the appropriate message text that you want to display. For this example, the value is "The Detailed message associated with this alert..".

Add a CI to a Service

To add a CI to a service, run the GCEventAddCmd batch program with an XML file containing the <Relationship> tag and appropriate required properties with corresponding values.

Note: For a complete list of USM property tags, see the USM schema to determine which USM properties are required for the specific class of CI.

The following example lets you add a CI to a service by running the batch program as follows:

```
GCEventAddCmd –hSOIserver.ca.com:7090 –fMyServiceRelation.xml
```

The MyServiceRelation.xml file includes the following information:

```
<Services>
        <Relationship>
          <property tag="eventType" value="AddRelationship" />
          <property tag="SourceMdrElementID" value="UCService" />
          <property tag="TargetMdrElementID" value="UCServer" />
          <property tag="ScopeMdrElementID" value="UCService" />
          <property tag="Semantic" value="HasMember" />
          <property tag="className" value="BinaryRelationship" />
        </Relationship>
</Services>
```

The parameters used in this example XML file are as follows:

eventType

Specifies the type of event. For this example, the value is AddRelationship.

SourceMdrElementID

Specifies the unique source identifier. For this example, the value is UCService.

TargetMdrElementID

Specifies the unique target identifier. For this example, the value is UCServer.

ScopeMdrElementID

Specifies the service in which the relationship exists. For this example, the value is UCService.

Semantic

Relates one USM instance to another. For this example, the value is HasMember.

className

Specifies the USM class name. For this example, the value is BinaryRelationship.

Update a CI

You can manage CIs in two different XML tag contexts—<Events> tag and <Services> tag.

- For Events tag, the structure to update a CI is as follows:

```
<Events>
    <Event>
      eventType = UpdateCIEvent
```

- For Services tag, the structure to update a CI is as follows:

```
<Services>
    <Service>
      eventType = UpdateCI
```

Note: For a complete list of USM property tags, see the USM schema to determine which USM properties are required for the specific class of CI.

The following example lets you update a CI description by running the batch program as follows:

```
GCEventAddCmd –hSOIserver.ca.com:7090 –fUpdateCI.xml
```

The UpdateCI.xml file includes the following information:

```
<Events>
        <Event>
          <property tag="eventType" value="UpdateCIEvent" />
          <property tag="MdrElementID" value="MyMachine" />
          <property tag="className" value="ComputerSystem" />
          <property tag="Label" value="MyMachine" />
          <property tag="Description" value="Update to the Description" />
          <property tag="AdministrativeStatus" value="Managed" />
          <property tag="Vendor" value="Dell" />
          <property tag="PrimaryDnsName" value="MyMachine.ca.com" />
          <property tag="ComputerName" value="MyMachine" />
          <property tag="MemoryInGB" value="1024" />
        </Event>
</Events>
```

The parameters used in this example XML file are as follows:

eventType

Specifies the type of event. For this example, the value is UpdateCIEvent.

MdrElementID

Specifies a unique identifier for the CI. For this example, the value is MyMachine.

className

Specifies the USM class name. For this example, the value is ComputerSystem.

Label

Specifies a short description for identification of the CI. It is important that the value of the Label uniquely distinguishes entities in the interface. For this example, the value is MyMachine.

Description

Specifies a description of the CI. For this example, the value is "Update to the Description".

AdministrativeStatus

Specifies the high-level administrative status of the CI. For this example, the value is Managed.

Vendor

Specifies the hardware vendor name. For this example, the value is Dell.

PrimaryDnsName

Specifies the fully qualified DNS name of the entity. For this example, the value is MyMachine.ca.com.

ComputerName

Specifies the name of the computer. For this example, the value is MyMachine.

MemoryInGB

Specifies the amount of system memory. For this example, the value is 1024.

Delete a CI

You can manage CIs in two different XML tag contexts—<Events> tag and <Services> tag.

- For <Events> tag, the structure to delete a CI is as follows:

```
<Events>
    <Event>
      eventType = DeleteCIEvent
```

- For <Services> tag, the structure to delete a CI is as follows:

```
<Services>
    <Service>
      eventType = DeleteCI
```

Note: For a complete list of USM property tags, see the USM schema to determine which USM properties are required for the specific class of CI.

You can delete only those CIs (by using the Universal connector) that the Universal connector has created. You cannot delete a CI from a different connector.

The following example lets you delete a CI called MyMachine of USM type ComputerSystem by running the batch program as follows:

```
GCEventAddCmd –hSOIserver.ca.com:7090 –fDeleteCI.xml
```

The DeleteCI.xml file includes the following information:

```
<Events>
        <Event>
          <property tag="eventType" value="DeleteCIEvent" />
          <property tag="MdrElementID" value="MyMachine" />
          <property tag="className" value="ComputerSystem" />
          <property tag="Label" value="MyMachine" />
          <property tag="Description" value="Delete the ComputerSystem" />
          <property tag="AdministrativeStatus" value="Managed" />
          <property tag="Vendor" value="Dell" />
          <property tag="PrimaryDnsName" value="MyMachine.ca.com" />
          <property tag="ComputerName" value="MyMachine" />
          <property tag="MemoryInGB" value="1024" />
        </Event>
</Events>
```

The parameters used in this example XML file are as follows:

eventType

Specifies the type of event. For this example, the value is DeleteCIEvent.

MdrElementID

Specifies a unique identifier for the CI. For this example, the value is MyMachine.

className

Specifies the USM class name. For this example, the value is ComputerSystem.

Label

Specifies a short description for identification of the CI. It is important that the value of the Label uniquely distinguishes entities in the interface. For this example, the value is MyMachine.

Description

Specifies a description of the CI. For this example, the value is "Delete the ComputerSystem".

AdministrativeStatus

Specifies the high-level administrative status of the CI. For this example, the value is Managed.

Vendor

Specifies the hardware vendor name. For this example, the value is Dell.

PrimaryDnsName

Specifies the fully qualified DNS name of the entity. For this example, the value is MyMachine.ca.com.

ComputerName

Specifies the name of the computer. For this example, the value is MyMachine.

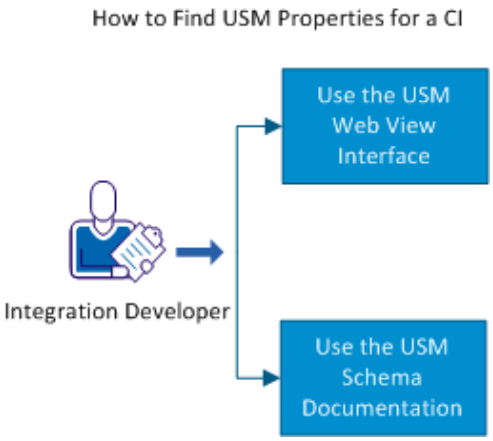
MemoryInGB

Specifies the amount of system memory. For this example, the value is 1024.

How to Find USM Properties for a CI

As an integration developer, you provide USM properties information while using the Universal connector command line utility. You can provide complete and correct information if you know how to find various USM properties that are associated with a CI. This scenario helps you locate all USM properties for a CI.

Use the following scenario to guide you through the process:



Use the following methods to find the USM CI properties:

- Use the USM Web View Interface.
- Use the USM Schema Documentation.

Use the USM Web View Interface

You can use the USM Web View interface to browse and search for all USM data that is available in the CA Catalyst Persistence Store.

You can access USM Web View from the CA SOI Dashboard. To access the USM Web View interface from the CA SOI Dashboard, open the CA SOI Dashboard and click the USM Web View link available on the Dashboard. The USM Web View interface opens to the home page.

Note: For more information about working with USM Web View, see the *Administration Guide*.

Use the USM Schema Documentation

The USM schema HTML documentation includes detailed information about all USM types, USM properties, schema summaries, and so on. You can access the USM schema documentation from the CA SOI Bookshelf.

Sample Universal Connector XML Files

Installing the Universal connector also installs sample XML files that help you understand the XML file structure and various property tags. These XML files provide examples of the format required for the type of items that you want to apply to CA SOI. The following are the sample XML files that you can find at the CA SOI root location:

- universalAdd_Alerts (see page 85)
- universalAdd_Application (see page 85)
- universalAdd_ApplicationServer (see page 85)
- universalAdd_ComputerSystem (see page 86)
- universalAdd_Database (see page 86)
- universalAdd_FileSystem (see page 87)
- universalAdd_Memory (see page 87)
- universalAdd_Processor (see page 88)
- universalAdd_Service (see page 88)
- universalAdd_ServiceModel (see page 89)
- universalDelete_ComputerSystem (see page 90)
- universalUpdate_ComputerSystem (see page 90)

universalAdd_Alerts

This sample XML file helps you add an alert. The structure of this sample file is as follows:

```
<Events>
        <Event>
          <property tag="eventType" value="StatusEvent" />
          <property tag="MdrElementID" value="A00001" />
          <property tag="className" value="Alert" />
          <property tag="AlertedMdrElementID" value="UCServer" />
          <property tag="AlertType" value="Risk" />
          <property tag="Severity" value="Minor" />
          <property tag="Summary" value="UC_Server has an infrastructure alarm.." />
          <property tag="Message" value="The Detailed message associated with this
alert.." />
        </Event>
</Events>
```

universalAdd_Application

This sample XML file helps you add a new application CI. The structure of this sample file is as follows:

```
<Events>
        <Event>
          <property tag="eventType" value="AddCIEvent" />
          <property tag="MdrElementID" value="Application:Login" />
          <property tag="Label" value="Application:Login" />
          <property tag="className" value="Application" />
          <property tag="Description" value="Adding a new Application" />
          <property tag="Vendor" value="CA" />
          <property tag="ProductName" value="Login" />
          <property tag="DeviceSysName" value="CA:Login" />
        </Event>
</Events>
```

universalAdd_ApplicationServer

This sample XML file helps you add a new application server CI. The structure of this sample file is as follows:

```
<Events>
        <Event>
          <property tag="eventType" value="AddCIEvent" />
          <property tag="MdrElementID" value="ExchangeServer" />
          <property tag="className" value="ApplicationServer" />
```

```
            <property tag="Description" value="Adding a new Application Server" />
            <property tag="category" value="Exchange Services" />
            <property tag="Label" value="ExchangeServer" />
            <property tag="Vendor" value="CA" />
            <property tag="ProductName" value="Exchange" />
            <property tag="ProcessID" value="01" />
            <property tag="AccessedViaTcpPort" value="01" />
            <property tag="ProcessDistinguishingID" value="01" />
            <property tag="DeviceSysName" value="CA:ExchangeServer" />
          </Event>
</Events>
```

universalAdd_ComputerSystem

This sample XML file helps you add a computer system CI. The structure of this sample file is as follows:

```
<Events>
        <Event>
          <property tag="eventType" value="AddCIEvent" />
          <property tag="MdrElementID" value="MyMachine21" />
          <property tag="className" value="ComputerSystem" />
          <property tag="Label" value="MyMachine21" />
          <property tag="Description" value="Description of a Universal Connector
Entity" />
          <property tag="AdministrativeStatus" value="Managed" />
          <property tag="Vendor" value="Dell" />
          <property tag="PrimaryDnsName" value="MyMachine21.ca.com" />
          <property tag="ComputerName" value="MyMachine21" />
          <property tag="MemoryInGB" value="1024" />
        </Event>
</Events>
```

universalAdd_Database

This sample XML file helps you add a database CI. The structure of this sample file is as follows:

```
<Events>
        <Event>
          <property tag="eventType" value="AddCIEvent" />
          <property tag="MdrElelentID" value="ReplicationDatabase01" />
          <property tag="className" value="Database" />
          <property tag="Description" value="Adding a new Database" />
          <property tag="category" value="Replication Services" />
          <property tag="Label" value="ReplicationDatabase01" />
```

```
        <property tag="DBInstanceName" value="ReplicationDatabase01" />
        <property tag="DeviceSysName" value="ReplicationDatabase01" />
        <property tag="DatabaseName" value="ReplicationDatabase01" />
      </Event>
</Events>
```

universalAdd_FileSystem

This sample XML file helps you add a new file system CI. The structure of this sample file is as follows:

```
<Events>
      <Event>
        <property tag="eventType" value="AddCIEvent" />
        <property tag="MdrElemmentID" value="FileSystem" />
        <property tag="className" value="File" />
        <property tag="Description" value="Adding a new FileSystem" />
        <property tag="category" value="Exchange Services" />
        <property tag="Label" value="FileSystem" />
        <property tag="StoreName" value="FileSystem" />
        <property tag="FilePathUrl" value="file://c" />
      </Event>
</Events>
```

universalAdd_Memory

This sample XML file helps you add a new memory CI. The structure of this sample file is as follows:

```
<Events>
      <Event>
        <property tag="eventType" value="AddCIEvent" />
        <property tag="MdrElementID" value="Memory" />
        <property tag="className" value="Memory" />
        <property tag="DEscription" value="Adding a new Memory" />
        <property tag="category" value="Exchange Services" />
        <property tag="Label" value="Memory" />
        <property tag="MemoryType" value="Memory" />
        <property tag="OSNumeric" value="01" />
        <property tag="ContaingIndex" value="01" />
        <property tag="DeviceSysName" value="Memory" />
      </Event>
</Events>
```

universalAdd_Processor

This sample XML file helps you add a new processor CI. The structure of this sample file is as follows:

```
<Events>
        <Event>
          <property tag="eventType" value="AddCIEvent" />
          <property tag="MdrElementID" value="CPU" />
          <property tag="className" value="Processor" />
          <property tag="Description" value="Adding a new Processor" />
          <property tag="category" value="Exchange Services" />
          <property tag="Label" value="CPU" />
          <property tag="ProcessorType" value="CPU" />
          <property tag="OSNumeric" value="01" />
          <property tag="ContainingIndex" value="01" />
          <property tag="DeviceSysName" value="CPU" />
        </Event>
</Events>
```

universalAdd_Service

This sample XML file helps you add a new service. The structure of this sample file is as follows:

```
<Events>
        <Event>
          <property tag="eventType" value="AddCIEvent" />
          <property tag="MdrElementID" value="MyService" />
          <property tag="className" value="Service" />
          <property tag="Label" value="MyService" />
          <property tag="Description" value="Description of a new Service" />
          <property tag="AdministrativeStatus" value="Managed" />
          <property tag="ServiceName" value="MyService" />
          <property tag="ServiceVersion" value="1" />
        </Event>
</Events>
```

universalAdd_ServiceModel

This sample XML file helps you add a service model. The structure of this sample file is as follows:

```
<Services>
        <Service>
          <property tag="eventType" value="AddService" />
          <property tag="MdrElementID" value="UCService" />
          <property tag="className" value="Service" />
          <property tag="ServiceName" value="UCService" />
          <property tag="ServiceVersion" value="0" />
          <property tag="Description" value="Adding a new Service" />
          <property tag="AdministrativeStatus" value="Managed" />
        </Service>
        <CI>
          <property tag="eventType" value="AddCI" />
          <property tag="MdrElementID" value="UCServer" />
          <property tag="className" value="ComputerSystem" />
          <property tag="Label" value="UCServer" />
          <property tag="Description" value="Description of a new ComputerSystem" />
          <property tag="AdministrativeStatus" value="Managed" />
          <property tag="Vendor" value="Dell" />
          <property tag="PrimaryDnsName" value="UCServer.ca.com" />
          <property tag="ComputerName" value="UCServer" />
          <property tag="MemoryInGB" value="1024" />
          <property tag="CIuserAttribute1" value="Attr1" />
          <property tag="CIuserAttribute2" value="Attr2" />
          <property tag="CIuserAttribute3" value="Attr3" />
          <property tag="CIuserAttribute4" value="Attr4" />
          <property tag="CIuserAttribute5" value="Attr5" />
        </CI>
        <Relationship>
          <property tag="eventType" value="AddRelationship" />
          <property tag="SourceMdrElementID" value="UCService" />
          <property tag="TargetMdrElementID" value="UCServer" />
          <property tag="ScopeMdrElementID" value="UCService" />
          <property tag="Semantic" value="HasMember" />
          <property tag="className" value="BinaryRelationship" />
        </Relationship>
</Services>
```

universalDelete_ComputerSystem

This sample XML file helps you delete a computer system CI. The structure of this sample file is as follows:

```
<Events>
        <Event>
          <property tag="eventType" value="DeleteCIEvent" />
          <property tag="MdrElementID" value="MyMachine21" />
          <property tag="className" value="ComputerSystem" />
          <property tag="Label" value="MyMachine21" />
          <property tag="Description" value="Delete the ComputerSystem" />
          <property tag="AdministrativeStatus" value="Managed" />
          <property tag="Vendor" value="Dell" />
          <property tag="PrimaryDnsName" value="MyMachine21.ca.com" />
          <property tag="ComputerName" value="MyMachine21" />
          <property tag="MemoryInGB" value="1024" />
        </Event>
</Events>
```

universalUpdate_ComputerSystem

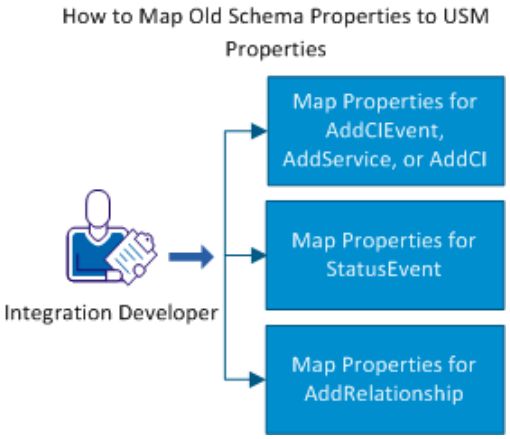
This sample XML file helps you update a computer system CI. The structure of this sample file is as follows:

```
<Events>
        <Event>
          <property tag="eventType" value="UpdateCIEvent" />
          <property tag="MdrElementID" value="MyMachine21" />
          <property tag="className" value="ComputerSystem" />
          <property tag="Label" value="MyMachine21" />
          <property tag="Description" value="Update to the Description" />
          <property tag="AdministrativeStatus" value="Managed" />
          <property tag="Vendor" value="Dell" />
          <property tag="PrimaryDnsName" value="MyMachine21.ca.com" />
          <property tag="ComputerName" value="MyMachine21" />
          <property tag="MemoryInGB" value="1024" />
        </Event>
</Events>
```

How to Map Old Schema Properties to USM Properties with Universal Connector

As an integration developer, you can map the old schema properties to USM properties to use the old XML files with the Universal connector. The Universal connector now supports only USM-based properties. XML files using the old schema properties, therefore, do not work with the Universal connector. If you want to use the old XML files, map the old schema properties to USM properties based on the eventType value.

Use the following scenario to guide you through the process:



You can do this mapping based on your specific requirements as follows:

- Map Properties for AddCIEvent, AddService, or AddCI (see page 91).
- Map Properties for StatusEvent (see page 92).
- Map Properties for AddRelationship (see page 92).

Map Properties for AddCIEvent, AddCI, or AddService

When you use AddCIEvent, AddCI, or AddService as the value for the eventType tag, you map the mandatory old properties to the corresponding USM properties as follows:

Old Properties	USM Properties
instanceID	MdrElementID
className	className (must represent a valid USM class)
situationMessage	Description
deviceID	Not Applicable
severity	Not Applicable

Map Properties for StatusEvent

When you use StatusEvent as the value for the eventType tag, you map the mandatory old properties to the corresponding USM properties as follows:

Old Properties	USM Properties
siloAlarmID	MdrElementID
instanceID	AlertedMdrElementID
className	className (must be Alert in this case)
situationMessage	Summary
severity	Severity (value represents Normal, Minor, Major, Critical, or Down)
situationType	AlertType
deviceID	Not Applicable

Map Properties for AddRelationship

When you use AddRelationship as the value for the eventType tag, you map the mandatory old properties to the corresponding USM properties as follows:

Old Properties	USM Properties
aNodeCI	SourceMdrElementID
bNodeCI	TargetMdrElementID
serviceCI	ScopeMdrElementID
associationType	Semantic
className	className (must be BinaryRelationship)

Universal Connector Programming Interface

You can access the Universal connector web client at the program level. This is especially useful if source control is available for your application and you want to provide a deeper level of integration with CA SOI. To successfully use the programming interface, you must first understand the Universal connector web service methods.

Universal Connector Web Service

The Universal connector provides a one-way interface using web services method calls that let CA Technologies and third-party products and clients send events to CA SOI. These events create new services that can include the following:

- CIs and relationships
- Create, update, and delete CIs
- Add infrastructure alerts using events

The Web Services Descriptor Language (WSDL) for the Universal connector web service is located at http://<SA Manager>:7090/axis2/services/GenericConnectorService?wsdl.

The Universal connector installation downloads all jar files into the SOI_HOME\lib\generic directory that are required to build Java-based web services clients. The file genericconnectorclient.jar contains all the web services stub classes that the methods implement.

The Java package that contains all of the web service Java class definitions is com.ca.sam.ifw.connector.generic.webservice.

Item Class

The Item class creates a Java object containing the following:

- Attributes of each event that you want to pass to the addEvents() method
- Attributes of each CI that you want to pass to the methods addCIs(), updateCIs(), or deleteCIs()
- Attributes of each service that you want to pass to the method addServices()
- Attributes of each service relationship object that you want to pass to the method addServiceRelationships()
- Attributes of each service CI object that you want to pass to the method addServiceCIs()

The Item class is as follows:

```
com.ca.sam.ifw.connector.generic.webservice.Item
public class Item
extends java.lang.Object
```

The constructor summary for the Item class is as follows:

```
Item()
```

The method summary for the Item class is as follows:

- String getClassName()
- void setClassName(String className)
- String getEventtype()
- void setEventtype(String eventtype)
- void setCiProperties(CIProperty[] ciProperties)
- CIProperty[] getCiProperties()

addEvents() Method

The addEvents() method adds an array of events to the web services queue. The Universal connector collects these events for publishing to CA SOI.

```
public boolean addEvents(Item[] statusEvents)
```

throws:

```
org.apache.axis2.AxisFault
```

updateCIs() Method

The updateCIs() method creates a list of CIs with updated attributes that are added to the web services queue. The Universal connector collects the CIs for publishing to CA SOI.

```
public boolean updateCIs(Item[] ciUpdates)
```

throws:

```
org.apache.axis2.AxisFault
```

addCIs() Method

The addCIs() method creates a list of new CIs that are added to the web services queue. The Universal connector collects these items for publishing to CA SOI.

```
public boolean addCIs(Item[] ciAdds)
```

throws:

```
org.apache.axis2.AxisFault
```

deleteCIs() Method

The deleteCIs() method creates a list of CIs that you want to delete from CA SOI and add to the web services queue. The Universal connector collects these items for publishing to CA SOI.

```
public boolean deleteCIs(Item[] ciDeletes)
```

throws:

```
org.apache.axis2.AxisFault
```

addServices() Method

The addServices() method creates a list of new services that are added to the web services queue. The Universal connector collects these items for publishing to CA SOI.

```
public boolean addServices(Item[] services)
```

throws:

```
org.apache.axis2.AxisFault
```

addServiceRelationships() Method

The addServiceRelationships() method creates a list of new service relationships that are added to the web services queue. The Universal connector collects these items for publishing to CA SOI.

```
public boolean addServiceRelationships(Item[] relationships)
```

throws:

```
org.apache.axis2.AxisFault
```

addServiceCIs() Method

The addServiceCIs() method creates a list of new service CIs that are added to the web services queue. The Universal connector collects these items for publishing to CA SOI.

```
public boolean addServiceCIs(Item[] cis)
```

throws:

```
org.apache.axis2.AxisFault
```

Troubleshooting the Universal Connector

This section provides information about how you can troubleshoot the Universal connector. You can verify whether your implementation of the Universal connector is working as expected.

Verify the Universal Connector Web Service

Use your web browser to verify whether the Universal connector web service on the CA SOI computer responds to queries. You can do this by verifying that the Web Services Description Language (WSDL) file content is displayed in an XML-like syntax and a list of available services for GenericConnectorService is displayed in a user-readable format.

- For WSDL, enter the URL *http://<SAManager_server>:7090/axis2/services/GenericConnectorService?wsdl* in your browser window. The web browser displays the WSDL file content in the appropriate XML format if the web service is responding correctly.
- For the list of available services, enter the URL *http://<SAManager_server>:7090/axis2/services/listServices* in your web browser window. The web browser displays the list of provided services for GenericConnectorService in a user-readable format if the web service is responding correctly.

Verify the Universal Connector Files

You can verify whether the appropriate Universal connector files are correctly installed. After the installation, you can search for a few files on the computer as follows:

- To verify the Universal connector files on the CA SOI computer where the Universal connector server is located, search for the following files:
  - SOI_Home\lib\ca-sam-connector-universal-server.jar
  - SOI_Home\lib\generic*.jar
  - SOI_Home\tomcat\webapps\axis2 folder
- To verify the Universal connector files on the Universal connector client computer, search for the following files:
  - SOI_Home\lib\ca-sam-connector-universal-client.jar
  - SOI_Home\GCEventAddCmd.bat
  - SOI_Home*.xml sample files

Chapter 7: Event Connector

This section includes information about the Event connector and provides installation and configuration procedures for the connector. Also, this section provides Event connector scenarios, troubleshooting, and uninstall procedures. Type and severity mapping for the Event connector are also included.

This section contains the following topics:

About the Event Connector

The Event connector uses CA Event Integration technology to integrate raw event sources with CA SOI. The connector collects events from the sources, transforms the events into the USM alert format, sends events to the local Event Store, and displays the events as infrastructure alerts in CA SOI. The connector creates CIs automatically for objects associated with the alerts such as servers, routers, and CPUs. You can create services from the created CIs to manage the event sources from a service-oriented perspective, or you can manage the alerts in defined queues.

The Event connector events use a version of the Event Store that is installed with CA Event Integration, controlled by the CA EI Event Management service. All events go into a CA Event Integration-specific Event Store. The stored events then become available for various Event Management operations. After you perform the Event Management operations, the processed events appear as actionable alerts in the Operations Console for further analysis.

Note: The Event Management functionality is not supported on Solaris and Linux operating platforms.

The Event connector can send events from the following sources to CA SOI:

Note: The connector does not support sources marked "Windows only" when installed on Solaris and Linux operating platforms.

- (Windows only) Application text log files
- (Windows only) CA NSM
- CA OPS/MVS Event Management and Automation
- CA SYSVIEW Performance Management
- HP Business Availability Center (HP BAC)
- SNMP traps
- Web services events
- (Windows only) Windows Event Log

Note: For more information about the specific events that the connector can collect from each source, see the topics specific to each source in Configure the Event Connector (see page 108). For more information about the Event connector, see the *CA Event Integration Product Guide*.

How to Install CA Event Integration and Event Connector

The Event connector uses CA Event Integration technology to enable integration with event sources. The connector relies on the CA Event Integration manager and requires a local or remote installation of the CA Event Integration manager. The connector also communicates with the SA Manager in CA SOI.

The Event connector supports installation on Windows, Solaris, and Linux systems. The CA Event Integration manager installation is supported only on Windows.

Note: For more information about specific platform support, see the *CA Event Integration Release Notes*.

By default, the Event connector is preconfigured to send the following events to CA SOI when installed on Windows:

- CA NSM agent messages from CA NSM Event
- Windows Event Log
- Application log file (local CA Event Integration Axis2 log by default)

The CA NSM and Windows Event Log sources must be local to the installation because the connector always collects events from the local Windows Event Log. If you want to collect CA NSM agent events, you must install the connector on a CA NSM server for the CA NSM event integration to work.

Note: You must customize the CA NSM source policy to handle non-agent events in the Event Console.

When installed on Solaris or Linux, no preconfiguration occurs, and you must manually configure the connector to collect events from specified sources and send them to CA SOI.

Installing the Event connector on a system that contains a previous version of CA Event Integration automatically upgrades the product. For more information, see Upgrade Existing CA Event Integration Connector (see page 120).

Note: Integration of CA Event Integration with Event Management is automatically enabled if you use the CA Event Integration Event Connector option (CA Event Integration installation image) during installation. This option is for installing the CA SOI-specific Event connector that automatically configures integration with CA SOI. However, if you install CA Event Integration separately from CA SOI using the CA Event Integration option, the integration is not enabled automatically. In this case, you must manually enable the integration (see page 107).

Install the CA Event Integration Manager

The Event connector requires the CA Event Integration manager component to function. Install this component in a separate operation before the Event connector. You can install the manager on the same system where you plan to install the Event connector or on a remote system.

Follow these steps:

1. Run the setupEI.hta file that is available on the CA Event Integration installation image in the package.

   The CA Event Integration setup page opens.

2. Click CA Event Integration.

   The Introduction page of the CA Event Integration installation wizard opens.

3. Click Next.

   The License Agreement page opens.

4. Scroll to the bottom of the agreement, select 'I accept the terms of the License Agreement', and click Next.

   The Choose Install Set page opens.

5. Select Manager and click Next.

   The Check License Key page opens.

6. Do one of the following:

   - Enter the license key required to enable integration with CA Spectrum and click Next.
   - Leave the field blank and click Next to install all standard functionality without CA Spectrum integration.

   The Choose Install Folder page opens.

7. Do *one* of the following to specify the installation folder and click Next:

   - Accept the default.
   - Enter the name of a new installation folder.
   - Click Browse and select an installation folder.

   The Enter Services User page opens.

8. Enter a user name and password for running the product's services and click Next. You can specify an existing local or domain administrator account. The user must be in the Administrators group. If you leave the fields blank, the installer creates a local ca_eis_user administrator account to run the services.

   The Enter Database Configuration - Step 1 page opens.

9. Do *one* of the following:

   - Click Next (without enabling trusted authentication) to use SQL Server credentials to create the manager database. This option requires you to enter an internal SQL Server database user and password on the next page.
   - Select the Enable check box and click Next to enable trusted authentication, which causes the installer to create the manager database using the installing user's credentials, as long as this user has the credentials necessary to create a database. After installation, CA Event Integration connects to the database using the services user's credentials.

   The Enter Database Configuration - Step 2 page opens.

10. Complete the following fields and click Next:

    Server name

    Specifies the server name, database instance, and port to use for the manager database. You can enter a node name or IP address for the database server. Use the following syntax to enter a server name, instance, and port in this field:

    dbserver\instance:port

You must enter a port number if you are using a named instance. You can omit the *instance* specification if you are not using a database instance or you want to use the default instance; and you can leave out the *:port* specification if you are using the default database port (1433) on the default instance.

User

(SQL authentication only) Specifies the user name of the user who will create the manager database and perform database operations. Enter an internal (mixed mode) SQL Server user with the authority to create a database.

Password

(SQL authentication only) Specifies the password associated with the user name defined in the User field.

The Configure Manager Application Server page opens.

11. Complete the following fields and click Next:

Note: To avoid conflicts, you should verify that no other applications are using the Tomcat ports. The installation program detects conflicts and alerts you to change the specified port if it is already in use.

Tomcat http port

Specifies the port Tomcat uses to connect to the administrative interface.

Default: 9091

Tomcat shutdown port

Specifies the port Tomcat uses to shut down.

Default: 8007

Tomcat AJP port

Specifies the port Tomcat uses to integrate with the Apache Web Server. Tomcat does not use this port, but requires it to be configured.

Default: 8011

Web UI application user

Specifies the user name for accessing the administrative interface.

Default: eiadmin

Web UI application password

Specifies the password to associate with the user specified in the Web UI application user field. Re-enter this password for verification in the Re-enter password field.

The Enter Communication Ports page opens.

12. Enter the port number to use for web services communications between the manager and connectors, and click Next. The default port number is 8083.

    The Configure Catalyst ActiveMQ Server page opens.

13. Select the Configure Catalyst ActiveMQ Server check box so that the manager can support an Event connector installation.

    The Enter Catalyst ActiveMQ server, port, and credentials page opens.

14. Complete the following fields to configure CA SOI integration:

    Server name

    Defines the server name where the CA Catalyst server is available.

    Port

    Defines the specified port number for the ActiveMQ Message Bus communication that you defined during CA Catalyst Server installation. Retain the default of 61616 unless you changed this port number when installing CA Catalyst.

    Default: 61616

    User

    Specifies the administrator user name that you defined during the CA Catalyst installation.

    Password

    Defines the password associated with the user.

    The Pre-Installation Summary page opens.

15. Verify the information about the Pre-Installation Summary page, and click Install.

    A page opens charting the installation progress. When installation is complete, the Install Complete page opens and summarizes the installation.

16. Click Done.

Install the Event Connector on Windows

When you install the Event connector, it is preconfigured to send events to CA SOI. An SA Manager must exist on the same or a different system. The Event connector also relies on and registers with the CA Event Integration manager. When you install the Event connector, you must enter connection information for the already installed CA Event Integration manager (see page 99). Multiple connectors can use the same CA Event Integration manager. The Event connector can be on a system that is local or remote to the CA Event Integration manager.

Follow these steps:

1. Double-click the setupEI.hta file that is available on the CA Event Integration installation image in the package.

   A setup page opens.

2. Click CA Event Integration Event Connector.

   The Introduction page of the Event connector installation wizard opens.

3. Click Next.

   The License Agreement page opens.

4. Scroll to the bottom of the agreement, accept the License Agreement, and click Next.

   The SOI Event Configuration page opens. This dialog specifies that the connector will be preconfigured to collect events from CA NSM (if CA NSM exists on the system), the Windows Event Log, and the CA Event Integration Axis2 application log file.

5. Click Continue.

   The Check License Key page opens.

6. Do *one* of the following and click Next:

   - Enter a license key to enable integration with CA Spectrum.
   - Leave the field blank to install all standard functionality without CA Spectrum integration.

   The Choose Install Folder page opens if you are installing the Event connector on a separate system from the CA Event Integration manager. If installing on the same system as the manager, skip to Step 10.

7. Do *one* of the following to specify the installation folder and click Next:

   - Accept the default.
   - Enter the name of a new installation folder.
   - Click Browse and select an installation folder.

   The Enter Services User page opens.

8. Enter a user name and password for running the product's services and click Next.

   You can specify an existing local or domain administrator account. The user must be in the Administrators group. If you leave the fields blank, the installer creates a local ca_eis_user administrator account to run the services.

   The Enter Communication Ports page opens.

9. Complete the following fields and click Next:

   Manager Host Name

   Specifies the host on which the CA Event Integration manager is installed. A connector must register with a manager so it can receive event integration and processing instructions and appear on the manager's administrative interface. You can use the node name or IP address as the value for this field.

   Manager port

   Specifies the port number for web services communications that you entered for the manager server.

   Default: 8083.

   Connector port

   Specifies the port number to use for web services communications on the connector server. This port is independent of the port that you specified for the manager; it can use the same or a different port.

   Default: 8083

   The Enter Catalyst ActiveMQ server, port, and credentials page opens.

10. Complete the following fields, and click Next:

    Server name

    Defines the server name where the CA Catalyst server is available.

    Port

    Defines the specified port number for the ActiveMQ Message Bus communication that you defined during CA Catalyst Server installation. Retain the default of 61616 unless you changed this port number when installing CA Catalyst.

    Default: 61616

    User

    Specifies the administrator user name that you defined during the CA Catalyst installation.

    Password

    Defines the password associated with the user.

    The Pre-Installation Summary page opens.

11. Verify the information on the Pre-Installation Summary page, Click Install.

A page opens charting the installation progress. When installation is complete, the Install Complete page opens and summarizes the installation.

12. Click Done.

Note: If the installation fails or does not produce the desired result, see Event Connector Troubleshooting (see page 119).

Install the Event Connector on Solaris or Linux

After you install the Event connector on a Solaris or Linux system, you can configure it to send events to CA SOI. An SA Manager and CA Event Integration manager must reside on a separate Windows system.

When you install the connector on a Solaris or Linux system, you must run the installation from an xterm.

Follow these steps:

1. Copy the appropriate .bin file (InstallEI.bin.Solaris or InstallEI.bin.Linux) from the CA Event Integration installation image to the Solaris or Linux system.
2. Verify that the file possesses the appropriate root ownership and execute permissions as follows:

   ```
   chown root ./InstallEI.bin.Linux
   chmod 555 ./InstallEI.bin.Linux
   ```

3. Start the installation from an xterm as follows:

   Linux

   ```
   ./InstallEI.bin.Linux
   ```

   Solaris

   ```
   ./InstallEI.bin.Solaris
   ```

   Preface the command with the appropriate file path for the installer location on the Solaris or Linux system.

   The introduction page of the CA Event Integration installation wizard displays.

4. Click Next.

   The License Agreement displays.

5. Scroll to the bottom of the agreement.
6. Select I accept the terms of the License Agreement and click Next.

   The Choose Install Set displays.

7. Leave the default selection of Connector and click Next.

   The Check License Key displays.

8. Do one of the following and click Next:
   - Enter a license key to enable integration with CA Spectrum.
   - Leave the field blank to install all standard functionality without CA Spectrum integration.

   The Choose Install Folder displays.

9. Retain the default or select a different folder and click Next.

   The Enter Services User displays.

10. Enter a user name for running the product services and click Next.

    You can specify an existing local or domain account. The specified user must be in the root group. If you leave the fields blank, the installer creates a local caeiuser administrator account to run the services.

    The Enter Communication Ports displays.

11. Complete the following fields and click Next:

    Manager Host Name

    Specifies the host on which a manager is installed.

    Manager port

    Specifies the port number for web services communications that you entered for the manager server.

    Default: 8083

    Connector port

    Specifies the port number to use for web services communications on the connector server. This port is independent of the port that you specified for the manager; it can use the same or a different port.

    Default: 8083

    The Pre-Installation Summary displays.

12. Verify the information on the Pre-Installation Summary page and click Install.

    A page displays to display installation progress. When installation is complete, the Install Complete displays and summarizes the installation.

    The Event connector is installed on Solaris or Linux.

Note: If the installation fails or does not produce the desired result, see Event Connector Troubleshooting (see page 119).

Enable CA Event Integration and Event Management Integration Manually

If you install CA Event Integration separately from CA SOI using the CA Event Integration option (CA Event Integration installation image), you must perform a manual configuration to enable integration between CA Event Integration and Event Management. This integration facilitates sending of collected events to the CA Event Integration-specific Event Store for use in Event Management searches and policies.

Note: You do not need to manually enable this integration if you install CA Event Integration as the Event connector using the CA Event Integration Event Connector option. In this case, the integration is enabled automatically.

Follow these steps:

1. Open the EI_HOME\Event\conf\eventManagerServerConfig.xml file on the CA Event Integration connector system, add values for the following properties, and save and close the file:

   name

   Defines the name of the CA Event Integration connector for display on the CA SOI interfaces. Format this property as the connector host name appended by '-EI' as in the following example: server1.ca.com-EI.

   host

   Defines the host name of the SA Manager, which contains the ActiveMQ Server that controls all messaging into CA SOI.

   protocol

   Defines the protocol used by the ActiveMQ Server, which is tcp by default.

   port

   Defines the ActiveMQ Server port, which is 61616 by default.

   user

   Defines the CA SOI administrator user name.

   password

   Defines the password for the CA SOI administrator user.

   The Event Management connection is configured.

2. Start the CA EI Event Management service.

   CA Event Integration connects to the Event Management component.

3. Redeploy all previously deployed catalogs that are dispatching events to CA SOI.

   Events are sent to the CA Event Integration-specific Event Store located at EI_HOME\Core\EventStore and are available for Event Management searches and policies.

4. (Optional) Repeat Steps 1-3 for additional CA Event Integration connector systems, if necessary.

Event Connector Import

The Event connector imports information from integrated sources as follows:

Automatic CI creation

Creates CIs in CA SOI automatically, based on the device address of collected events. Top-level CIs (such as device, server, and application) are created automatically and you can configure the connector to also create more granular CIs based on data in the collected events (file system, CPU, and so on).

For more information about Event connector configuration, see Configure Event Connector CI Creation and Association. (see page 116)

The connector does not import or create services. You must model services based on the created CIs manually.

Events and Alarms

Sends collected events to the CA Event Integration-specific Event Store and displays all collected, processed events as infrastructure alerts in the Operations Console. The CA Event Integration-specific Event Store is available on the computer where you install the Event connector.

Configure the Event Connector

If the Event connector installs correctly, it registers with the SA Manager and CA Event Integration manager. If you installed on a Windows system, a catalog named samevent is created and deployed in the CA Event Integration manager. By default, this catalog dispatches events from the following sources to CA SOI:

- Agent events from the local CA NSM Event installation (if it exists)
- Local Windows Event Log
- Application log file (CA Event Integration Axis2 log by default)

Each event source is controlled by a source policy file. While the default sources do not require policy configuration, other available sources do require you to configure policy by adding attributes that let the connector establish a connection with the event source. To customize the Event connector configuration, you may need to refine how events are collected from the default sources, add and configure additional sources, or remove the default sources.

Note: You must include the CIs created by the Event connector in services only if you want to see the alerts in the context of services. Event connector events appear in the Event Store and in alert queues even if they are not associated with services.

You can configure the sources that the Event connector collects from the CA Event Integration administrative interface.

Note: This procedure covers a full configuration scenario, which is required for a connector installed on a Solaris or Linux system. You may not need to perform all steps in this procedure if the Event connector on Windows preconfigured correctly or if you want to leave the default sources. No configuration is required if you installed on a Windows system and want to keep the default configuration.

Follow these steps:

1. (Optional) Complete any required configuration on the event sources.

   HP BAC, CA OPS/MVS EMA, and CA SYSVIEW PM require domain manager configuration before the connector can integrate with these sources.

   Note: For more information, see the topics specific to these sources.

2. Select Start, Programs, CA, Event Integration, Manager User Interface to launch CA Event Integration.

   The CA Event Integration administrative interface login page displays.

3. Enter the credentials you provided for the CA Event Integration interface user during installation, and click Log In.

   The CA Event Integration administrative interface displays with the Dashboard tab visible.

4. Select the Policies tab.

   The View Policies page displays.

5. Do the following for each source whose events you want to collect:

   a. Select the *source*-src.xml file.

   b. Configure the source policy attributes.

   Note: For more information about configuring policy for each source, see the source-specific topics that follow or the *CA Event Integration Online Help*.

6. Select the sampc-dest.xml file.

   The Policy Configuration: SAMAdapter page displays.

7. Verify that all connection settings are entered correctly, make any necessary changes to connect to the appropriate SA Manager, and click Save.

   Note: This file should be configured correctly if the preconfiguration was successful. Do not change the topicout field value. For more information about the reconcile_level field, see Configure Event Connector CI Creation and Association (see page 116).

   The connection to CA SOI is configured.

8. Select the Catalogs tab.

   The View Catalogs page displays.

9. Do one of the following:

   - If the samevent catalog exists, select it and click Edit.
   - If the samevent catalog does not exist, click New Catalog.

   The Edit Catalog or New Catalog page displays.

10. Edit the source policy included in the catalog so that all necessary sources are represented in the Selected Sources pane and click Next.

    The Select Destination Policies page displays.

11. Ensure that sampc-dest.xml is in the Selected Destinations pane and click Next.

    The Select Enrichment Policies page displays.

12. Click Next.

13. (Optional) Name the catalog and click Finish.

14. Select the Connectors tab and do one of the following:

    - If the samevent catalog was created by default, it should already be assigned to the connector that corresponds to the Event connector system. Select this connector and click Deploy.
    - If you created the catalog, select the connector that corresponds to the Event connector system and click Assign Catalog. Assign the created catalog to the connector and deploy the catalog after assignment.

      The Event connector starts collecting events from the configured sources and dispatching them to CA SOI. The connector creates CIs in CA SOI associated with collected events.

15. (Optional) Create services in CA SOI that contain the associated CIs.

    Note: You can search for CIs by source in the Service Modeler to view all CIs created by the Event connector.

    The events appear as infrastructure alerts associated with the appropriate CIs in the Operations Console.

CA NSM Events

The CA NSM connector sends DSM state change events and WorldView state changes to CA SOI as infrastructure alerts. However, these two alert sources do not encompass all of the data that you can manage with CA NSM Event Management. The Event connector collects all CA NSM agent event messages displayed in the CA NSM Event Console from a local CA NSM Event Manager or Event Agent. Integrating these events into CA SOI lets you manage all systems management agent events from a service or alert queue perspective in the CA SOI Operations Console. The Event connector also transforms these events into the USM alert format. This transformation makes it easier to manage the varied agent event types in the Event Console by creating a unified format in CA SOI.

You can also integrate CA NSM Event Management with CA SOI Event Management. This integration helps you address scenarios where you use CA NSM for event management and want to migrate to CA SOI for having a unified event management solution. In such cases, you can create a migration path using CA SOI Event Management along with the Event connector to integrate with CA NSM Event Management. With this integration, CA SOI functions as a single point of contact for such enterprise-level organizations. This comprehensive event management integration can provide administrators a clearer view of complete business impact when service problems occur.

Note: Using the CA Event Integration filter policy helps ensure that only the CA NSM agent events that you are interested in managing display in the Operations Console. For more information about filter policy, see the *Event and Alert Management Best Practices Guide*.

The Event connector requires a local CA NSM installation to work. A CA NSM Enterprise Management Event Agent or Event Manager must be present on the connector system. If the installation detects that CA NSM is not present on the system, it disables the default CA NSM source configuration. If you want to collect CA NSM events from multiple Event Managers or Event Agents, install an Event connector on each manager or agent.

Note: If you want to send events to a remote CA NSM server, you also can install a connector on a server with an Enterprise Management Administrative Client installation. For more information, see the *CA Event Integration Product Guide*.

No source policy configuration is required to begin collecting events from CA NSM. When CA NSM is present on the connector server, CA NSM event collection is enabled by default. However, customize the CA NSM source policy to collect nonagent events from the Event Manager or Event Agent. Nonagent events are filtered out in the default policy. For example, you can customize the policy to recognize events from other sources such as an integrated product, SAMPacks, or basic SNMP traps. For more information about policy customization, see the *CA Event Integration Product Guide*.

Windows Event Log Events

The Event connector can collect events from the Windows operating system event log, which also is called the Event Viewer. The Windows Event Viewer contains three separate event logs: System, Application, and Security. These logs report important Windows activity, such as service terminations, initialization problems, application failures, and authentication failures. The connector collects events from all three logs. Managing Windows Event Log events as a part of your CA SOI environment can help alert you to operating system faults such as authentication or service failures and more accurately represent impact to services caused by these types of faults.

The connector automatically collects events from the local Windows Event Log. If you want to manage multiple Windows Event Logs, install a connector on each Windows system. No policy configuration is required.

CA OPS/MVS EMA Alarms

The Event connector can collect CA OPS/MVS EMA alarms and dispatch them to CA SOI. CA OPS/MVS EMA is a mainframe event management tool that acts as a console for mainframe events and lets you write event automation rules.

When you dispatch CA OPS/MVS EMA alarms to CA SOI, the alarms appear as events in the Event Store and infrastructure alerts in the Operations Console. Each alert generates a Server class CI associated with the CA OPS/MVS EMA server and the entity on which the trap is reporting (batch job number, for example). Include the CIs in a service to monitor mainframe alarms in the context of services. You can add functionality such as escalation policy and enrichment to the alerts to enhance the provided information.

The connector collects CA OPS/MVS EMA alarms through SNMP traps and can be installed anywhere relative to the product. To enable SNMP traps to reach the Event connector, you must configure its server as a trap destination in CA OPS/MVS EMA.

After configuring the trap destination, you must configure the port information in the mfopsmvs-src.xml source policy from the CA Event Integration administrative interface, add the mfopsmvs-src.xml file to the samevent catalog, and redeploy the catalog for the connector to begin collecting CA OPS/MVS EMA alarms and dispatching them to CA SOI.

Important! The default CA OPS/MVS EMA source policy is a basic framework for collecting traps from the product. Edit the mfopsmvs-src.xml file directly to customize the policy to appropriately process the traps in your environment. For more information about policy customization, see the *CA Event Integration Product Guide*.

For a complete CA OPS/MVS EMA configuration scenario, see Event Connector Scenario: Integrating with CA OPS/MVS EMA (see page 117).

CA SYSVIEW PM Alerts

The Event connector can collect CA SYSVIEW PM alerts and dispatch them to CA SOI. CA SYSVIEW PM is a mainframe performance management tool that lets you monitor and manage mainframe performance metrics such as system activity, CPU usage, and transaction details. Managing CA SYSVIEW PM alerts in CA SOI lets you view how mainframe performance affects business services.

When you dispatch CA SYSVIEW PM alerts to CA SOI, the alarms appear as events in the Event Store and infrastructure alerts in the Operations Console. Each alert generates an Application class CI associated with the CA SYSVIEW PM server and the entity on which the trap is reporting. Include the CIs generated by CA SYSVIEW PM alerts in a service to monitor mainframe performance alerts in the context of services.

By default, the Event connector processes CA SYSVIEW PM alerts originating from the following sources:

- MVS
- CICS
- IBM Websphere MQSeries
- IMS
- TCP/IP

These sources report on batch jobs and mainframe performance.

The connector collects CA SYSVIEW PM alerts through SNMP traps and can be installed anywhere relative to the product. To enable SNMP traps to reach the Event connector, you must configure its server as a trap destination in CA SYSVIEW PM.

After configuring the trap destination, configure the port information in the mfopsysview-src.xml source policy from the CA Event Integration administrative interface, add the mfsysview-src.xml file to the samevent catalog, and redeploy the catalog for the connector to begin collecting CA SYSVIEW PM alerts and dispatching them to CA SOI.

Important! The default CA SYSVIEW PM source policy is a basic framework for collecting traps from the product. Edit the mfsysview-src.xml file directly to customize the policy to suit your environment. For more information about policy customization, see the *CA Event Integration Product Guide*.

HP BAC Alerts

The Event connector can collect HP Business Availability Center (HP BAC) alerts and dispatch them to CA SOI. HP BAC is an application performance management tool that lets you monitor and manage end-to-end application performance using script-driven transactions, which produce alerts.

Managing HP BAC alerts in CA SOI lets you view how application performance alerts affect business services. If you also have CA APM products integrated into CA SOI, integrating HP BAC information provides a consolidated interface for managing all application performance information.

When you dispatch HP BAC alerts to CA SOI, they appear as events in the Event Store and infrastructure alerts in the Operations Console. Each alert generates an Application class CI associated with the HP BAC application on which the alert is reporting. Include these CIs in a service to monitor the alerts in the context of services.

The connector collects HP BAC alerts through SNMP traps and can be installed anywhere relative to the product. To enable SNMP traps to reach the Event connector, you must configure its server as a trap destination in HP BAC.

After configuring the trap destination, configure the port information in the hpbac-src.xml source policy from the CA Event Integration administrative interface. Then, you add the hpbac-src.xml file to the samevent catalog and redeploy the catalog for the connector to begin collecting HP BAC alerts and dispatching them to CA SOI.

Important! The default HP BAC source policy is a basic framework for collecting traps from the product. Edit the hpbac-src.xml file directly to customize the policy to suit your environment. For more information about policy customization, see the *CA Event Integration Product Guide*.

SNMP Traps

The Event connector can collect SNMP traps sent to the connector system. Where you install the connector does not matter, as long as you configure SNMP traps sources to send their traps to the connector system.

Note: SNMPv3 is not supported.

Before you begin collecting traps, the SNMP policy requires you to configure the port number in the administrative interface to receive all traps on the configured port.

For additional configuration, you must edit the SNMP source policy file directly to customize the policy to suit your environment. For more information about policy customization, see the *CA Event Integration Product Guide*.

After customization, you must add the snmp-src.xml file to the samevent catalog and redeploy the catalog for the connector to begin collecting traps and dispatching them to CA SOI.

Important! You must use the CA Catalyst connector for SNMP traps rather than the SNMP integration through the Event connector in most cases when you want to develop a custom trap-based Level 2 integration. In cases where you have written custom policy for SNMP trap collection from the Event connector, consider rewriting that policy for use with the SNMP connector. For more information about writing custom policy for the SNMP connector, see the *SNMP Connector Guide*.

Application Log File Events

The Event connector can collect event data from any generic text log file. Managing log files through CA SOI lets you monitor application logs for failure events that may contribute to service degradation. For example, you can configure the application log policy to monitor an important network activity log file for specific failures so that you can include these events in service impact analysis.

The connector can read any text log file on its system. You can monitor one log file at a time.

By default, the Event connector is configured to monitor the CA Event Integration Axis2 log file. Configure the source policy to monitor a different log file. You can specify the log file name and path in the applog-src.xml file from the CA Event Integration administrative interface. However, you must also customize the policy file directly to match the format of the log file to monitor and specify the type of information to collect from the file. For more information about policy customization, see the *CA Event Integration Product Guide*.

After customizing the policy file, deploy the catalog again for the connector to begin monitoring the log file according to the customized policy.

Web Services Events

Web services eventing is a way of sending notifications through web services. A web service can establish a subscription to another web service to receive events published by that service. The Event connector can collect notifications sent through web services according to the web services eventing standard.

Use the wsevent-src.xml source policy to configure from where the web services eventing adaptor collects web service notifications and how to process these events. This policy file requires you to define the following information in the administrative interface:

- Web services publisher from which to collect notifications
- Event properties such as prefix and namespace

When you deploy a catalog with web services eventing source policy, the connector collects events that the defined publisher outputs. You may want to collect web services events from important managed services and applications if these entities are part of a critical business service.

By default, the source policy is customized to process web service notifications from Microsoft Live Meeting servers. Customize the policy to process web services events from other sources. For more information about customizing policy, see the *CA Event Integration Product Guide*.

Add the wsevent-src.xml file to the samevent catalog and redeploy the catalog for the connector to begin collecting web services events and dispatching them to CA SOI.

Configure Event Connector CI Creation and Association

The Event connector creates CIs based on alert information that you can add to service models, or associates the alert with an appropriate existing CI. You can control the granularity of this CI creation and association in the connector to match the granularity of your modeled services and ensure that all alerts from the connector are considered appropriately in service impact analysis. The Event connector can create CIs or associate alerts with existing CIs at *one* of the following levels:

Device

Creates CIs based on the alert's top-level device. For example, if an alert is associated with a specific process or memory usage, the connector creates an associated CI for the server that contains the process or memory. This is the default CI granularity.

Subdevice

Creates CIs based on the specific objects with which alerts are associated. This more granular method creates CIs based on the specific class association of the alert's subdevice. For example, the connector would create a CI for a CPU for an alert associated with high CPU usage.

Properly configuring the CI granularity level is vital to ensure that Event connector alerts are accurately evaluated in CA SOI. For example, if your service models in CA SOI contain only high-level CIs such as servers and network devices and the Event connector is configured to associate alerts at the subdevice level, alerts for specific CIs (such as services or users) within those high-level CIs will be associated with the specific CIs and not be considered as impacting the service. Conversely, if your services are granularly modeled in CA SOI but the Event connector is configured to associate alerts at the device level, alerts associated with specific CIs will be associated with device-level CIs, making it more difficult to discover the root cause of service degradation.

Follow these steps:

1. Access the CA Event Integration administrative interface and click the Policies tab.

   The View Policies page displays.

2. Select sampc-dest.xml.

   The Policy Configuration: SAMAdapter page displays.

3. Set the reconcile_level field to the appropriate granularity level: device or subdevice. By default, the granularity level is device.

   Click Save.

   The policy is configured.

4. Redeploy any currently deployed catalog with this policy assigned.

   The change takes effect.

Event Connector Scenario: Integrating with CA OPS/MVS EMA

The Event connector can integrate alarms from CA OPS/MVS EMA to expose mainframe data for management in CA SOI.

When you dispatch alarms to CA SOI, you can model services with the CA OPS/MVS EMA Server class CIs created by the connector to view mainframe alarms from the Operations Console and enable functionality such as escalation policy and enrichment for these alarms.

In this scenario, you configure CA OPS/MVS EMA alarm collection through SNMP traps and dispatch the alarms to CA SOI as infrastructure alerts.

Note: This scenario assumes that the Event connector was installed successfully with all preconfiguration complete.

Follow these steps:

1. Configure a trap destination in CA OPS/MVS EMA for the Event connector server.
2. Access the CA Event Integration administrative interface.

   The CA Event Integration administrative interface displays. CA OPS/MVS EMA source policy requires a basic configuration before you can use it in a catalog.
3. Select View Policies.

   The View Policies displays.
4. Select mfopsmvs-src.xml.

   The Policy Configuration: SAMAdapter displays.
5. Enter the port number you want to use for receiving traps in the port field, then click Save.

   The CA OPS/MVS EMA source policy is configured. The SNMP source sends all traps to the CA Event Integration host on this port.
6. Select the Catalogs tab, select samevent, and click Edit.

   The Edit Catalog displays.
7. Add mfopsmvs-src.xml to the Selected Sources pane on the Select Source Policies page and click Next.

   The Select Destination Policies displays.
8. Verify that the Selected Destination Policies pane contains sampc-dest.xml and click Next.

   The Select Enrichment Policies displays.
9. Click Finish.

   The catalog is saved and the View Catalogs page displays again.
10. Select the Connectors tab.

    The View Connectors displays. The samevent catalog should already be assigned to the Event connector.

11. Select the connector corresponding to the Event connector system and click Deploy.

    The catalog deploys and the connector begins collecting CA OPS/MVS EMA alarms and dispatching them to CA SOI.

    Generated alarms publish as SNMP traps, where the connector collects, processes, and publishes them to CA SOI as infrastructure alerts.

12. Model a service in CA SOI with the Server class CIs created by the CA OPS/MVS EMA alarms.

    The alarms display as infrastructure alerts in the Alerts tab of the Operations Console for the associated CIs.

Event Connector Troubleshooting

If you notice problems with the Event connector after installation, use the following to troubleshoot the connector:

- Review the CA_Event_Integration_InstallLog.log file located in the root of the installation directory. This log primarily shows items such as installation errors related to file copies and configuration settings.
- Review the install.log file located in the EI_HOME\Logs directory. This log shows pre-installation or post-installation errors such as database creation failure and connector registration problems.
- Open the CA Event Integration administrative interface and verify that a connector displays in the Connectors tab for the Event connector system. If the connector does not display in the interface, registration with the CA Event Integration manager failed. To manually register the connector with the agent, run the following command from EI_HOME\Core\bin on the connector system:

  `register_agent` *`EI_Manager_host`* `8083` *`connector_host`* `8083`

- Verify that the samevent catalog was created and assigned to the connector. Check the catalog contents to verify the correct contents. Also check the sampc-dest.xml file to verify that the CA SOI connection settings are accurate.
- Ensure that you include the CIs created by the Event connector in services only if you want to see the alerts in the context of services.

Upgrade Existing Event Connector or CA Event Integration Installation

The Event connector relies on the CA Event Integration manager component, which was previously installed as part of event enrichment. If the Event connector exists on a separate system from the manager, first upgrade the manager using the instructions in the *CA Event Integration Product Guide*.

When you install the Event connector on a system that already contains an Event connector, event enrichment and Event connector, or connector-only standalone installation of CA Event Integration, the Previous Install Detected dialog opens after you accept the license agreement. After you click Continue on this dialog and begin the installation, the connector automatically upgrades the existing installation. All pre-existing policies and catalogs are maintained. For standalone CA Event Integration installations, the upgrade only works if the previous standalone version of CA Event Integration contains only a connector. Upgrading to the Event connector from a manager or manager and connector standalone CA Event Integration installation is not supported.

Note: Manual configuration can be required to enable the connector and begin dispatching events to CA SOI after an upgrade. For more information, see Configure the Event Connector (see page 108).

If you install the Event connector on the same system as the event enrichment feature installed with a previous release, the installation is upgraded without the event enrichment feature. The Mid-tier plugin provides the enrichment functionality. Therefore, remove the catalog previously used for event enrichment, or edit the catalog and remove the CA SOI source policy file if you are also using the catalog for the Event connector.

Uninstall the Event Connector on Windows

You can uninstall the Event connector when it is no longer required. To uninstall the Event connector, uninstall CA Event Integration from the Event connector server.

Note: If the Event connector and CA Event Integration manager are installed on separate systems, uninstall the connector and manager separately.

Follow these steps:

1. Select Start, Programs, CA, Event Integration, Uninstall CA Event Integration on the Event connector system.

   The Uninstall CA Event Integration page opens.

2. Do one of the following:
   - Click Next for an uninstallation that includes the manager.

     The Delete Database page opens.
   - Click Uninstall.

     The uninstallation initializes, runs, and completes. A page displays summarizing the uninstallation. This page may list files or directories that were not removed.
3. (Manager uninstallation only) Select the Remove database check box if you want to delete the database and click Uninstall.
4. Click OK.

   When the dialog closes, the uninstallation program performs further cleanup. Check the directories listed in the summary to verify that they were deleted.

Clean Up Windows User Information

When you uninstall the Event connector on Windows, the following directories and registry entry related to the operating system user created with the product (ca_eis_user by default) may persist:

- C:\Documents and Settings*username* (Windows 2003)
- C:\Users*username* (Windows 2008)
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileLists\S-1-5-21-*

  Note: The full name of the registry entry varies.

If you uninstall the Event connector and do not plan to reinstall, you should remove these persistent files.

Follow these steps:

1. Delete the C:\Documents and Settings*username* (Windows 2003) or C:\Users*username* (Windows 2008) directory, if it still exists.
2. Run the delprof.exe utility provided with Windows to delete all other materials, including the registry entry, associated with the user profile (and other inactive user profiles).

   You can download this utility from the Microsoft website if it is not on your system.

Uninstall the Event Connector on Solaris or Linux

You can uninstall an Event connector installed on a Solaris or Linux system. You must run the uninstallation from an xterm.

Follow these steps:

1. Start the uninstallation from an xterm with the following command:

   ```
   'EI_HOME/Uninstall_CA Event Integration/Uninstall_CA_Event_Integration'
   ```

   Replace EI_HOME with the path to the installation directory. For example, if you are using the default installation directory, the command would resemble the following:

   ```
   'opt/CA/EventIntegration/Uninstall_CAEventIntegration/Uninstall_CA_Event_Integration'
   ```

   The Uninstall CA Event Integration dialog opens.

2. Click Uninstall.

   The uninstallation runs. A page displays summarizing the uninstallation. This page may list files or directories that were not removed.

3. Click OK.

   When the dialog closes, the uninstallation program performs further cleanup. Check the directories listed in the summary to verify that they were deleted.

Event Connector Type and Severity Mapping

Connectors normalize the types and severities in the domain manager application to USM format when importing services and CIs from domain managers.

When the Event connector collects alerts from event sources, it normalizes class, severity, and all values into common event property values. The Event connector then transforms these values into USM types and severities.

Note: For information about adding a type mapping and how to create a connector policy, see the *CA Catalyst Implementation Guide.*

Type Mapping

The Event connector internal_resourceclass property values correspond to USM types. The USM type assigned to each CI depends on the granularity of CI creation and association (see page 116) that you perform from the Event connector (device or subdevice).

Severity Mapping

The internal_newseverity event property performs Event connector severity mapping from properties assigned to each collected event during processing. The internal_newseverity property is derived by mapping the severity from the event source to numeric values.

The following shows the mapping of the internal_newseverity event property to USM severities in the output alerts:

internal_newseverity Event Tag	USM Severity
10	Unknown
30	Normal
50	Minor
70	Critical
90	Fatal
Other	Unknown

Chapter 8: Building a Custom Connector Using the Sample Connector

This section describes how to install and run the Sample connector and build a custom connector using the connector framework that the Sample connector provides.

This section contains the following topics:

Sample Connector

The Sample connector is a fully functional connector that creates sample CIs, alerts, and relationships in CA SOI to demonstrate and test connector functionality. The Sample connector uses a mock domain manager based on sample data files, which you can easily manipulate to change the connector's behavior. It also provides the materials to build a custom Level 4, 5, or 6 connector (see page 59), including Java code, connector policy, configuration, installer kit, and test data.

Running the Sample connector or using the Sample connector to create a connector helps you understand how a custom connector works and how the methods and policy are defined.

How the Sample Connector Works

The Sample connector uses a mock domain manager interface to retrieve sample data and demonstrate connector functionality. The connector gets the data to create CIs, relationships, and alerts based on XML files in the SOI_HOME\resources\SampleConnector\data directory. The data in the files is represented in both the USM format and a Sample connector format that simulates a domain manager format.

The following files supply data to the Sample connector through the mock domain manager:

sample-cis.xml

Contains sample CIs and relationships that are imported when the connector starts. The sample data demonstrates the following CI types:

- ComputerSystem
- Router
- InterfaceCard
- Processor
- File
- BackgroundProcess
- DatabaseInstance
- BinaryRelationship
- Service

sample-alerts.xml

Contains sample open alerts that are loaded at startup.

sample-ci-changes.xml

Contains sample data to simulate changes in the Sample connector domain manager such as create, update, and delete operations. Any changes recorded in this file apply at the connector startup and occur while the connector is running.

sample-metrics.xml

Contains sample metric data.

The Sample connector contains policy files that transform Sample connector data to the USM format for transmittal to CA SOI and transform USM data to the Sample connector format for invoking inbound operations on the mock domain manager data.

This provided Sample connector functionality lets you view sample data in CA SOI, simulate CI changes, and test inbound operations on other connectors.

Use Cases

You can use the Sample connector in *any* of the following ways:

To verify a CA SOI installation

The Sample connector can help you verify the correct operation of the CA SOI IFW to ensure that the product is running connectors and displaying connector data.

For more information, see Running the Sample Connector (see page 127).

As a base for creating a custom connector

The Sample connector contains the materials necessary to build a custom Level 4, 5, or 6 connector (see page 59). It demonstrates the implementation of all required Java interfaces and the connector policy. It also demonstrates how to use the test harness provided with the system so that you can verify the connector's functionality before deploying the connector to CA SOI. The Sample connector implementation works with mock data that simulates a domain manager. You can use this interface for testing or when the domain manager data is not easily available.

For more information, see Building a Custom Connector (see page 135).

To test advanced custom connector functionality

The Sample connector can serve as a trigger to test the inbound operations that enable a custom connector to create, update, and delete data in its domain manager based on CI changes in CA SOI. This testing technique forces a change to an instance of a USM type managed by the Sample connector. The CA Catalyst Synchronizer then sends that change to all connectors enabled for the USM type. If a custom connector supports inbound operations for the USM type, the connector's create, update, or delete method is called and tested.

For more information, see Update Sample Connector Data (see page 132).

How to Install and Run the Sample Connector

The Sample connector uses test data and a mock domain manager to simulate data retrieval. The connector interacts with the IFW and the CA Catalyst Synchronizer components. You can install the connector to integrate sample data into CA SOI, test CA SOI functionality, and illustrate how a connector operates.

Install the Sample Connector

Install the Sample connector on any supported operating system using the installer provided with CA SOI. The connector must be able to connect to an SA Manager system, but does not have to be on the same system as the SA Manager.

Note: If you want to build a custom connector (see page 135) using the Sample connector framework, install the Sample connector on the SA Manager.

Follow these steps:

1. Run the Connector_Sample.exe file at the root of the CA SOI installation image and click Next.
2. Accept the license agreement, then accept the third-party license agreements, and click Next.

   The Choose Install Folder screen opens.

   Note: This page does not open if CA SOI components already reside on the system.
3. Accept, enter, or choose the installation folder.

   Note: The maximum installation path length is 150 characters. The installer blocks paths with more than 150 characters.

   If you are not installing on the SA Manager server, the Integration Services Configuration page opens; otherwise, skip to Step 5.
4. Specify the required information to connect to the appropriate SA Manager and configure connector preferences.

   Note: Refer to your Installation Worksheet in the SA Manager section for these values. For more information about the Installation Worksheet, see the *Implementation Guide*.
5. Select whether to start the CA SOI services after installation and click Next.
6. Click Install.

   The connector installs on the system and integrates with the appropriate CA SOI instance. An installation summary page opens when the installation finishes.
7. (Optional) Review the installation log file (Sample_Install_*releasenumber*) that is in the SOI_HOME\log folder to check for installation errors.

Installed Components

The Sample connector installs the following components:

Note: The string *SOI_HOME* refers to the CA SOI installation directory. The default is C:\Program Files (x86)\CA\SOI. If you install the Sample connector on a machine that already contains CA SOI components, the connector is installed into the existing directory.

SOI_HOME\lib\sample.catalyst.connector.jar

Contains the code that runs the Sample connector in CA SOI.

SOI_HOME\resources\Configurations

sampleConnector_*connectorserver*.xml

Contains the following Sample connector configuration information for use by the SA Manager:

- Java class that implements the connector
- Connector policy files
- Indication if the connector is enabled

The file also contains configuration information used by the Sample connector, such as the names of the provided sample data files and connection information for the mock domain manager.

SOI_HOME\resources\Core\Catalogpolicy\sampleconnector_policy.xml and sampleconnector_policySB.xml

Contains the policy for transforming Sample connector data to and from the USM format. The sampleconnector_policy.xml file transforms outbound data from the format produced by the Java code to the USM format. The sampleconnector_policySB.xml file transforms inbound data from the USM format to the format consumed by the connector Java code.

SOI_HOME\resources\SampleConnector\data

Contains sample data files used by the Sample connector.

SOI_HOME\SampleConnector

Contains the files that implement the Sample connector. The SampleConnector folder contains the Eclipse project files (.project and .classpath) and the following subfolders:

src\java

Contains the Java source files.

src\resources

Contains same sample data and policy files included in the SOI_HOME\resources directory.

test\java

Contains SampleConnectorUTest, the main class for testing the connector using the test harness (see page 167).

test\resources

Contains data files used by the Sample connector when running with the test harness.

lib

Contains libraries required to run connectors with the test harness that are not required to run connectors with CA SOI.

docs

Contains the Javadoc generated from the Sample connector Java source.

InstallKit

Contains template files for creating a connector configuration file (see page 160), connector policy files (see page 154), and a connector installer (see page 174).

Important! If you make changes to the sample data or policy files and want to view the effect on a Sample connector running with CA SOI, you must make the changes in the SOI_HOME\resources\SampleConnector\data directory and restart the CA SAM Integration Services service (you do not have to restart the service if changing the sample-ci-changes.xml file). Changing the files in SOI_HOME\SampleConnector has no effect on a Sample connector running with CA SOI. However, when running the Sample connector with the test harness (see page 167), the files in SOI_HOME\SampleConnector are used.

Enable the Sample Connector

When you install the Sample connector, it is disabled by default. Enable the connector in its connector configuration file to run the connector in CA SOI.

Follow these steps:

1. Open the sampleConnector_*connectorserver*.xml file located at SOI_HOME\resources\Configurations on the connector system.
2. Set the State attribute in the element '<Silo>' to "Enabled".
3. Save and close the file.
4. Restart the CA SAM Integration Services service.

Verify that the Sample Connector is Running

You verify that the Sample connector is running and test its basic functionality to ensure that it is working correctly.

Follow these steps:

1. Open the Dashboard.
2. Select the Administration tab.
3. Expand Connector Configuration and the server where the Sample connector resides.
4. Select the CA:09998_*domainserver@connectorserver* entry.

   Note: Because the Sample connector works with an embedded mock domain manager, the domain manager server is always the same as the connector server.

   Information about the Sample Connector displays in the right pane. The connector displays as Online after it finishes initializing.

If the Sample connector entry does not appear, ensure that you have enabled the connector (see page 130) and restarted the CA SAM Integration Services service.

Verify Service, CI, and Alert Creation

You verify that the Sample connector can create services, CIs, and alerts by importing a provided sample service into the Operations Console.

Follow these steps:

1. Open the Operations Console.
2. Select Tools, Import Services.

   The Configure Data Sources dialog opens.
3. Select the CA:09998_*domainserverr@connectorserver* entry and click Import.

   The Import Services dialog opens.
4. Select SampleService in the left pane, click the right arrow to move SampleService to the right pane, and click OK.

   The service imports. The service displays as SampleService in the Operations Console and should contain CIs and alerts.

   The service, CIs, and alerts are verified.

Update Sample Connector Data

You can update or delete CIs, alerts, and services created by the Sample connector or create new ones by modifying the sample-ci-changes.xml file. You can view the changes to these objects in CA SOI. In a custom connector, you perform this operation by creating, updating, or deleting data in the source domain manager.

You use this functionality to test inbound create, update, and delete operations in other connectors including a custom connector. For example, when you create a CI in the Sample connector, the CA Catalyst Synchronizer sends that creation to all connectors enabled for the USM type that was created.

Follow these steps:

1. Save a backup copy of the SOI_HOME\resources\SampleConnector\data\sample-ci-changes.xml file and open the file.

   Note: Make sure that you are accessing this file from the correct location. Changing the files in the SOI_HOME\SampleConnector directory has no effect on a running Sample connector.

   The file lists several predefined change operations.

2. Modify the first <event> element in the file as follows:

   - Modify the action and entitytype attributes as follows to add a new operation that adheres to the established schema in the file:

     ```
     <event xsi:type="usm:SiloDataChange" action="" entityType="">
     ```

     action

     Defines the type of action to perform. Valid values are create, update, and delete.

     entitytype

     Defines the type of entity to create, update, or delete. Valid values are Relationship, Alert, and Item. Use the same value that you used in this attribute in the nested element <property_name="entitytype" />.

   - Modify the <property> elements nested in the <silodata> element. Each <property> element defines one property name and value. The properties can be USM properties or a more compact set of Sample connector properties. Define the USM format by setting the value of the eventtype property to USM-Entity as follows:

     ```
     <property name="eventtype" value="USM-Entity" />
     ```

     Omit this property to use the Sample connector properties.

- Specify the type of entity in the entitytype property. Valid values are Item, Alert, and Relationship. This value should match the value of the entitytype attribute in the <event> element.

  Example: <property name="entitytype" value="Item" />

- Define the object type in the class property for the Sample connector format or the ClassName property for the USM format as follows:

  Example: <property name="class" value="Router" />

  Example: <property name="ClassName" value="Service" />

  Examples of USM and Sample connector formats are included in the file.

  Note: The Sample connector supports only a subset of the USM object types.

- When the action is update or delete, you must specify the correct value for id (Sample connector format) or MdrElementID (USM format). If you are referring to a CI created in a prior update to sample-ci-changes.xml, use the value of the generated ID. Find the value of MdrElementID for existing objects by examining the USM properties for the CI in the USM Web View.

  Note: For more information about the USM Web View, see the *Administration Guide*.

  When the action is create, the value of id or MdrElementID is ignored, because the item does not yet exist in CA SOI. The Sample connector assigns a value automatically.

3. Save and close the file.

   The connector detects that the file has changed and sends change events to CA SOI. Verify updates by checking the CIs in the Operations Console.

   The CA Catalyst Synchronizer (if enabled) sends change events back to the configured connectors that have inbound operations enabled. Sample connector data is updated.

   Note: The CA Catalyst Synchronizer is disabled by default. Synchronization is only supported for specific use cases. For more information about enabling the Synchronizer and supported use cases, see the *Administration Guide*.

Example: Create a new Router CI in Sample connector format

The following example creates a new Router class CI in Sample connector format.

```
<event xsi:type="usm:SiloDataChange" action="create" entitytype="Item">
    <silodata xsi:type="usm:SiloData" entitytype="Item">
        <properties>
            <property name="entitytype" value="Item" />
            <property name="id" value="1" />
            <property name="name" value="testrouter" />
```

```
            <property name="ip_address" value="13.1.1.1" />
            <property name="class" value="Router" />
            <property name="description" value="Cisco Router" />
            <property name="sysname" value="testrouter" />
            <property name="dnsname" value="testrouter.ca.com" />
        </properties>
    </silodata>
</event>
```

The create action specifies to create a new entity. The absence of the property eventtype=USMEntity indicates that the properties are in the Sample connector format. The class of Router indicates the entity type. Although an id value is provided, this value is ignored, and the Sample connector automatically assigns a unique value.

Example: Update an existing Alert in USM format

The following example updates an alert that already exists in the Operations Console in USM format:

```
<event xsi:type="usm:SiloDataChange" action="update">
    <silodata entitytype="Alert">
        <properties>
            <property name="eventtype" value="USM-Entity" />
            <property name="entitytype" value="Alert" />
            <property name="ClassName" value="Alert" />
            <property name="MdrElementID" value="1008" />
             <property name="AlertedMdrElementID" value="100" />
            <property name="AlertedMdrProduct" value="CA:09998" />
            <property name="UrlParams" value="http://localhost?id=8" />
            <property name="Message" value="Service is stopped - Again" />
            <property name="Summary" value="Service is stopped - Again" />
            <property name="Severity" value="Critical" />
            <property name="AlertType" value="Quality" />
            <property name="OccurrenceTimestamp"
            value="0001-05-01T00:00:00-00:00" />
            <property name="ReportTimestamp"
            value="0001-07-01T00:00:00-00:00" />
        </properties>
    </silodata>
</event>
```

The action of update specifies to update an existing object, the eventtype of USM-Entity indicates the USM format, and the entitytype of Alert indicates that the object is an existing alert. The connector uses the value of the AlertedMdrElementID property to match the information with an existing object in CA SOI.

How to Build a Custom Connector

As an integration developer, you can build a custom connector using the Sample connector as a template. Using the Sample connector as a template is the recommended method for developing a custom connector.

Complete the following process to build a custom connector based on the Sample connector:

1. Initialize an Eclipse project for the connector (see page 137).
2. Implement the required interfaces and methods (see page 141).
3. Create the connector configuration file (see page 160).
4. Create connector policy (see page 154).
5. Test the connector (see page 167).
6. Deploy the connector (see page 173).

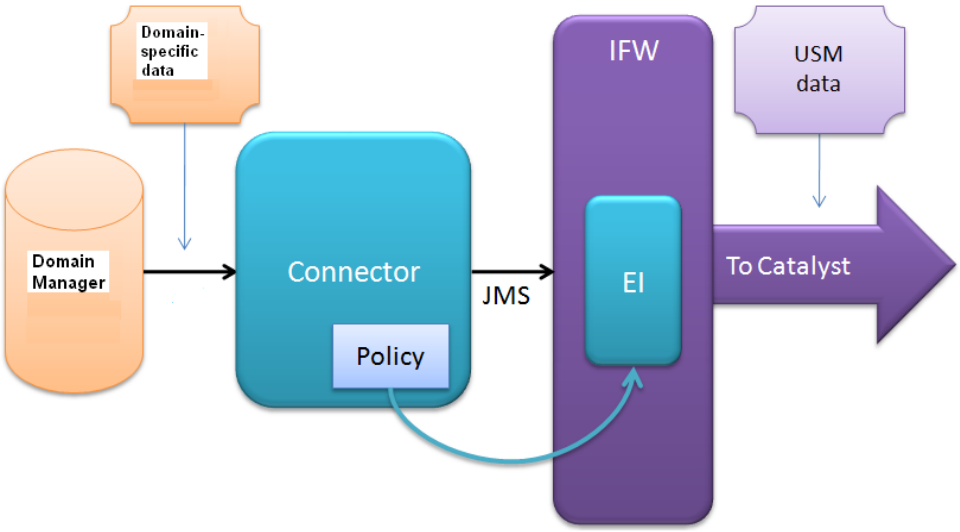
Connector Writing Basics

You can write custom connectors to create Level 4, 5, or 6 connector integrations (see page 59) for any domain manager product. Connectors require the following main components:

- Java code to implement the required interfaces for the connector
- Methods to retrieve raw data from a domain manager
- Connector policy that specifies how to transform the outbound raw domain manager data into a standard USM format
- Optional methods and policy that allow for inbound create, update, and delete operations in the domain manager

Connectors retrieve raw domain manager data such as CIs, alerts, and relationships as property-value pairs and normalize properties and values so that they adhere to the USM schema, which is the internal schema of the CA Catalyst framework and SA Manager.

The following illustration shows the process:



1. The connector methods interface with the domain manager and retrieve entities in the format of the domain manager.
2. The connector sends domain-specific data to the IFW through JMS.
3. The IFW processing module (pictured above as EI) uses connector policy to transform the entities to the USM format if necessary.
4. The IFW transmits the USM entities through the ActiveMQ server to CA Catalyst and the SA Manager for display on the CA SOI interfaces as CIs, alerts, or relationships.

Connector Considerations

There are some basic questions to consider before developing a connector. Depending on the domain manager and the complexity of the data, you can take different approaches, such as the following:

- Get every property for every entity and then determine how to normalize them
- Identify the data needed to fill all the USM property values and determine what domain manager data you need for those properties

Usually, the process falls somewhere in between and becomes an iterative approach as you start seeing what the domain manager data looks like and try to determine how to map it to USM types.

Some common questions to address when planning a custom connector are as follows:

- What entities and associated properties can the domain manager provide and how do I get them?
- Which of these entities can be represented in USM?

- Does the domain manager provide a group or service concept?
- How is an alert represented?
- How can you access the domain manager CI and alert data?
- Does the domain manager provide launch in context capabilities?
- Will CIs be updated or deleted?
- What USM types best describe the types of resources these entities represent?
- What USM properties require data and what is their format?
- What properties of the domain manager entities can fill the required properties of USM?
- What normalization is necessary to convert the domain manager property values to the USM format?

For more information about the USM types to which you can normalize all domain manager resources, see the USM schema documentation. For information about how to access the USM documentation, see How to Access the USM Schema Documentation.

System Prerequisites

The system that you use to build a connector using the Sample connector requires the following:

- SA Manager installed
- Sample connector installed (to gain access to the project files)
- JDK 1.6
- Eclipse 3.5
- TestNG plug-in for Eclipse

How to Set Up the Sample Connector in Eclipse IDE

Load the Sample connector project into Eclipse to build a connector and run the test harness. The Sample connector project in Eclipse provides a framework for building and testing a connector, including the required interfaces and optional test harness and mock data interfaces.

You set up a Sample connector project in Eclipse as follows:

1. Create a workspace directory in Eclipse and specify the workspace directory when you start the application for the first time.
2. Configure Eclipse to use the JRE used by CA SOI (see page 138).

3. Import the Sample connector project into Eclipse (see page 138).
4. Install the TestNG plug-in (see page 140).
5. Configure the project launch settings (see page 140).
6. Start Eclipse with a new workspace.

Configure Eclipse to Use Correct JRE

Configure Eclipse to use the JRE that CA SOI uses before you import the Sample connector.

Follow these steps:

1. Open Eclipse and select Window, Preferences.

   The Preferences dialog opens.
2. Expand Java and select Installed JREs.

   The Installed JREs page opens.
3. Click Add.

   The Add JRE dialog opens.
4. Select Standard VM and click Next.

   The JRE Definition page opens.
5. Select Directory, select the SOI_HOME\jre folder, and click OK.

   The JRE information appears in the JRE Definition page.
6. Click Finish.

   The JRE appears in the Installed JREs page.
7. Select the check box next to the CA SOI JRE and click OK.

   The JRE preference is saved and Eclipse is configured.

Import Sample Connector Project

You import the Sample connector project into Eclipse in a new workspace.

Follow these steps:

1. Start Eclipse with a new workspace and Select File, Import.

   The Import page opens.
2. Expand General, select Existing Projects into Workspace, and click Next.

   The Import Projects page opens.

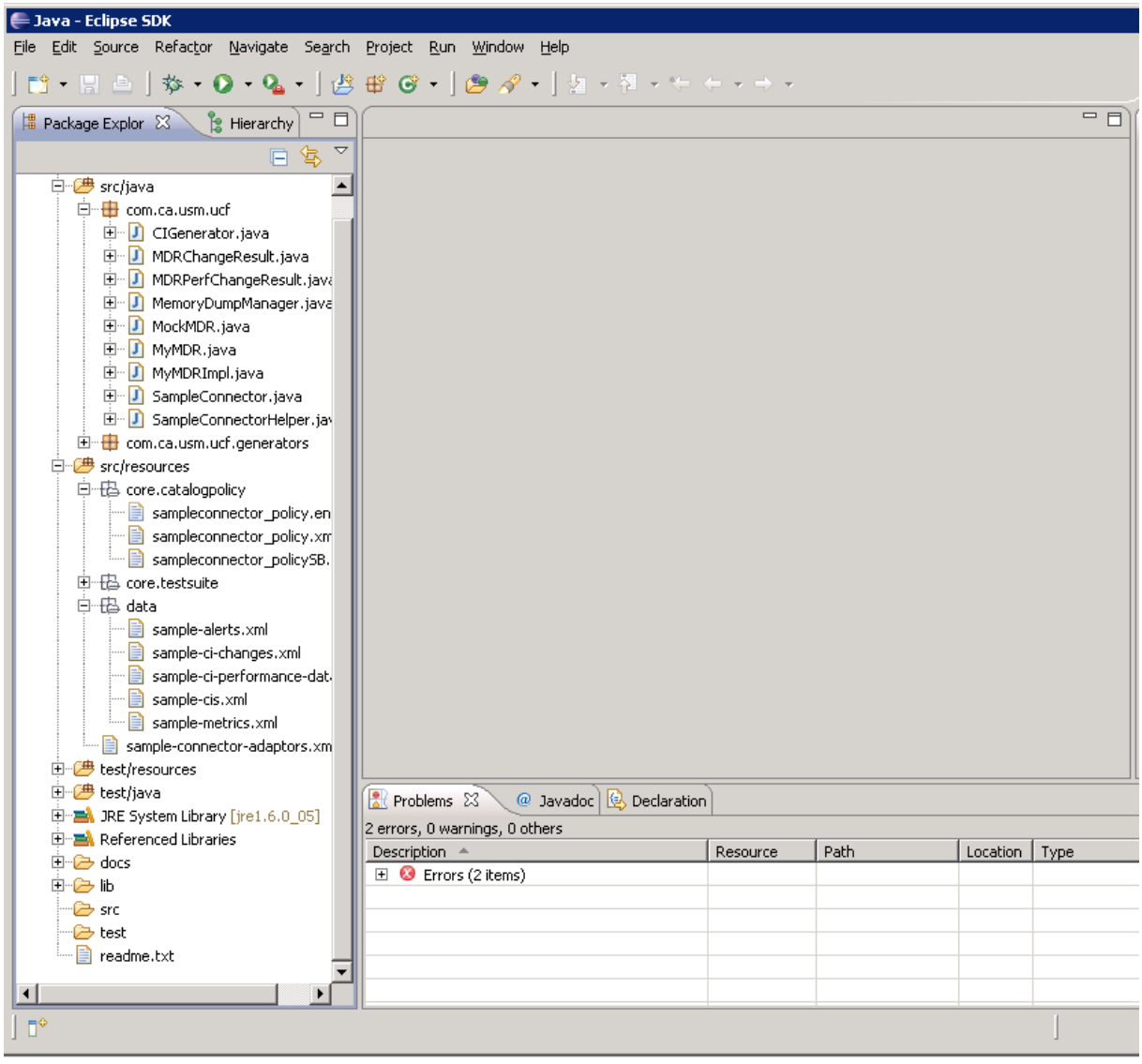
3. Choose Select root directory, click Browse, navigate to the SOI_HOME\SampleConnector directory, and click OK.

   The SampleConnector project displays as selected in the Projects pane.

4. Click Finish.

   The Sample connector project is imported.

The following screen shows the imported SampleConnector project expanded to display the provided interfaces, policy, and sample data:



Install TestNG Eclipse Plug-in

To run connector unit tests, install the TestNG plug-in in your Eclipse environment.

Follow these steps:

1. Uninstall any existing previous version of TestNG.
2. Start Eclipse and select Help, Install New Software.

   The Install page opens.
3. Click Add.

   The Add Site page opens.
4. Enter TestNG in the Name field, enter http://beust.com/eclipse in the Location field, and click OK.

   Information for the plug-in displays in the table on the Install dialog.
5. Expand TestNG, select org.testng.eclipse, and click Next twice.

   The Review Licenses page opens.
6. Accept the License Agreement and click Finish.

   TestNG installs, and Eclipse asks for the permission to restart.
7. Click Yes.

   Eclipse restarts, and the TestNG plug-in is now available for use.

Configure Project Launch Settings

You configure the project launch settings to specify where the Sample connector should start.

Follow these steps:

1. Open Eclipse.
2. Select Run, Debug Configurations.

   The Debug Configurations dialog opens.
3. Right-click TestNG and select New.

   The right pane is populated with the TestNG configuration information.
4. Enter a meaningful name in the Name field and click Browse in the Project field.

   The Project Selection dialog opens.
5. Select SampleConnector and click OK.

   SampleConnector displays in the Project field.

6. Select Class in the Run pane and click Browse next to the field.

   The TestNG dialog opens.

7. Select SampleConnectorUTest and click OK.

   The class displays in the Class field.

8. Click Apply.

   The project launch settings are configured. Complete Steps 9-11 to run a unit test.

9. Click Debug.

   The unit test runs.

10. Select the TestNG tab.

    The unit test results display.

11. Click the icon labeled Open TestNG Report.

    The execution results report displays in HTML format. All tests pass if the environment is configured correctly.

How You Implement a Custom Connector

After you have loaded the Sample connector project into Eclipse, you can begin implementing the domain manager integration for a custom connector.

Note: You must install the Sample connector on the SA Manager to implement the connector interfaces and methods.

Connector Interfaces and Methods

The Sample connector extends the USMBaseConnector class, which implements some of the common connector framework code. The Sample connector also implements the following interfaces that define connector functionality. When writing a custom connector, you can code the necessary interfaces and methods using the Sample connector framework to properly integrate with a domain manager:

Note: View these interfaces in the Eclipse project in the SampleConnector.java class located in the src/java folder.

com.ca.connector.runtime.Connector

Contains method declarations to support the connector lifecycle.

com.ca.connector.runtime.EntityManager

Contains method declarations for retrieving data from the domain manager and creating, update, and deleting data in the domain manager.

com.ca.connector.runtime.EntityChangePublisher

Contains method declarations for events related to changes in CIs.

com.ca.connector.runtime.ConnectorEventPublisher

(Not supported) Contains method declarations for non-CI related events.

com.ca.connector.runtime.EntityOperationRunner

(Not supported) Contains method declarations to support custom operations related to specific entities.

com.ca.connector.runtime.ConnectorOperationRunner

(Not supported) Contains method declarations to support connector custom operations.

com.ca.connector.runtime.OperationRunner

(Not supported) Contains method declarations required to run and monitor the progress of synchronous and asynchronous operations.

Important! The ConnectorEventPublisher, ConnectorOperationRunner, EntityOperationRunner, and OperationRunner interfaces in the connector code are placeholders with no current implementation. The operations provided by these interfaces are not supported in this release of CA SOI.

Connector Interface

The com.ca.connector.runtime.Connector interface contains the methods that control the connector lifecycle with the domain manager. This interface includes the following methods:

public void initialize(UUID *uuid*, DataObject *config*, Properties *properties*) throws UCFException;

Initializes the connector, based on a configuration file that you define in the SOI_HOME\resources\Configurations directory. The integration framework reads any configuration parameters that are listed in the connector configuration file (*ConnectionInfo* tag), and pass them as input parameters to the initialize() method. As the name indicates, this method completes any tasks required for the connector to become ready for fulfilling requests for data.

uuid

Uniquely identifies this instance of the connector.

config

Created from an instance of the ConnectorConfigDesc associated with the connector. The config object is edited by installers, the administration user interface, and other means.

properties

Defines properties that are provided to the instance and are typically used by proxy client implementations to connect to the domain manager.

public boolean isSystemUp();

Determines whether the domain manager is available. This method checks whether the system is running and returns a flag indicating whether the underlying system is up or down. The IFW uses this method to help detect error states. If this method does not return *true*, the connector is not able to come online.

public void restart();

Restarts the connector. This is an equivalent to a shutdown and reinitialize.

public void shutdown();

Shuts down the connector and cleans up the connection with the underlying system.

EntityManager Interface

The com.ca.connector.runtime.EntityManager interface contains the methods that support the data retrieval from the domain manager and inbound operations in the domain manager. Only the get() method is required if you do not want to enable inbound operations in the domain manager. This interface includes the following methods:

public Collection<DataObject> get(DataObject *selector*) throws UCFException;

Retrieves the entities that match the provided selector. The method reads data from the domain manager in response to a request from the IFW. The get() method accepts a selector of type DataObject that instructs the connector about the type of data that is being requested. In CA SOI, the underlying type for this selector DataObject is SiloDataFilter *(see page 145).* The return type for the get() method is a Collection of DataObjects. Each DataObject in the return collection is a single entity from the domain manager, having the CA SOI underlying type of SiloData (see page 146).

selector

Defines an instance of the selector type associated with one of the entity types managed by the connector.

public DataObject create(DataObject *newValue*) throws UCFException;

(Optional) Creates the specified entity in the domain manager and returns the created entity.

newValue

Defines the new value for the managed entity to be created in the domain manager.

public DataObject update(DataObject *newValue*) throws UCFException;

(Optional) Updates the specified entity in the domain manager and returns the updated entity.

newValue

Defines the new value for the managed entity to be updated in the domain manager.

public void delete(DataObject *selector*) throws UCFException;

(Optional) Deletes the entity in the domain manager that matches the provided selector.

selector

Defines an instance of the selector type associated with one of the entity types managed by the connector.

public void discover(DataObject *selector*) throws UCFException;

(Not supported) Discovers the entities that match the provided selector.

selector

Defines an instance of the selector type associated with one of the entity types managed by the connector. If the selector is found, it is reported as a normal onCreate event to CI change subscribers.

public Enumeration<DataObject> enumerate(DataObject *selector*, UUID *enumID*) throws UCFException;

(Not supported) Enumerates the entities that match the provided selector and returns an enumeration of the entity type that matches the selector. While this method is semantically equivalent to a (Java) enumeration over the results of the get(EntitySelector sel) method, this call provides the opportunity to iteratively build/transfer the results. For example, an implementation may compute/retrieve matching entities from the managed system in batches.

selector

Defines an instance of the selector type associated with one of the entity types managed by the connector.

enumID

Defines a unique ID that is subsequently used by the caller in a call to closeEnumeration() to indicate that it is no longer interested in the enumeration.

public void closeEnumeration(UUID *enumID*) throws UCFException;

(Not supported) Indicates that callers are no longer interested in the remaining values of an enumeration started earlier with the same ID.

enumID

Defines a unique ID that is subsequently used by the caller in a call to closeEnumeration() to indicate that it is no longer interested in the enumeration.

Important! While the connector framework supports the discover(), enumerate(), and closeEnumeration() methods, CA SOI does not currently use the functionality enabled by these methods. While these methods are available for implementation, the Sample connector does not contain sample code or implement any of these methods by default.

SiloDataFilter

SiloDataFilter is the primary filter that CA SOI uses when calling interfaces that the connector implements. SiloDataFilter supports the following filter elements:

entitytype

Represents the category of the USM data types in which a user is interested. The data type of the entitytype filter element is String. The valid values for entitytype are as follows:

Item

Represents all entities that a domain manager manages, such as routers, applications, services, and so on. It does not include alerts and relationships.

Relationship

Represents all USM-supported BinaryRelationships.

Alert

Represents all USM alerts that a domain manager generates.

itemtype

Note: The itemtype filter element is valid only when the entitype is set to Item. You cannot use itemtype if entitytype is set to Relationship or Alert.

Represents the specific item type that is requested. The specific USM type names define the valid itemtype values, such as ComputerSystem, Service, Router, and so on. You can use NULL if you do not want to set any refinement condition for itemtype. The data type of the itemtype filter element is String.

recursive

Specifies whether the connector recursively collects the item and its constituent children and relationships. The data type of the recursive filter element is Boolean. The valid values are true and false. True retrieves complete definition of an item, including its subitems, constituent items and the relationship between these items. False retrieves only the top-level items.

For example, if entitytype is set to Item, itemtype to Service, and recursive to true, all relationships and referenced items for a service are retrieved. If recursive is set to false, only the top-level services are retrieved.

Id

Specifies the entity instance ID as represented within the domain manager. This ID is the same as MdrElementID. When set to NULL, all entities of type set in the entitytype and itemtype filter elements are returned. The data type of the Id filter element is String.

updatedAfter

Specifies all entities of the type set in entitytype and itemtype that were updated after the specified date. The valid format is YYYY-MM-DDThh: mm: ss.sTZD. The data type of the updatedAfter filter element is DateTime.

Note: When filter is NULL; that is, no filter element is present, all entities are returned. Additionally, when multiple filter elements are present, they are combined with an AND operator.

SiloData

SiloData is the DataObject type that a connector uses to report all data to CA SOI. SiloData supports the following data elements:

entitytype

Represents the category of the USM data types that the connector returns. The valid entitytype values are as follows:

Item

Represents all entities that a domain manager manages, such as routers, applications, services, and so on. It does not include alerts and relationships.

Relationship

Represents all USM-supported BinaryRelationships.

Alert

Represents all USM alerts that a domain manager generates.

Properties

Stores the *attribute=value* pairs that the connector returns. The Properties element is of type KeyWordValuePairs.

Details

(Optional) Provides details on the USM entity being returned.

EntityChangeEventPublisher Interface

The com.ca.connector.runtime.EntityChangePublisher interface contains the method declarations necessary to subscribe to changes in domain manager data. This interface contains the following methods:

public synchronized String subscribeToChanges(DataObject *selector*, EntityChangeSubscriber *subscriber*);

Subscribes to entity change events. Returns a String containing a subscription ID which may later be used to unregister the subscription.

After the data is retrieved and returned to CA SOI in a synchronous manner, using the get() method, it is desirable to keep the domain manager view of that data in synch with the CA SOI view. To provide this capability, a connector must implement the subscribeToChanges() method. This method launches a thread to provide asynchronous updates to the IFW through a callback for any changes to the domain manager data. The method accepts two parameters as input: a DataObject selector (SiloDataFilter) and an EntityChangeSubscriber object. To manage subscriptions, the connector creates an instance of EntityChangeSubscriptionManager. The incoming subscription is added to the manager instance, which the connector can then use to return changed data to CA SOI.

selector

Defines a filter on entities to which to subscribe.

subscriber

Defines the subscriber to entity change events and returns the subscription ID.

public synchronized void unsubscribeFromChanges(String *subscriptionId*);

Unsubscribes from entity change events.

subscriptionID

Defines the subscription ID to unsubscribe.

ConnectorEventPublisher Interface

The com.ca.connector.runtime.EventPublisher interface contains the methods that control how the connector interacts with generic events not related to CIs from the domain manager. This interface includes the following methods:

Important! While the connector framework supports this interface, CA SOI does not currently use the functionality provided by these methods in integrated connectors. While these methods are available for implementation, the Sample connector does not contain sample code or implement any of these methods by default.

public Object getEarliestAvailableBookmark;

Gets a unique bookmark associated with the earliest event available to the connector.

public String subscribeToEvents(String *eventTypeID*, DataObject *filter*, ConnectorEventSubscriber *subscriber*);

Subscribes to general events published by the connector.

eventTypeID

Defines the ID of a type of event published by the connector.

filter

Defines restrictions of events to which the connector subscribes. The object type is the filter type supported by the event type in the eventTypeID parameter.

subscriber

Defines the subscriber to connector events and returns a subscription ID.

public String subscribeToEvents(String *eventTypeID*, DataObject *filter*, Object *bookmark*, ConnectorEventSubscriber *subscriber*);

Subscribes to general events published by the connector. This variant of the subscription method lets subscribers ask for events (that the connector may have missed, for example) generated since a previously received event. The start event is identified by an ID associated with the event, or a bookmark (see the ConnectorEventSubscriber API for details). The bookmark is opaque to the subscriber. This mechanism is derived from WS-MAN and provides some level of support for recovery.

eventTypeID

Defines the ID of a type of event published by the connector.

filter

Defines restrictions of events to which the connector subscribes. The object type is the filter type supported by the event type in the eventTypeID parameter.

bookmark

Identifies a previously received event. Any event that occurred after the bookmarked event is regenerated for the subscriber. Subscribers use a special value of the bookmark to request all available events returned by the getEarliestAvailableBookmark() method.

subscriber

Defines the subscriber to connector events and returns a subscription ID.

public void unsubscribeFromEvents(String *subscriptionID)*;

Subscribes to entity change events.

subscriptionID

Defines the subscription ID to unsubscribe.

EntityOperationRunner Interface

The com.ca.connector.runtime.EntityOperationRunner interface contains the method declarations to implement custom operations for specific entities within the domain manager. All methods in this interface are optional.

Important! While the connector framework supports custom operations, CA SOI does not currently use custom operations in integrated connectors. While these methods are available for implementation, the Sample connector does not contain sample code or implement any of these methods by default.

This interface contains the following methods:

public DataObject execute(String *operID*, DataObject *entityKey*, DataObject *operParams*) throws UCFException;

Executes a synchronous operation against managed entities and returns the result of the operation.

operID

Defines the name of the operation.

entityKey

Defines the entity instances to which the operation applies.

operParams

Defines the operation input parameters.

public String start(String *operID*, DataObject *selector*, DataObject *operParams*, OperationListener *listener*, String *refID*) throws UCFException;

Starts an asynchronous operation against managed entities and returns an operation unique instance ID that you can use to refer to the operation.

operID

Defines the name of the operation.

entityKey

Defines the entity instances to which the operation applies.

operParams

Defines the operation input parameters.

listener

Listens for state changes related to the operation instance. This parameter can be null, and a listener can also register for the operation later.

refID

Defines an ID understood by the caller.

ConnectorOperationRunner Interface

The com.ca.connector.runtime.ConnectorOperationRunner interface contains the method declarations to support connector custom operations for specific entities within the domain manager. All methods in this interface are optional.

Important! While the connector framework supports custom operations, CA SOI does not currently use custom operations in integrated connectors. While these methods are available for implementation, the Sample connector does not contain sample code or implement any of these methods by default.

This interface contains the following methods:

public DataObject execute(String *operID*, DataObject *operParams*);

Executes a synchronous operation against a connector and returns the result of the operation.

operID

Defines the name of the operation.

operParams

Defines the operation input parameters.

public String start(String *operID*, DataObject *operParams*, OperationListener *observer*, String *refID*);

Starts an asynchronous operation against the connector and returns an operation unique instance ID that you can use to refer to the operation.

operID

Defines the name of the operation.

operParams

Defines the operation input parameters.

observer

Listens for state changes related to the operation instance. This parameter can be 'null' and a listener can also register for the operation later.

refID

Defines an ID understood by the caller.

OperationRunner Interface

The com.ca.connector.runtime.OperationRunner interface contains the method declarations to monitor the progress of synchronous and asynchronous custom operations on the domain manager and connector. All methods in this interface are optional.

Important! While the connector framework supports custom operations, CA SOI does not currently use custom operations in integrated connectors. While these methods are available for implementation, the Sample connector does not contain sample code or implement any of these methods by default.

This interface contains the following methods:

public void abort(String *operInstID*) throws Exception;

Aborts the operation.

operInstID

Defines the unique instance ID of a running operation obtained through an initial call to start().

public void forget(String *operInstID*) throws Exception;

Forgets the operation by indicating to the underlying system that the caller is no longer interested in the results, but that the operation does not necessarily need to be aborted.

operInstID

Defines the unique instance ID of a running operation obtained through an initial call to start().

public void recover(String *operInstID*, DataObject *lastState*, OperationListener *listener*, String *refID*) throws UCFException;

Recovers an operation by restarting it from the state previously saved in an intermediateState() callback to the listener. Some operations may simply restart, others may abort as a result, and others may be able to restart from the obtained state.

operInstID

Defines the unique instance ID of a running operation obtained through an initial call to start().

lastState

Defines the last known operation state.

listener

Listens for state changes related to the operation instance. This parameter can be null, and a listener can also register for the operation later.

refID

Defines an ID understood by the caller.

public void setOperationListener(String *operInstID*, String *refID*, OperationListener *listener*) throws UCFException;

Listens to ongoing operation state changes. The interface supports a single listener per runner. If more are required, you can use a simple multiplexer pattern.

operInstID

Defines the unique instance ID of a running operation obtained through an initial call to start().

refID

Defines an ID understood by the caller.

listener

Listens for state changes related to the operation instance. This parameter can be null, and a listener can also register for the operation later.

Sample Connector Interface Implementation

When you initialize the provided Sample connector, it does the following:

- Determines the names of the data files from the configuration properties and sends them to the constructor of the MockMDR instance
- Gets the configuration properties related to MockMDR
- Connects to MockMDR
- Starts an event simulation thread that periodically checks for changes in the sample-ci-changes.xml file and generates events to send to CA SOI

In response to the get() method call, the Sample connector filtering logic first reads all of the data from the XML file into memory and then applies the filter criteria to select the objects that qualify. This may not be practical for a custom connector with a very large number of objects. The connector should translate the filter into one or more native queries to directly select only the objects that satisfy the filter criteria.

The connector methods create(), update(), and delete() let CA SOI change the contents of the connector's data repository. This enables the connector's data repository to be synchronized to the reconciled entities persisted by CA Catalyst. In the Sample connector, there is only an in-memory data repository (list of CIs). The implementation of the create(), update(), and delete() methods make corresponding changes to the in-memory list of CIs.

To support the EntityChangeEventPublisher interface, the Sample connector makes use of a shared helper class to manage multiple CI change event subscribers (EntityChangeSubscriptionManager). The helper class dispatches events based on matching subscription filters.

Mock Domain Manager and Sample Data

A connector integrates with a domain manager to retrieve data. The Sample connector uses a mock domain manager, called the mock MDR, to simulate data creation and event generation. Even when you are building a custom connector to integrate with a real domain manager, in some cases you cannot automate unit tests with the actual domain manager. In this case, you can use the mock MDR interfaces to populate your connector with sample data for testing purposes.

The Sample connector contains the following interfaces for developing or using a mock domain manager:

com.ca.usm.ucf.MyMDR

Contains method declarations to implement a mock domain manager. This interface lets you switch between retrieving mock data and real data. Two implementations of MyMDR are included in the Sample connector: MockMDR and MyMDRImpl. Since the Sample connector always uses a mock domain manager (implemented in MockMDR.java), the implementation of retrieving data from the real domain manager is merely a placeholder (MyMDRImpl.java).

com.ca.usm.ucf.MockMDR

Contains method declarations for a full implementation of a mock domain manager by the Sample connector. The Sample connector uses this interface and the provided sample data as its primary data repository. You can use this class as the mock domain manager for your custom connector, but you must modify the methods in this class so that the mock domain manager and sample data mirrors the integration method and data of your domain manager.

com.ca.usm.ucf.MyMDRImpl

Contains method declarations for maintaining a shell of the real domain manager instance. All methods in this class throw an exception indicating that the operation is not supported because the Sample connector does not have a real domain manager. Any custom connector would have to replace the method implementation in this class with the code to access the real domain manager.

Using a mock domain manager in a custom connector is optional. If you want to do so, you can use the MockMDR interface to create a mock domain manager that simulates data from your domain manager. See the code in the MyMDR interface for an example of how the Sample connector uses MockMDR to create its data repository.

The Sample connector configuration contains a mockMode parameter that controls whether it is retrieving mock data. Because the Sample connector can only retrieve mock data, this parameter is true by default. If you create a mock domain manager for your custom connector, you can include this parameter as false by default, so that you can switch to mock data whenever necessary.

Create Connector Policy

Create a connector policy for your custom connector to transform data from the source domain manager format to the USM format and optionally to transform USM data back to the domain manager format to push changes to the domain manager.

Policy for each connector must be present in the SOI_HOME\resources\Core\Catalogpolicy directory for any processing to occur on objects retrieved from the domain manager. Without policy, domain manager data cannot be displayed in CA SOI.

For information about writing connector policy, including detailed syntax and examples for all available policy operations, see Writing Connector Policy (see page 177).

All outbound policy must transform data to adhere to the USM schema for the data to appear correctly in CA SOI. For more information about the USM schema, see Unified Service Model (see page 51). For more information about the types and properties supported in the USM schema, see the USM schema documentation.

Sample Connector Policy Implementation

The Sample connector implements the following policy files in the SOI_HOME\resources\Core\Catalogpolicy directory:

sampleconnector_policy.xml

Transforms outbound data from the source format defined in the Sample connector mock domain manager to the USM format. This is the outbound policy file that transforms all objects from the connector as outputs from the get(), create(), and update() methods. The Sample connector outbound policy leverages global policy, which eliminates the need for the connector to handle actions common to all connectors. For more information about global policy and how to use it in outbound policy, see Global Policy (see page 180).

sampleconnector_policySB.xml

Transforms inbound data from the USM format to the source format defined in the Sample connector mock domain manager. This is the inbound policy file that transforms all objects stored in CA SOI as inputs to the create(), update(), and delete() methods.

The Sample connector supports both USM and non-USM formats and has a mixture of both entity types in the sample data files. Therefore, the provided connector policy handles entities that require transformation and entities that are already in the USM format and pass through the policy with no further processing required. The test data sets the eventtype property to USM-Entity to indicate that the entity is already in the USM format and does not need transformation.

The Sample connector sample data includes examples of several CI types. These types are covered in the connector policy. The mappings from the Sample connector types to USM types in the Sample connector policy are as follows:

Sample Connector Type	USM Type
System	ComputerSystem
Windows Server	BackgroundProcess
Database	DatabaseInstance
Disk Partition	File
LAN Interface	InterfaceCard
Processor	Processor
Router	Router
Service	Service

The Sample connector supports the native CI types listed in the Sample Connector Type column. However, it supports any USM CI type if the data is in USM format for outbound and inbound operations.

You can use the provided Sample connector policy as a point of reference or the framework from which you create a policy for your custom connector.

Outbound Policy Example

The following policy example shows how the provided Sample connector outbound policy transforms sample data of the System class to the ComputerSystem USM CI type:

```
<EventClass name='Item' >
  <Classify>
    <Field input='class' pattern='^System$' output='eventtype'
    outval='ComputerSystem' />
  </Classify>
```

The classify operation classifies CIs with a class of System to the USM type of ComputerSystem. This operation directs all specialized processing to occur under the policy for the ComputerSystem event class. The object also inherits policy operations from the parent Item event class.

```
<Normalize>
  <Field output='MdrProdInstance' outputtype='ref' type='map'
  input='MdrProdInstance' >
    <mapentry mapin='^MdrProdInstance$' mapout='{fqdn(localhost)}' />
    <mapentry mapin='localhost' mapout='{fqdn(localhost)}' />
  </Field>
</Normalize>
```

The normalize operation in the generic Item event class derives the name of the local host to include as the source of the information.

```
<Format>
  <!-- Five digit CA Product identifier as defined by the USM
  MdrProductEnum -->
  <Field output='CAProductIdentifier' format='09998' input='' />
  <Field output='MdrProduct' format='CA:{0}' input='CAProductIdentifier' />
  <!-- MdrProdInstance setup above in normalize section
  <Field output='MdrProdInstance' format='{0}' input='{fqdn(localhost)}' /-->
  <Field conditional='id' output='MdrElementID' format='{0}' input='id' />
  <Field output="UrlParams" format="http://{0}:8080?id={1}"
  input="MdrProdInstance,MdrElementID" />
  <Field conditional='description' output='Description' format='{0}'
  input='description' />
  <Field conditional='name' output='Label' format='{0}' input='name' />
</Format>
```

The format operations in the generic Item event class add the '09998' identifier to any data retrieved from the Sample connector. Every connector has a similar identifier that is included in the final USM entity. Other operations configure the basic CI properties related to the data source.

```
<EventClass name='ComputerSystem' extends='Item' >
  <Format>
  <!-- Assign class name -->
    <Field output='ClassName' format='ComputerSystem' input='' />
    <!-- Correlatable properties, must populate at least one -->
    <Field conditional='dnsname' output='PrimaryDnsName' format='{0}'
    input='dnsname' />
    <Field conditional='sysname' output='SysName' format='{0}' input='sysname' />
    <Field conditional='macaddress' output='PrimaryMacAddress' format='{0}'
    input='macaddress' />
    <Field conditional='ip_address' output='PrimaryIPV4Address' format='{0}'
    input='ip_address' />
    <!-- Non-Correlatable properties -->
    <Field conditional='name' output='ComputerName' format='{0}' input='name' />
  </Format>
```

The format operation in the ComputerSystem event class converts the properties from the source Sample connector CI format to the USM format. Properties such as DNS name, sysname, and MAC address are converted to the standard USM format. After all format operations complete, the CI is a USM entity.

Inbound Policy Examples

The following policy example shows how the provided Sample connector inbound policy transforms USM data of the Router type class to the Router class specifications in the Sample connector mock domain manager:

```
<EventClass name='Item' >
  <Classify>
    <Field input='ClassName' pattern='^Router$' output='eventtype'
    outval='Router' />
  </Classify>
```

The classify operation classifies CIs with a USM type of Router to the Sample connector class of Router. This operation directs all specialized processing to occur under the policy for the Router event class. The object also inherits policy operations from the parent Item event class.

```
  <Format>
    <Field input='LastModActivity' format='{0}' output='action' />
    <Field output='CAProductIdentifier' format='09998' input='' />
    <Field conditional='MdrElementID' input='MdrElementID' format='{0}'
    output='id' />
    <Field conditional='Description' input='Description' format='{0}'
    output='description' />
    <Field conditional='Label' input='Label' format='{0}' output='name' />
    <!-- correlated properties -->
    <Field conditional='SysName' input='SysName' format='{0}' output='sysname' />
    <Field conditional='PrimaryDnsName' input='PrimaryDnsName' format='{0}'
    output='dnsname' />
    <Field conditional='PrimaryMacAddress' input='PrimaryMacAddress' format='{0}'
    output='macaddress' />
    <Field conditional='PrimaryIPV4Address' input='PrimaryIPV4Address'
    format='{0}' output='ip_address' />
  </Format>
```

The format operations in the generic Item event class convert USM properties to the source format in the Sample connector mock domain manager.

```
  <Write>
    <Field type='file' name='outfile' properties='*' />
    <Field type='publishcache' properties='name,description,id,action,sysname,
    dnsname,macaddress,ip_address' />
  </Write>
</EventClass>
```

The write operation in the generic Item event class writes the derived properties to the output CI. The CI is then processed further by the specialized policy under the Router event class.

```
<EventClass name='Router' extends='Item' >
  <Format>
    <Field output='class' format='Router' input='' />
    <Field conditional='AdministrativeStatus' input='AdministrativeStatus'
    format='{0}' output='adminstatus' />
    <Field conditional='IsInMaintenance' input='IsInMaintenance'
    format='{0}' output='maintenanceFlag' />
  </Format>
```

The format operation in the Router event class converts the class property in the output CI to Router, the AdministrativeStatus property to adminstatus, and the IsInMaintenance property to maintenanceFlag.

```
  <Write>
    <Field type='file' name='outfile' properties='*' />
    <Field type='publishcache' properties='class,adminstatus,maintenanceFlag' />
  </Write>
</EventClass>
```

The write operation in the Router event class writes the final CI to an outfile for transmission to the Sample connector mock domain manager sample data.

In addition to this example, all inbound policy must include an event class of SiloDataFilter. All DataObject instances passed to the create(), update(), and delete() methods are transformed before the methods are invoked. Since the top-level type name is SiloDataFilter for the delete() method, this event class is used as the value for the property eventtype when the filter properties are converted. Therefore, SiloDataFilter must match the name of an event class. The Sample connector policy implements this event class as follows:

```
<EventClass name='SiloDataFilter' >
    <!-- properties are as follows (with example below): -->
    <!-- + id (xs:string) -->
    <!-- + updatedAfter (xs:dateTime) -->
    <!-- + entitytype (xs:string) Item, Relationship, or Alert -->
    <!-- + itemtype (xs:string) -->
    <!-- + recursive (xs:boolean) -->
    <!-- Format>
       <Field output='mdrelementid' input='id' format='{0}' />
    </Format-->
```

```
    <Write>
      <Field type='file' name='outfile' properties='*' />
      <Field type='publishcache' properties='id,updatedAfter,
      entitytype,itemtype,recursive' />
    </Write>
</EventClass>
```

USM Metrics Support

The Sample connector provides an example of how to support USM metrics. It implements the following operations as defined in USM:

Note: CA SOI currently does not support viewing metric information (that connectors collect) in its user interface.

GetAvailableMetrics

Retrieves the metrics that can be queried as follows:

- Identifies the CI for which metrics are requested by its MdrProduct, MdrProdInstance, and MdrElementID values
- Returns all possible metrics (standard and domain-specific) for the CI and its domain-specific identifier for each metric

GetMetricsValues

Determines the metric values for the supplied metric names. Metric values can be either the historical values or the current live value.

The operations are implemented as custom operations using the execute() method of the EntityOperationRunner interface (see page 149).

Note: For more information about the USM metric collection capabilities, see the USM schema documentation.

Metric Data Repository

The SOI_HOME\resources\SampleConnector\data\sample-metrics.xml file is the data repository for metric data. You can add new metric definitions and values to the file, and the Sample connector uses all metric definitions and values for sample metric collection.

The following example shows how metrics are defined in the sample-metrics.xml file:

```
<silodata xsi:type="usm:SiloData">
   <properties>
       <property name="MdrElementID" value="3" />
       <property name="MetricName" value="total_accounts" />
       <property name="MetricDescription" value="total_accounts" />
```

```
        <property name="MetricType" value="Gauge" />
        <property name="MetricUnitDefinition" value="Number" />
        <property name="MetricDataType" value="Int" />
        <property name="IsOnlyLiveData" value="false" />
        <property name="MinGranularityInSecs" value="3600.0" />
        <property name="Value" value="100" />
        <property name="Timestamp" value="2010-08-05T11:00:00-06:00" />
        <property name="Value1" value="200" />
        <property name="Timestamp1" value="2010-08-17T11:00:00-06:00" />
    </properties>
</silodata>
```

This syntax defines the total_accounts sample metric as follows:

- It is a gauge metric type and integer data type expressed as a number.
- It tracks historical data (because the IsOnlyLiveData property is false).
- It has two historical value entries with corresponding timestamps.

To indicate a historical data metric, define multiple Value properties incremented by integers, and define a timestamp for each value.

Connector Configuration Files

To deploy and use a custom connector in the CA SOI IFW, you must define a configuration file for the connector in the SOI_HOME\resources\Configurations directory. This configuration file is an XML file that helps perform various tasks similar to any other connector configuration file. For example, it contains data pertaining to how to connect to the domain manager, the location of the connector class, the name of the connector, and so on.

You can create the connector configuration file manually by following the template file *myMDR_template.xml* provided with the Sample connector installer kit. Comments marked within this template file explain the properties that help you create your configuration file. You must create a copy of the template file, edit it as required, and rename it based on the naming convention (<MdrProduct>_template.xml). Additionally, while creating your configuration file, you must consider the following points:

- Some entries in the configuration files begin and end with the @ symbol. These are tokenized fields that the installer replaces with data that the user provides during installation. Any data that you want to collect from the user must have an entry with a tokenized value in the configuration file.
- Connector configuration files must begin and end with the main ConnectorConfig tag. Additionally, files must include the following tags and properties:
  - Must begin and end with the <Silo> tag
  - Must contain the fields *State* and *name*

- Must contain the <ImplementationClass> tag and *name* and *policy* fields
- Must contain the <ConnectionInfo> tag
- Must contain the <ConnectorControls> tag

- The installer does not change the string literal values in the template file; therefore, they remain *as is* after installation. You can change these values while creating the configuration file, by directly editing the file after the connector installation, or using the Administration UI.

During installation, the installer replaces the word *template* in <MdrProduct>_template.xml with the location of the domain manager server to which the connector is pointing (MdrProdInstance). For example, the sample template is called *myMDR_template.xml*. During installation, the user is prompted for the location of the domain manager server. If the user enters the server name as myServerX, the configuration file is renamed to myMDR_myServerX.xml.

Note: The connector configuration file becomes available under the SOI_HOME\resources\Configurations folder when the connector installation is done.

Create the Connector Configuration File

To create the connector configuration file, you must use the template file *myMDR_template.xml* provided with the Sample connector installer kit. The template file includes various comments about properties that help you create your configuration file.

Follow these steps:

1. Unzip the file *connector-installkit.zip* available at SOI_HOME\SampleConnector\InstallKit.

   The connector-installkit folder is extracted into the SOI_HOME\SampleConnector\InstallKit folder.

2. Open the ..\connectors\resources\ConnectorConfigTemplates folder, and locate the template XML file *myMDR_template.xml.*

3. Create a copy of the template file, rename the copied file (for example, <MdrProduct>_template.xml), and open the renamed file in a text editor.

   Note: During installation, the installer replaces the word *template* in <MdrProduct>_template.xml with the location of the domain manager server to which the connector is pointing (MdrProdInstance).

   The XML file opens in the text editor.

4. Complete the following parameters, and save the file:

   Note: Notice that several properties already have default tokenized values (see page 160) in the template file. These properties typically require custom values that users supply during installation.

   MdrProdInstance

   Identifies the domain manager instance. For example, ABC123.ca.com.

   This parameter is helpful when you have multiple instances of a product installed in your enterprise. You can enter a tokenized value (see page 160) so that the user can define the domain manager instance during installation.

   MdrProduct

   Specifies a unique identifier (as defined by the open enumeration, MdrProductEnum) naming the domain manager. For example, CA:09998.

   name

   Specifies the name as <MdrProduct>_<MdrProductInstance>. For example, CA:09998_ABC123.ca.com.

   State

   Specifies whether the connector is set to enabled or disabled state. Enabled and notEnabled are the valid values.

   ImplementationClass

   Specifies the following parameters in the ImplementationClass section:

   descriptorClass

   Specifies the implementation of the com.ca.connector.metadata.ConnectorDescriptor interface for the connector. Set to com.ca.usm.ucf.utils.USMBaseConnectorDescriptor.

   name

   Specifies the name of the Java class for the connector that implements the com.ca.connector.runtime.Connector interface.

   policy

   Specifies the policy file used to transform raw data collected from a domain manager into USM data.

   sbpolicy

   Specifies the policy file used to transform USM data into domain manager format for create, update, or delete requests.

ConnectionInfo

Specifies any configuration values that you must make available to the connector. Any attribute and its assigned values listed in this tag are passed to the initialize method of the connector when the connector starts. These properties often required tokenized values (see page 160) that a user must supply during installation.

EncryptedProperties

Specifies which of the attributes listed in ConnectionInfo must be stored and treated as encrypted values. It includes the following parameter:

name

Specifies the comma-separated list of attribute names.

ConnectorControls

Specifies the following set of Boolean flags that control various aspects of how the framework uses the connector:

Note: 1 and 0 are the two valid values for these flags. 1 represents that the flag is turned on and 0 implies that it is turned off.

dns_resolution

Specifies whether to use DNS resolution to resolve device names. If a reliable DNS mechanism is not in place (for example, no DNS server on the network, or CIs not defined to the DNS), disable DNS lookups to prevent CI resolution and normalization failure.

Default: 1

getCIsAtStartUp

Specifies whether to rediscover CIs every time the connector starts. You typically enable this control so that your connector always provides a current record of all CIs from their domain managers. Turn this control off if the connector does not support collecting CIs at startup, which applies for Level 1 and 2 integrations.

Default: 1

getRelationshipsAtStartup

Specifies whether to rediscover relationships every time the connector starts. You typically disable this control so that relationships are only obtained and imported as a part of service model imports. You should only enable this control if you require relationship CIs outside of imported service models.

Default: 0

isRemotable

Specifies whether to allow the connector framework to access the connector remotely for create, update, and delete operations on the source domain manager. Only enable this control if you have coded the create(), update(), and delete() (see page 143) methods for inbound to connector operations.

Default: 1

performDeltaProcessing

Specifies whether to process and publish deltas on CIs between the time the connector or SA Manager was last stopped or restarted. When enabled, this setting also performs delta processing on relationships if the getRelationshipsAtStartup property is enabled.

Default: 1

useAlertFilter

Specifies whether to filter alerts based on their existence in a managed service in CA SOI. If the control is turned on, the connector only sends domain manager alerts that are associated with a CI that is part of an existing managed service in CA SOI. If the control is turned off, the connector forwards all alerts from the domain manager, regardless of whether they relate to a service.

Default: 1

useServiceFilter

Specifies whether to send all relationships to CA SOI or only the ones associated with modeled services. Set the control to true to run relationships through a service filter and receive only the relationships associated with modeled services.

Default: 1

LICURLS

Helps CA SOI to allow launch of web-based UIs in context of an Item or Alert. A connector can specify URL information for any web-based UIs that the domain manager provides. You can then launch the URL in context of an Item or Alert from the CA SOI user interfaces. The URL parameter under LICURLS defines a single launch-in-context URL to be displayed in the CA SOI UI context menus. You can define multiple URL entries by including multiple configuration entries. URL includes the following parameters:

Type

Specifies whether a URL is associated with an Item or an Alert. Possible values of Type are Item or Alert.

seqNum

Specifies the identifier for the URL and the order in which URLs are displayed. For a connector, each URL of a specific Type must have a unique seqNum.

url

Specifies the URL string to launch. URLs are formed using substitutable parameters. Typically, you can include any property of an Item in the URL. To specify a substitutable parameter, enclose it within {}.

Examples are url="{Protocol}://{host}:{Port}/CIPortal?id={MdrElementID}", url ="http://{host}:7070/sam/ui?parm={UrlParms}", url="http://www.ca.com", url="{UrlParams}", or url="http://www.google.com/search?q={MdrElementID}".

Protocol

(Optional) Identifies the protocol portion of the URL (if needed by the URL). For example, Protocol="http" or Protocol="https".

host

(Optional) Identifies the host name portion of the URL (if needed by the URL).

Port

(Optional) Identifies the port number to use in the URL (if needed by the URL). For example, Port="8080".

Label

Specifies the string that the CA SOI user interface displays. This can be any string that easily identifies the URL being launched. For example, Label="Spectrum IM OneClick UI".

BindingProtocol

Specifies the transport protocol on which connectors are published as a service. It includes the following parameter:

supportedProtocols

Specifies the supported protocols, which are jms and ws.

connector-type-meta-data

Specifies the connector properties describing the capabilities of the connector. These properties are common to all instances of the connector.

instance-meta-data

Specifies the connector properties specific to a particular instance of the connector.

The connector configuration file is created.

Connector Operations

You can classify connector operations in the following two phases:

- Start phase (see page 166)
- Run phase (see page 166)

Start Phase

During this phase, the connector performs the following operations:

- Performs the initialization task and connects to the domain manager.
- Retrieves all relevant objects from the domain manager for the CI import, including service CIs.
- Retrieves the open alerts for the relevant objects from the domain manager.

Run Phase

During this phase, the connector may receive requests from the SA manager (for example, for service imports or a list of currently open alerts), and needs to publish new CIs and alerts from the domain manager. You can achieve this task by active polling or a notification mechanism, depending on what the application provides.

In this phase, the connector performs the following operations:

- Requests for service import and publishes the complete service definition, comprising at least the service CI and relationship definitions for the included CIs.
- Requests for open alerts and retrieves the open alerts for the relevant objects from the domain manager.
- Adds a CI or service and retrieves a new object created in the domain manager and publishes the CI in CA SOI.
- Adds alerts and retrieves a new alert created in the domain manager and publishes it in CA SOI.

How to Test a Custom Connector

You can use a provided test harness to test your custom connector before you deploy it to a production CA SOI environment. The test harness uses your implemented connector, connector policy, data files, and a configuration file to operate the connector in a test environment. The test harness initializes the connector, invokes get, create, update, and delete operations, subscribes to CI change events, and waits for the connector to send events. All output objects are transformed into the USM format using the connector policy and validated in the USM schema. Any errors are reported in the console log and flagged as TestNG errors.

Note: You must install the Sample connector on the SA Manager to use the test harness.

Test Harness Method Implementation

You invoke the test harness from the Eclipse IDE. To implement the test harness, extend the abstract class com.ca.usm.ucf.ConnectorTest and implement the abstract methods in the subclass. The test harness base class provides a number of standard test cases to exercise the basic functions of your connector. These tests run by default. Your subclass also serves as an extension point to let you write more test cases for the specifics of your connector.

The Sample connector uses the SampleConnectorUTest class to implement the test harness. You can use this class as an example or as a starting point for writing custom connector test cases. The SampleConnectorUTest class implements the following methods:

public abstract void setupTest()

Reads any connector-specific property file to get configuration values, instantiate the connector, and assign it to the connector field. The test harness uses the connector instance provided to run tests against it.

This method takes information from the usmtest.properties file (see page 168) by default, which is the single configuration file for the connector test cases. You can reuse this file when configuring tests for your custom connector by modifying the necessary properties to fit with your connector.

This is the only required method that the connector's test class must implement. All other test harness methods are optional. The default implementation is provided by the ConnectorTest class, but you can override it in the connector's test class to modify the test harness behavior.

public void beforeTestMethod(Method *method*)

Defines any custom setup to perform before each test is invoked.

method

Defines an object of a class java.lang.reflect.Method representing a test method to invoke. It contains information about the method, such as method name.

public void afterTestMethod(Method *method*)

Defines any custom setup to perform after each test is invoked.

method

Defines an object of a class java.lang.reflect.Method representing a test method to invoke. It contains information about the method, such as method name.

Configure the Test Harness Properties

By default, the test harness uses the usmtest.properties file to determine the test cases and data. You can modify the usmtest properties file to configure tests for your custom connector or create a new usmtest properties file. The usmtest properties file must reside in the classpath for reading configuration values, but the SOI_HOME\SampleConnector\test\resources directory is the directory where files are included in the classpath by default. The property file must be present for the test harness to work correctly.

The following information must be defined in the properties file:

Configuration parameters

Defines the resources to use to perform the tests. The following parameters are included in the usmtest.properties file:

data_source, alert_data_source, change_event_data_source

Define where the test harness retrieves data for the tests. The Sample connector usmtest.properties file contains separate properties for CIs (sample-cis.xml), alerts (sample-alerts.xml), and CI changes (sample-ci-changes.xml).

policyFile

Defines the location and file name of the connector policy file that the test harness should use to test outbound operations.

southboundPolicyFile

Defines the location and file name of the connector policy file that the test harness should use to test inbound operations.

The Sample connector includes other configuration parameters that the test harness uses. For more information, see the SOI_HOME\SampleConnector\test\resources\usmtest.properties file.

Test cases

Defines specific operations to test using specified data. The IFW calls the methods implemented by the connectors using filters, also called selectors. Selectors define what kind of data you want to retrieve or pass it as a parameter. You can define multiple test selectors, and each selector results in a call to a method identified by a selector. Each selector can have multiple properties (comma-separated key value pairs).The examples that follow describe some of the test cases included in the usmtest.properties file. For additional examples, see the usmtest.properties file.

Example: Testing the get() method

The test harness recognizes a selector for the get() method named "getSelector". You can run multiple instances of the test by appending a digit to the property name, such as getSelector1, getSelector2, and so on. The following example of a get() selector asks to return a ComputerSystem CI with an id value of 3 from the in-memory data repository of the Sample connector:

```
getSelector5="entitytype=Item,itemtype=ComputerSystem,id=3"
getSelector5.expected.datafile=ComputerSystem_3.txt
getSelector5.compare.ignorecase=true
getSelector5.compare.locatorfields=MdrElementID
getSelector5.compare.ignorefields=CreationTimestamp,LastModTimestamp,
MdrProdInstance,UrlParams
getSelector5.expected.result=true
```

The first line in the example instructs the test harness to look for a Computer System CI with an id value of 3. The second line specifies that the data returned by the get() method should look like properties and values listed in the ComputerSystem_3.txt file. The expected.result value of true indicates that the test is expected to be able to correctly find the item and that the actual returned data matches the expected data.

The locatorfields property lists properties that the test harness can use to speed up the lookup process, while the ignorefields property lists properties to ignore in the comparison between actual and expected data.

Example: Testing the create() method

Specifying selectors for create() and update() methods is very similar. The following example specifies to create the objects located in the SOI_HOME\SampleConnector\test\resources\SampleConnectorCreateValidDataFile.txt file:

```
create1.input.datafile=SampleConnectorCreateValidDatafile.txt
create1.compare.ignorecase=false
create1.compare.ignorefields=CreationTimestamp,LastModTimestamp,MdrProdInstance,
UrlParams,MdrElementID,id,outfile,entitytype
```

Several CIs exist in the data file that the test harness is expected to create. Both the create() and update() methods return a data object that has been created or updated. The test harness compares the property values from the returned objects with the values specified as input. The test harness ignores the list of properties defined in ignorefields when comparing between actual and expected data.

Example: Testing the delete() method

Each delete() method test requires a selector that is expected to match only one CI or none at all. The following example deletes a CI that the test harness has created in a previous operation:

```
delete1="entitytype=Item,id=SampleConnector-0"
```

Example: Performing negative tests

You can create negative test cases that are designed to test how the connector responds to failure situations.

The following example of a get() test intentionally specifies an invalid entitytype:

```
getSelector13="entitytype=Negative.Test"
getSelector13.expected.result=false
getSelector13.expected.exception=true
getSelector13.expected.exception.name=com.ca.ucf.api.UCFException
```

This test intentionally specifies an invalid entitytype of Negative.Test and defines the expected result as false. The test also specifies an exception that is expected to return.

The following example of a delete() test defines an invalid selector:

```
delete3="entitytype=Item,id=SampleConnector-NoNExIsTiNG"
delete3.expected.result=false
delete3.expected.exception=true
delete3.expected.exception.name=com.ca.ucf.api.InvalidParameterException
```

Test Harness Flow

You run the test harness using TestNG in Eclipse. The test harness follows a simple workflow that begins with any @BeforeClass annotated methods that signify execution before the connector class instantiation. This includes the initializeContainer() method, which calls the SampleConnectorUTest.setupTest() method to get the test configuration, starts the transformation engine, and initializes the connector according to the test configuration. Once the initializeConnector() method completes, the test harness is now connected to the data source according to the configured test properties in the usmtest.properties file.

The test harness calls each @Test annotated method in the appropriate order. This includes the following:

getTest()

Retrieves CIs according to the configured selector filters, prints out the name and value property pairs for each CI, converts the CIs to USM XML, and performs a USM validation against that XML.

testSubscribeCIChanges(), testCreateEvents(), testDeleteEvents(), testUpdateEvents()

The subscribeCIChanges() method listens for CI change notifications from the connector, and when events are received, they are added to create/delete/update queues. The test methods then extract the information from the associated queue, validate it, and publish it to the test console. The usmtest.properties file has more information about how to configure the number and type of events the test harness expects.

testCreate(), testUpdate(), testDelete()

Test writing to the connector by running create, update, and delete test cases in the usmtest.properties file.

The test harness tests all methods sequentially, requiring you to only write a single test class (SampleConnectorUTest, in the case of the Sample connector) and implement setupTest().

Run the Test Harness

You can run the test harness as a TestNG test in the Eclipse debugger.

Follow these steps:

1. Open Eclipse and set any initial breakpoints of interest in your connector and your connector test class.
2. Right-click your test class in the Project Explorer and select Debug As, TestNG Test.

   The test harness runs and displays the test results in the Console tab.

Custom Connector Logging

The IFW and all connectors use Apache log4j for logging and debugging messages. The default log4j.xml file is located at SOI_HOME\resources. This file is loaded at IFW startup and defines multiple loggers used by the IFW and a logger that connectors can use to direct their log and debug messages to a default log file.

Default Connector Log File

The SOI_HOME\resources\log4j.xml file directs connector logging information by default to the SOI_HOME\log\ifw.log file. To use this default log file, your connector package must start with "com.ca.usm.ucf." The default logger definition in the log4j.xml file is as follows:

```
<!-- Default connectors ->
<logger name="com.ca.usm.ucf" additivity="false"
   <level value="ERROR" />
   <appender-ref="IFW" />
</logger>
```

How to Create a Connector-Specific Log File

You can define an additional log4j appender and logger for your connector so that it uses its own dedicated log file and sets log levels independently of other connectors on the same system.

Follow these steps:

1. Create a connector-specific log4j.xml file with an appender and logger formatted as follows:

   Appender:

   ```
   <appender name="TemplateConnector"
   class="org.apache.log4j.RollingFileAppender">
       <param name="File" value="&logDir;/Template_Connector.log"/>
       <param name="Append" value="true"/>
       <param name="MaxFileSize" value="20MB"/>
       <param name="MaxBackupIndex" value="10"/>
       <layout class="org.apache.log4j.PatternLayout">
           <param name="ConversionPattern" value="&filePattern;"/>
       </layout>
   </appender>
   ```

   Logger:

   ```
   <logger name="com.ca.usm.TemplateConnector">
       <level value="ERROR" />
       <appender-ref ref="TemplateConnector" />
   </logger>
   ```

   Adhere to the following conventions for overall log4j file conventions and appender and logger properties:

   log4j file name and location

   Use a unique and easily identifiable name, such as <ProductName>Connector_log4j.xml. Install the <ProductName>Connector_log4j.xml file to the following location: SOI_HOME\Configurations\log4j.

appender name

Use a name unique to your connector, such as <ProductName>Connector.

logger name

Use a name that makes the logger unique to your connector, such as com.ca.<ProductName>.

log file name and location

Use a name that adheres to the following convention: <ProductName>_connector.log. Install the <ProductName>_connector.log file to the following location: SOI_HOME\log.

Do not define a ROOT logger in this configuration file.

2. Update the connector code to load the created log4j configuration file at connector startup by inserting code similar to the following the Initialize() method:

```
if(System.getProperty("IFW_RESOURCES")!=null)

DOMConfigurator.configureAndWatch(System.getProperty("IFW_RESOURCES")+File.se
parator+
"resources"+File.separator+"Configurations"+File.separator+"log4j"+File.separ
ator+
"ProductNameConnector_log4j.xml",30000);
```

This code should be the first called after the connector is instantiated so that the log file can capture all relevant information.

3. Package and install the connector-specific log4j configuration file with the connector.

For more information about using the provided installer kit to add and install the connector-specific log4j configuration file to the appropriate location, see Custom Connector Deployment (see page 173).

Custom Connector Deployment

You can deploy a custom connector in one of the following ways, depending on your requirements:

- Create a custom connector installer (see page 174)
- Deploy a custom connector manually (see page 174)

How to Create a Custom Connector Installer

For properly deploying a custom connector, use the provided installer kit to create an installer, package the appropriate files, and put all files in the right places. Connector materials that you need to package are as follows:

- Connector-specific binaries such as executables, java classes and jar files, and libraries that make up the connector
- Connector-specific dependencies such as third-party utilities and product-specific APIs, which the connector uses at runtime
- Connector policy files
- Connector configuration file

The installer kit files are located at SOI_HOME\SampleConnector\InstallKit. Unzip the file in this directory to gain access to all materials necessary for generating an installer, including a custom InstallAnywhere project. The installer kit helps you create an installer that does the following:

- Installs the connector materials to the appropriate places.
- Installs the IFW on a system where it does not exist.
- Populates the connector configuration file that you created based on information provided during installation.

The kit uses InstallAnywhere to build a fully functional installer for your connector. Use the following InstallAnywhere connector installer template file to create the installer: SOI_HOME\SampleConnector\InstallKit\InstallProject\Connector_Template\Connector_Template.iap_xml.

Note: Previous experience building InstallAnywhere installers is necessary to use the materials in this kit. If you do not have the necessary experience, consider manually deploying your custom connector (see page 174).

How to Manually Deploy a Custom Connector

Follow these steps for manually deploying a custom connector on a computer where the IFW is already available:

1. Compile the connector code into a jar file.
2. Stop the CA SAM Integration Services service on the connector system.
3. Copy all connector files to their appropriate places manually:
   - Connector-specific binaries, such as executables, Java classes, jars, and libraries, that make up the connector:
     - User-built executables and Java classes at SOI_HOME\bin
     - User-built jars and libraries at SOI_HOME\lib

- Connector-specific dependencies, such as third-party utilities, product-specific APIs, and so on, that the connector uses at runtime:
  - Domain-specific APIs at SOI_HOME\lib\<ProductName>
  - Third-party utilities at SOI_HOME\lib\Common
- Connector policy at SOI_HOME\resources\Core\Catalogpolicy
- Connector configuration file at SOI_HOME\resources\configurations

4. Change the connector configuration file name manually to use the domain manager instance (that is, *MDRID*_server5.xml).
5. Edit the connector configuration file manually to replace the tokenized values with custom ones for integrating with the data source.
6. Restart the CA SAM Integration Services service.

Chapter 9: Writing Connector Policy

This section provides an overview of connector policy and explains the policy operations.

This section contains the following topics:

Connector Policy Overview

Connector policy defines the conventions the connectors use to classify, filter, parse, normalize, enrich, and format CIs, relationships, and alerts received from a domain manager and send them in a specified format to the SA Manager for display in the Operations Console. Policy is provided for all connectors, and the default policy transforms all CA SOI entities so that they are compliant with the USM schema. Additional policy may also be provided for connectors that support create, update, and delete operations in the domain manager product.

The IFW uses an event-based mechanism to retrieve and process CIs, relationships, and alerts from connectors. Therefore, the term *event* in this section refers to any object retrieved from a domain manager's connector (including actual events).

You can write new policy or customize existing policy.

New Policy

All connectors require a policy to process events received from domain managers and sent to the SA Manager. Without a policy, the IFW does not receive instructions about how to process an event, and no processing takes place.

If you create a custom connector for a Level 4, 5, or 6 integration (see page 59), set up a policy to process events from its domain manager and perform operations on events in the domain manager, if the connector supports these operations.

Policy Customization

You can change certain policy settings for processing CIs and events from a domain manager. For example, classification policy may require a new event classification category for each existing class to accommodate an updated integrated product release. Or, you may want to place limits or remove existing limits on the data that an existing connector sends to CA SOI.

For Level 2 integrations (see page 60) using a generic connector, customize a template policy file so that the policy appropriately processes data from the data source. For example, you can customize the policy provided with the SNMP connector to collect and process traps from a mainframe product.

Policy Types

You can create or customize the following types of connector policy:

Outbound from connector policy

Transforms data from the source domain manager format to USM format. Outbound from connector policy converts domain manager data into a format that CA SOI can understand and display. All connectors have outbound from connector policy. Outbound from connector policy files use the following naming convention:

*connectorname*_policy.xml

Example: sampleconnector_policy.xml

Inbound to connector policy

Transforms data from the USM format to the source domain manager format. Inbound to connector policy facilitates inbound operations, where connectors make changes to source domain manager data to synchronize data across domains. Not all connectors have inbound to connector policy, and not all connectors support inbound to connector operations. Inbound from connector policy files use the following naming convention:

*connectorname*_policySB.xml

Example: sampleconnector_policySB.xml

The structure (see page 181), available functions (see page 182), and available operations (see page 185) are the same for both types of policy. The only difference is the source format and the format to which the policy transforms the data.

In the <Field> elements inside the <Format> section, inbound policy has USM properties as input and domain-specific properties as output, whereas outbound policy has domain-specific properties as input and USM properties as output. In the <Field> elements inside the <Write> section, inbound policy lists domain-specific properties to pass to the source domain manager.

Inbound to Connector Policy Considerations

Inbound to connector policy transforms data from the USM format to the format of the source domain manager for inbound to connector operations. Inbound to connector policy is not required and only works under the following circumstances:

- The inbound to connector policy file must contain an <EventClass> definition of SiloDataFilter. Inbound policy transforms data before passing it to the methods that invoke synchronization. The delete() method specifically uses SiloDataFilter as the top level class, so this class must exist in the inbound policy file for the delete operation to work. Include this event class as follows:

```
<EventClass name='SiloDataFilter' >
    <!-- properties are as follows (with example below): -->
    <!-- + id (xs:string) -->
    <!-- + updatedAfter (xs:dateTime) -->
    <!-- + entitytype (xs:string) Item, Relationship, or Alert -->
    <!-- + itemtype (xs:string) -->
    <!-- + recursive (xs:boolean) -->
    <!-- Format>
       <Field output='mdrelementid' input='id' format='{0}' />
    </Format-->
    <Write>
      <Field type='file' name='outfile' properties='*' />
      <Field type='publishcache' properties='id,updatedAfter,
      entitytype,itemtype,recursive' />
    </Write>
</EventClass>
```

- The connector must support inbound to connector operations.

  Note: For information about whether a specific connector supports inbound to connector operations, see the *Connector Guide* for that connector.

  Even if a connector does support inbound to connector operations and already contains an inbound to connector policy file that you want to customize, the connector may only support inbound operations for specific types. Find supported inbound to connector CI types on the connector detail page in the Administration UI. The InboundToConnectorTypes field in the Connector Type Data table contains all supported CI types for inbound to connector operations.

  If you are writing new policy for a custom connector, you must include the appropriate methods for inbound to connector operations (see page 141) when building the connector.

- The connector configuration file must include the appropriate file name for the inbound to connector policy file in the *sbpolicy* attribute.

  Each connector has a configuration file located at SOI_HOME\resources\Configurations.

The following example shows the Sample connector policy file definition in its configuration file:

```
<ImplementationClass
descriptorClass="com.ca.usm.ucf.utils.USMBaseConnectorDescriptor"
name="com.ca.usm.ucf.SampleConnector" policy="sampleconnector_policy.xml"
sbpolicy="sampleconnector_policySB.xml"/>
```

The connector configuration file assumes a connector policy file location of SOI_HOME\resources\Core\Catalogpolicy.

- The connector must have inbound to connector operations enabled.

  The isRemotable control defines whether inbound to connector operations are enabled for a connector. Verify that this control is enabled on the connector detail page in the Connector Controls table. The isRemotable control is enabled by default for most connectors.

- The CA Catalyst Synchronizer must be enabled.

  The CA Catalyst Synchronizer determines when synchronization is required and pushes required changes to the connectors. The Synchronizer is disabled by default, and you must enable it for any inbound to connector operations to occur.

  Note: For information about synchronization and enabling the Synchronizer, see the *Administration Guide*.

Important! CA SOI supports only specific synchronization use cases. You should not enable the Synchronizer except as part of a supported use case. Connectors can support inbound to connector operations and contain inbound to connector policy, but support for the operations and policy may depend on whether they are part of a supported synchronization use case. For more information about synchronization and supported use cases, see the *Administration Guide*. For the most current information about synchronization support, see the *Readme* and *Release Notes* on CA Support Online.

Global Policy

Global policy is an embedded set of policy instructions that handles elements and actions that are common for all connectors. Outbound from connector policy should leverage global policy to avoid having to redefine common elements and actions. Inbound to connector policy should not use global policy.

Leverage global policy in outbound policy files by writing the initial <Catalog> element as follows:

```
<Catalog version='1.0' globalextends='GLOBAL!'>
```

Global policy handles the following common outbound policy items:

- Class name assignment

  Example: <Field output='ClassName' format='ComputerSystem' input=''/>
- Instance name assignment
- The entire <Write> element

Policy Structure and Deployment

Separate XML files hold a connector policy for each connector. The connector policy installs in the following location:

SOI_HOME\resources\Core\CatalogPolicy

You have one outbound policy and one optional inbound policy for each connector.

You encapsulate the entire policy file by a single <Catalog> element. In outbound policy files, this element should include the global policy definition (see page 180).

Event Classes

An event class represents a container for all processing operations related to a certain type of event. The event type is determined by a special property named eventtype. The initial eventtype property is set in the connector-specific code.

The name attribute of each <EventClass> property in the catalog is matched to the eventtype property in the event and the operations of that class are carried out on the event. Next, the <Classify> property changes the eventtype to reference a more specific <EventClass>. Event classes are hierarchical, so you can create subclasses of a base event class to further classify events. For more information, see Classification Policy (see page 186).

You can modify the eventtype property through a connector policy just like any other property in policy files. Changing the eventtype in connector policy files changes the event class to the new eventtype value.

Inbound to connector policy must contain an <EventClass> definition of SiloDataFilter at the end of the file for delete operations to work. For more information, see Inbound to Connector Policy Considerations (see page 179).

Hierarchy and Inheritance

Connector policy is hierarchical, meaning that a child event class inherits all policy operations from a parent class. The following code fragment shows the OPR-DSMEVENT class inheriting all connector policy operations defined in the parent OPR-BASE class:

```
<EventClass name="OPR-BASE")
   <Classify ...../>
   <Filter ...../>
</EventClass>
<EventClass name="OPR-DSMEVENT" extends="OPR-BASE">
   <Parse ...../>
</EventClass>
```

In cases where a parent class and child class have similar operations, the parent operations are enacted first, followed by the child operations. For example, if a parent and child include a parse operation, the parent parsing occurs first, followed by the child parsing. You must understand this rule so that the connector policy you write is processed in the intended order. The following example shows this rule:

```
<EventClass name="OPR-BASE">
   <Parse>
       <Field input="tagA" pattern="^(\w+)-(\w+)$" output="tagA1,tagA2"/>
   </Parse>
</EventClass>
<EventClass name="OPR-DSMEVENT" extends="OPR-BASE">
   <Parse>
       <Field input="tagA1" pattern="^(\d\d)(\w+)$" output="tagA1a,tagA1b"/>
   </Parse>
</EventClass>
```

In this example, assume that tagA="12Buckle-Shoe". The property is parsed by the parent operation, which transforms the property into two properties, tagA1 (12Buckle) and tagA2 (Shoe). Afterwards, the tagA1 value, "12Buckle", is parsed by the child operation into two more separate properties, tagA1a (12) and tagA1b (Buckle).

There are no limits on the levels of inheritance or the number of inheriting children. All connector policy operations support inheritance except for classification.

Property Functions

In connector policy operations, various functions help transform or generate event properties such as date, time, and server name.

Use any functions in the function library for connector policy to convert tags to a specific output. Write these functions enclosed in curly brackets {} as part of the input attribute in a Field element.

You can use these functions in the input attribute of any operation. For example:

```
<Format>
<Field  output='' format='{0}' input='{somefunction(param1)}' />
</Format>
```

Functions with no parameters use the following syntax:

```
{function}
```

Functions with additional parameters use a different syntax as follows:

```
{function([param1,param2,param3])}
```

Examples of the functions that CA SOI includes are as follows:

- Host
  - {localhost}—Returns the local hostname (fully qualified)
  - {ip(propname)}— Dereferences the property (usually a host name) and converts to an IPV4 address
  - {fqdn(propname)}—Dereferences the property (usually an IPV4 address) and converts to a fully qualified domain name
  - {convertHexToMac([propname,-])}—Dereferences the property (a hexadecimal code for a MAC address) and converts to a delimited string using the second parameter delimiter character
- DateTime
  - {xsdateTime(now)}—Returns a datetime stamp formatted in xs:dateTime (yyyy-mm-ddThh:mm:ss-zz:zz)
  - {xsdateTime(propname)}—Dereferences the property (epoch time in seconds or milliseconds) and converts to xs:dateTime format
  - {convertxsdateTime([propname,MMM d yyyy K:mm:ss a])}—Dereferences the property (a datetime formatted string), parses the property according to the second parameter, and converts to xs:dateTime
  - {datetime()}—Generates a date and time string for the current date and time and results in the following output structure: March 12, 2008 1:31:00PM
  - {datetime(propname)}—Generates a date and time string for the date and time represented by an event property value, where the property is a long integer representing some number of seconds since January 1, 1970
  - {timet()}—Generates a long integer representing the current number of seconds since January 1, 1970
  - {timet(propname)}—Generates a long integer representing the number of seconds since January 1, 1970, using the event property value, which must be a date and time string

- Date
  - {xsDate(now)}—Returns a date stamp formatted in xs:date (yyyy-mm-dd-zz:zz)
  - {xsDate(propname)}—Dereferences the property (epoch time in seconds or milliseconds) and converts to xs:date format
  - {convertxsDate([propname,MMM d yyyy K:mm:ss a])}—Dereferences the property (a date time formatted string), parses the property according to the second parameter, and converts to xs:date
- Time
  - {xsTime(now)}—Returns a time stamp formatted in xs:time (hh:mm:ss-zz:zz)
  - {xsTime(propname)}—Dereferences the property (epoch time in seconds or milliseconds) and converts to xs:time format
  - {convertxsTime([propname,MMM d yyyy K:mm:ss a])}—Dereferences the property (a date time formatted string), parses it according to the 2nd parameter, and converts to xs:time
- Array
  - {entry(propname)}—References an entry in an array and returns the first property value in a list. Functions are available for entry1 and entry2 to reference values at different places in an array
- String
  - {replace([propname,ch1,ch2])}—Replaces any character or string in a specified property with another character
    - propname—Specifies the property to search for a character or string to replace
    - ch1—Specifies a character or string to search for and replace in the specified property
    - ch2—Specifies a replacement character or string
  - {toLower(propname)}—Converts uppercase characters in the specified property to lowercase
  - {toUpper(propname)}—Converts lowercase characters in the specified property to uppercase
- Other
  - {uniqueidentifier}—Generates a SQL unique identifier, such as 61CD55D1-F142-2E04-8A2A-9667118CF65E
  - {prepareconsolidatefield}—Parses combinations of event fields entered in consolidation operations

How to Add a Function

The SOI_HOME\resources\Core\Conf\tagfunctions.properties file stores the provided functions. You can create and add new functions to the function library for use in policy files.

Follow these steps:

1. Create a java class or method for the new function.

   Note: The method must return a string and accept an array of strings.

2. Package the Java class into a .jar file and copy the file into the SOI_HOME\resources\Core\Bin directory.
3. Add the method or class to the functions.properties file following the conventions of the other entries.
4. Restart the CA SAM Integration Services service.

Policy Operations

Connector policy consists of operations that provide processing and transformation instructions for events retrieved from a specific domain manager. You can write connector policy using the following operations in the connector policy file for any event (CI, relationship, or alert) in a domain manager:

- Classify
- Parse
- Enrich
- Normalize
- Format
- Evaluate
- Filter
- Write

Note: All policies must use valid USM types, properties, and enumerated values defined in the USM schema documentation (see page 231).

Leverage global policy in outbound from connector policy to automate common operation (such as Write) by writing the initial element in the file as follows:

```
<Catalog version='1.0' globalextends='GLOBAL!'>
```

For more information, see Global Policy (see page 180).

Classify Operation

Classify operations refine an event class from the generic eventtype to more specific classifications. This operation enables specific connector policy to be enacted using different types of events from the same source. The eventtype value searches for a matching <EventClass> in the classification operation to classify an event. Each <EventClass> property can contain several subclasses. For example, you may want to classify events coming from CA NSM into more specific classes according to the source of the event the Event Manager or Event Agent received.

Classify operations do not support inheritance, because classification makes an eventtype property more specific, where inheritance extends more general eventtype properties.

The processing engine traverses all field elements in order until a field is matched, after which no other fields are considered.

Note: Inbound to connector policy must include a specific <EventClass> definition for SiloDataFilter for subsequent delete operations to work. For more information and the detailed syntax for this <EventClass>, see Inbound to Connector Policy Considerations (see page 179).

Classify Property—Refines an EventClass

Classify operations begin with a <Classify> property. The <Classify> property refines an <EventClass>.

This property has the following format:

```
<EventClass name=>
   <Classify>
       <Field input= pattern= output= outval= />
   </Classify>
</EventClass>
```

name

Defines the name of the event class that you are using to create the connector policy.

input

Defines the value of the event property used for the classification.

pattern

Defines the regular expression value that the input value must match for an event to be matched.

output

Defines the assigned value for the outval attribute. Normally, the output is the eventtype.

outval

Defines the value assigned to the output. This is usually a more specific eventtype value.

Example: Classify Windows Event Log events into specific subgroups

The following example separates events received from the Windows Event Log event source that match the specified patterns into two subgroups: SYSLOG-SEC and SYSLOG-APP.

```
<EventClass name="SYSLOG">
   <Classify>
       <Field input="msg" pattern="^Sec.*$" output="eventtype"
       outval=SYSLOG-SEC" />
       <Field input="msg" pattern="^App.*$" output="eventtype"
       outval=SYSLOG-APP" />
   </Classify>
</EventClass>
```

The <Classify> property searches the message text (defined by the "msg" attribute) of events received from the Windows Event Log for words or strings beginning with Sec or App and classifies events that qualify into the more specific SYSLOG-SEC and SYSLOG-APP eventtypes. This property creates new Windows Event Log subgroups for events received from the application and security logs.

Parse Operation

Parse operations split event properties into additional properties using regular expression subgroups. For example, if an event source groups the old and new severity of a metric into one property, you can parse the severities into separate properties to make the information easier to understand.

Parse operations support inheritance, so you can parse operations that were created by parsing operations in higher levels. The framework traverses all parsing operation field elements in order from top to bottom until all field elements are processed. Matches are recorded and processed.

Parse Property—Splits Event Properties

Parse operations begin with a <Parse> property. The <Parse> property splits event properties into additional properties using regular expression subgroups.

This property has the following format:

```
<EventClass name=>
   <Parse>
       <Field input= pattern= output= />
   </Parse>
</EventClass>
```

name

Defines the name of the event class that you are using to create the connector policy.

input

Defines the event property to parse into subgroups.

pattern

Defines the regular expression pattern that the input event property must match for the event to be parsed. The pattern is divided into subgroups designated by parentheses. If the input event property matches the pattern, it is separated into one property for each subgroup.

output

Defines the output properties to assign to the parsed input event property subgroups. The output properties correspond to the regular expression subgroups in the pattern. The first output property is assigned to the first subgroup, the second output is assigned to the second subgroup, and so on. Output property values may be new properties or existing properties.

Example: Parse CA NSM DSM events into additional properties

The following example creates three new properties from one specific event property of events received from CA NSM (through the DSM component) and moves the trap description in the event to the trapdef output property:

```
<EventClass name="OPR-DSMEVENT">
   <Parse>
       <Field input="tagA" pattern="^(.*)-(.*)$" output="tagA1,tagA2" />
       <Field input="tagA1" pattern="^(\d\d)(\w+)$" output="tagA1a,tagA1b" />
       <Field input="tagQ" pattern="^TRAP:(.*)$" output="trapdef" />
   </Parse>
</EventClass>
```

This Parse operation searches the event class OPR-DSMEVENT for events containing the input property "tagA" and parses this property into two separate properties (tagA1, tagA2) by separating the values on either side of a dash. The value parsed into tagA1 is then further parsed into two more separate properties (tagA1a, tagA1b). This operation also takes the TRAP description from events with the "tagQ" property and puts it into the trapdef output property.

Enrich Operation

Enrich operations look up additional properties from an external source using current event property values and create new event properties from the retrieved properties. For example, you may want to enrich an event received from CA NSM with contact information for the resource in WorldView or CMDB Configuration Item (CI) information.

You can write enrich operations based on the following types of transformation:

- Regular expression
- JDBC query
- Java method call
- Command line executable

Enrich operations support inheritance. These operations traverse all enrichment operation field elements in order, from top to bottom, until all field elements are processed. Matches are recorded and processed.

Enrich Property—Create New Event Properties From Retrieved Properties

Enrich operations begin with an <Enrich> property. The <Enrich> property looks up additional properties from an external source using current event property values and creates new event properties from the retrieved properties.

This property has the following format:

```
<Enrich>
   <Field input= type= outputtype= [inputtype= connectionstring= jdbcdriver=
   query= returntype= column=]|[jclass= method=]|[cmdline=] output= />
       [<mapentry mapin= mapout=>]
</Enrich>
```

Note: Only the input, type, outputtype, and output attributes are required for all enrich properties. The other attributes you must use depend on the type of enrich operation you are writing. The type definition specifies the requirements for each enrichment type.

input

Defines a list of properties to enrich with information from external sources.

If are using a single multiple-column property as input, enter the column values in a comma-delimited list using the following format:

`property_column_order`

property

Specifies the event property name.

column

Specifies a column value from the column attribute.

order

Specifies an integer, starting with 0, indicating the order in which the specific column value is returned.

For example, if you are using a users property as input that is made up of column values firstname and lastname, the input property would read as follows:

```
user_firstname_0,user_lastname_1
```

type

Defines the type of enrichment. You can have multiple fields of the same or different types within a single <Enrich> property with no restrictions. The following are available types:

map

Matches and enriches properties using regular expressions. Map enrichment uses mapentry elements that represent an expression and an output to assign to the property if the expression is matched. These elements are read from top to bottom until a property matches an element, after which additional mapentries are not considered. The map type requires the following attributes:

- mapin
- mapout

jdbc

Uses property values as input parameters in a JDBC query to determine an enriched value for the properties. The jdbc type requires the following attributes:

- inputtype
- connectionstring
- jdbcdriver
- query
- returntype

You can return multiple columns and rows from a complex JDBC query and use the column attribute in the policy to identify each of these returned columns uniquely.

Note: The jdbc enrichment type supports only the *pairedlist* output type. Therefore, the returned output value is always a paired comma-delimited list of values. This enrichment type does not support other output types (such as *std* and *list*).

methodcall

Uses property values as input parameters in a Java method call to determine an enriched value for the properties. Properties are treated as strings using this option and the Java method must accept a string array as its only parameter. The methodcall type requires the following attributes:

- jclass
- method

exe

Uses property values as input parameters in an executable to determine an enriched value for the properties. The exe type requires the following attribute:

- cmdline

outputtype

Defines the type of output to return. The Enrich operation can return standard output, a list output, or a paired list output. The following are valid values for this attribute:

std

Indicates that the returned output is a singular value.

ref

Indicates that the given mapout value is a variable that contains the output to return.

list

Indicates that the returned output value is a comma-delimited list of values.

Example: red,blue,green

The resulting properties are referenced as follows:

xxxxxx_y

xxxxxx is the name of the output property (as designated by the output attribute)

y is the index of the list value (where 0 is the first element in the list)

For example, if the output attribute is color, red would be referenced as color_1.

pairedlist

Indicates that the returned output value is a paired comma-delimited list of values.

Example: color,red,size,large,name,fido

The resulting properties are referenced as follows:

xxxxxx_zzzzz

xxxxxx is the name of the output property (as designated by the output attribute)

zzzzz is the name of the returned property in the list (color, size, and name in the example).

For example, if the output attribute is myenrichsource, size in the example (whose value is large) is referenced as myenrichsource_size.

mapin

(map only) Defines a regular expression pattern that compares the input property value.

mapout

(map only) Defines the assigned value to the output property if the input property matches the mapin regular expression. Specify an event property for mapping to the event property's value.

inputtype

(jdbc only) Defines the value types for the input properties. Valid values are any Java primitive types such as int, string, long, and bool.

connectionstring

(jdbc only) Defines a JDBC connection string to a database instance. This string must include the database instance, name, user name, and password. The subsequent JDBC example shows use of a string.

jdbc driver

(jdbc only) Defines the JDBC driver Java class. Write the class for this attribute without the .class extension.

query

(jdbc only) Defines a SQL SELECT query that returns the value to use for the input property.

returntype

(jdbc only) Defines the value type of the value returned from the JDBC query. Valid types are any Java primitive type such as int, string, and bool. If you are using a complex JDBC query that returns multiple values, specify the return type for each value in a comma-delimited list. The order of the list must correspond to the values defined in the column attribute.

column

Defines aliases for returned columns when multiple columns of data are returned from a JDBC query. Specify a comma-delimited list of identifiers for each column value so that each piece of data is uniquely referenced in separate properties. This attribute is only required with a multiple column JDBC query.

jclass

(methodcall only) Defines the Java class full name where you run a method.

method

(methodcall only) Defines the name of the Java method that returns the value used for the input property.

cmdline

(exe only) Defines the command line that includes the full pathname and returns the value for the input property. Use substitution markers ({0}, {1}, and {2}) that are replaced with the input property values.

output

Defines the property that is assigned the output value of the enrich operation. The output property is assigned the following value for each enrichment type:

- For map enrichment, the output property is assigned the value of the mapout attribute.
- For jdbc enrichment, the output property is assigned the return value of the jdbc query.
- For methodcall enrichment, the output property is assigned the return value of the method call.
- For exe enrichment, the output property is assigned the stdout value of the executable.

For all enrichment types, the output property can be a new or existing property.

Example: Enrich city and state output with region information

The following example maps the city and state input properties to one region output property according to regular expressions:

```
<Enrich>
   <Field input="city,state" type="map" output="region" outputtype="std">
       <mapentry mapin="^Cin.*,OH$" mapout="Midwest" />
       <mapentry mapin="^New York.*,NY$" mapout="East" />
   </Field>
</Enrich>
```

The <Enrich> property makes the following searches:

- City and state tags that begin with Cin
- State tag of OH
- Begin with New York
- Have a state tag of NY

This property enriches the tags by adding the appropriate region for each of these locations.

Example: Enrich resource output with a complex JDBC query

The following example enriches an event with multiple columns and rows from a database table. This scenario is similar to the previous example with the name and description being queried from the ca_resource_department_table:

```
<EventClass name="OPR">
   <Enrich>
       <Field input="internal_resourceaddr" inputtype="string" type="jdbc"
       outputtype="pairedlist" column="name,desc,org"
       connectionstring="jdbc:sqlserver://server01;databaseName=mdb;
       user=nsmadmin;password=admin;"
       jdbcdriver="com.microsoft.sqlserver.jdbc.SQLServerDriver"
       query="select name,description,organization_uuid from
       ca_resource_department where id=?"
       returntype="string,string,string" output="department" />
   </Enrich>
</EventClass>
```

The <Enrich> property uses the internal_resourceaddr value as in the previous example. The query returns the corresponding name, description, and organization_uuid. The query also assigns this information to new properties using the defined column attributes as follows:

- Returned name is assigned department_name_0
- Returned description is assigned department_desc_0
- Returned organization is assigned department_org_0

When a query returns multiple rows, the trailing number on the property represents the row number. For example, a second returned row would be labeled as department_name_1, department_desc_1, and department_org_1.

Normalize Operation

Normalize operations transform the syntax of event property values to give values from all sources a uniform nomenclature. For example, you may want to map several similar severity property values (Ok, Success, Good, and so on) to Normal for uniformity purposes.

You can write normalize operations based on the following types of transformation:

- Regular expression
- JDBC query
- Java method call
- Command line executable

Normalize operations support inheritance. The normalize operation process traverses all normalization policy field elements in order from top to bottom until all field elements are processed. Matches are recorded and processed.

Normalize Property—Transforms Event Property Values

Normalize operations begin with a <Normalize> property. Normalize operations transform the syntax of event property values to give values from all sources a uniform nomenclature.

This property has the following format:

```
<Normalize>
   <Field input= type= [inputtype= connectionstring= jdbcdriver= query=
   returntype=]|[jclass= method=]|[cmdline=] output= />
       [<mapentry mapin= mapout=>]
</Normalize>
```

Note: The input, type, and output attributes are required for type in the <Normalize> property. Other required attributes depend on the type of normalize operation you are writing.

input

Defines the list of properties to normalize.

type

Defines the type of normalization to perform. You can have multiple fields of the same or different types within a single Normalize property with no restrictions. The following are available types:

map

Matches and normalizes properties against regular expressions. Map normalization uses mapentry elements that represent an expression and an output for assigning to the property if the expression is matched. These elements are read from top to bottom until a property matches an element, after which additional mapentries are not considered. This type requires use of the following attributes:

- mapin
- mapout

jdbc

Uses property values as input parameters in a JDBC query to determine the normalized value for the properties. This type requires the following attributes:

- inputtype
- connectionstring
- jdbcdriver
- query
- returntype

methodcall

Uses property values as input parameters in a Java method call to determine the normalized value for the properties. Properties are treated as strings using this option and the Java method must accept a string array as its only parameter. This type requires the following attributes:

- jclass
- method

exe

Uses property values as input parameters in an executable to determine the normalized value for the properties. This type requires the following attribute:

- cmdline

mapin

(map only) Defines a regular expression pattern that compares the input property value.

mapout

(map only) Defines the assigned value to the output property if the input property matches the mapin regular expression.

inputtype

(jdbc only) Defines the value types for the input properties. Valid values are any Java primitive types such as int, string, long, and bool.

connectionstring

(jdbc only) Defines a JDBC connection string to a database instance. This string must include the database instance, name, user name, and password. The subsequent JDBC example shows use of a string.

jdbc driver

(jdbc only) Defines the JDBC driver Java class. Write the class for this attribute without the .class extension.

query

(jdbc only) Defines a SQL SELECT query that returns the value to use for the input property.

returntype

(jdbc only) Defines the value type of the value returned from the JDBC query. Valid types are any Java primitive type such as int, string, and bool.

jclass

(methodcall only) Defines the Java class full name where you run a method.

method

(methodcall only) Defines the name of the Java method that returns the value used for the input property.

cmdline

(exe only) Defines the command line that includes the full pathname and returns the value for the input property. Use substitution markers ({0}, {1}, and {2}) that are replaced with the input property values.

output

Defines the property that is assigned the output value of the normalization operation. The output property is assigned the following value for each normalization type:

- For map normalization, the output property is assigned the value of the mapout attribute.
- For jdbc normalization, the output property is assigned the return value of the jdbc query.
- For methodcall normalization, the output property is assigned the return value of the method call.
- For exe normalization, the output property is assigned the stdout value of the executable.

For all normalization types, you can use a new or existing output property.

Example: Normalizing city output by mapping with regular expressions

The following example maps the city and state input properties to one city output property according to regular expressions:

```
<Normalize>
   <Field input="city,state" type="map" output="city">
       <mapentry mapin="^Cin.*,IA$" mapout="Cincinnati" />
       <mapentry mapin="^Cin.*,OH$" mapout="Cincinnati" />
   </Field>
</Normalize>
```

The <Normalize> property searches for city properties that begin with Cin, have a state property of IA or OH, and normalizes these input properties to a city output that reads Cincinnati.

Example: Normalizing vendor output using a jdbc query

The following example finds the vendorid input property and normalizes the property to display the vendor's name using a jdbc query:

```
<Normalize>
   <Field input="vendorid" inputtype="string" type="jdbc"
   connectionstring="jdbc:sqlserver://server01;databaseName=trapdb;
   user=sa;password=sa;"
   jdbcdriver="com.microsoft.sqlserver.jdbc.SQLServerDriver"
   query="select vendorname from traptable where vendorid=?" returntype="string"
   output="vendorname" />
</Normalize>
```

The <Normalize> property searches for vendorid properties and normalizes these properties by running a JDBC query to find the vendor name for each vendorid in a database that contains this information. The property displays the vendor name in place of the vendorid in a new vendorname output property.

Example: Normalizing zip code output using a Java method

The following example finds the city, state, and zip input properties and normalizes this information into one ninedigitzipcode property by running a Java method:

```
<Normalize>
   <Field input="city,state,zip" type="methodcall"
   jclass="com.ca.eventplus.catalog.methods.ZipCode" method="ConvertZip"
   output="ninedigitzip" />
</Normalize>
```

The <Normalize> property normalizes the city, state, and zip input properties into the value of the zip code expressed in nine digits by running a Java method to obtain this value. The property displays the nine-digit zip code in a new output property in place of the input properties.

Example: Normalizing zip code output using a command line executable

The following example finds the city, state, and zip input properties and normalizes this information into one ninedigitzip property by running a command line executable:

```
<Normalize>
   <Field input="city,state,zip" type="exe" cmdline="c:\\normzip.exe {0} {1} {2}"
   output="ninedigitzip" />
</Normalize>
```

The <Normalize> property normalizes the city, state, and zip input properties into the value of the zip code expressed in nine digits by running a command line executable to obtain this value. The executable contains substitution markers that are replaced with each input property value to calculate the nine-digit zip code using this information. The property displays the nine-digit zip code in a new output property in place of the input properties.

Format Operation

Format operations combine property values into a new or existing property using a specified format. You can use format operations to define information in events received from event sources using a new property and adhering to a specified format.

Format operations fully support inheritance. The formatting process traverses all format operation fields in order from top to bottom until all field elements are processed. Any matches are recorded and processed.

Format Property—Define Information and Event Property Format

Format operations begin with a <Format> property. Format operations can define information in events received from event sources using a new property and adhering to a specified format.

This property has the following format:

```
<Format>
   <Field conditional= input= format= output= />
</Format>
```

conditional

(Optional) References an event property whose existence or non-existence determines whether the format operation is performed.

input

Defines an event input property or list of event properties to be output in another property with a new format.

format

Defines the format for the specified input properties. Use substitution markers to indicate each property according to the order in the input attribute.

output

Defines a single output property that is assigned the value of the reformatted input properties.

Example: Format severity event input into a meaningful description

The following example searches for events that indicate a severity change for a resource and reformats this information into a short description:

```
<Format>
   <Field input="resource,hostwork,severity" format="The {0} on machine {1} is
   {2}" output="description" />
</Format>
```

The <Format> property searches for the resource, hostwork, and severity properties that indicates a change in severity for a resource on a host system. The property reformats the input values into a short description and outputs the description to the description property. The format attribute uses substitution markers according to the input properties order in the input attribute to put the property values correctly in the description. An example output description property looks similar to the following:

```
The CPU on machine server01 is Critical
```

Example: Format an assigned property with conditional criteria

The following example assigns the property referenced by township to the output property, but only if township exists. If township does not exist, the property referenced by city is assigned.

```
<Format>
   <Field conditional='township'  input='township' format="{0}}"
output="municipality" />
   <Field conditional='!township'  input='city' format="{0}}" output="municipality"
/>
</Format>
```

Evaluate Operation

Evaluate is the policy operation that enables correlations and actions. It evaluates streams of events against defined rules and runs workflow actions when the rules are met. You can write evaluate operations to automate intelligent actions in response to one or more event conditions that require more acknowledgment or action than a simple resolution.

Some of the examples where you can use the Evaluate operation are as follows:

- You can write a rule that detects if a source (SNMP, application log, and so on) generates three events reporting poor response time in a ten minute interval and creates a new event with a higher severity than the previous events that indicates a consistently poor response time.
- You can write rules to infer a clear event for low-level event sources that do not contain this mechanism. For example, an exception from an application log may not have the ability to clear a previously logged exception. You can write a rule for these exceptions to detect symptoms that the exception has been resolved and send a true clear alert to the event destination.
- You can write rules to associate events being sent to CA SOI as infrastructure alerts with the appropriate CI type. For example, you can write a rule to detect multiple events that indicate a database problem (where one event would not make this clear) and map the events to a Database CI.

You write evaluate operations in policy, but the operations rely on the Drools language, which adheres to a different format and must be inserted in the evaluate operation. For more information about writing Drools event-based rules and workflow actions, see the following page for Drools documentation (version 5):
http://www.jboss.org/drools/documentation.html.

The Event Policies dialog in the Operations Console provides an interface for defining event actions such as correlation, enrichment, filtering, and event creation that are automatically converted to the Drools language. Writing evaluate operations directly in policy files is only necessary if your action is not possible through the Event Policies dialog.

Note: For more information about writing event policies and actions in the Operations Console, see the *Event and Alert Management Best Practices Guide*.

Evaluate Property—Evaluate Events Based on Defined Rules and Run Workflow Actions

Evaluate operations begin with an <Evaluate> property, which has the following basic syntax:

```
<Evaluate>
   <Field input="rule name" output="DRL">
      <!CDATA[
      <Drools rule>
   </Field>
   <Field input="action name" output="DRF">
      <!CDATA[
      <Drools action>
   </Field>
</Evaluate>
```

input

Defines the name of the event rule in the rule section and the name of the corresponding action in the action section.

output

Defines the type of Drools language to output. Use DRL for rules and DRF for actions.

Drools rule

Defines the event rule criteria in the Drools language.

Drools action

Defines the event workflow action to run if the rule criteria are met. A workflow action is not required for every rule if the rule itself can perform the appropriate action.

Example: Detect immediate service shutdown and create a higher severity event

The following example detects when a Windows service shuts down within 30 seconds after starting. These operations are tracked in separate events, so an event rule is required to correlate the events and trigger an appropriate action. In this case, the operation creates a new event to replace the other events with a message and severity that reflects the more serious nature of the situation, and also prints the message to a CSV file. This evaluate operation contains a rule and does not require a separate action.

Important! This is a basic example that is easily configurable using the Event Policies dialog in the Operations Console. You should always use the Event Policies dialog to create evaluate operations for event policies, unless the operation is not supported by the interface. For information about creating more complex Drools rules, see the Drools documentation. For other syntax examples (for example, if you want to create a complex enrichment evaluate operation and need a frame of reference), create event policies from the Event Policies dialog and see the resultant syntax at SA_HOME\resources\EventManagement\Policies.

The rule for this example is as follows:

```
<EventClass name='Alert'>
    <Evaluate>
      <Field input='test drools' output='DRL'>
<![CDATA[
package com.ca.eventplus.catalog;
import com.ca.eventplus.catalog.util.EPEvent;
import java.util.HashMap;
declare EPEvent
  @role(event)
end

rule "test drools"
no-loop true
when
pattrn1 : EPEvent((alertedMdrElementID=="?" && message matches ".**entered the
running state.*") && reEvaluate!="test drools")
pattrn2 : EPEvent((alertedMdrElementID=="?" && message matches ".**entered the
stopped state.*") && reEvaluate!="test drools", this after[-30s,30s] pattrn1)
then
    pattrn1.createEvent("test drools",true,false,pattrn1,pattrn2);
end
]]>
      </Field>
    </Evaluate>
</EventClass>
```

```
<EventClass name='test drools' extends='Alert'>
   <FormatPostN>
      <Field  output='AlertType' format='Quality' input='' />
      <Field  output='Severity' format='Major' input='' />
      <Field conditional='pattern1.AlertedMdrProduct'
      output='AlertedMdrProduct' format='{0}'
      input='pattern1.AlertedMdrProduct' />
      <Field conditional='pattern1.AlertedMdrProdInstance'
      output='AlertedMdrProdInstance' format='{0}'
      input='pattern1.AlertedMdrProdInstance' />
      <Field conditional='pattern1.AlertedMdrElementID'
      output='AlertedMdrElementID' format='{0}'
      input='pattern1.AlertedMdrElementID' />
      <Field output='Message' format='{0}' input='Service crashing' />
      <Field output='MdrProduct' format='{0}' input='pattern1.MdrProduct' />
      <Field output='MdrProdInstance' format='{0}'
      input='pattern1.MdrProdInstance' />
      <Field output='MdrElementID' format='{0}' input='{uniqueidentifier}' />
      <Field output='ReportTimestamp' format='{0}' input='{xsdateTime}' />
      <Field output='OccurrenceTimestamp' format='{0}'
      input='pattern1.OccurrenceTimestamp' />
   </FormatPostN>
</EventClass>
```

Note: This syntax comes from an event policy file created through the Event Policies dialog. If you are manually writing evaluate operations, you would write them directly in a connector policy file integrated with the rest of the connector policy.

The input and output properties define the rule name and output. The Drools rule is embedded in the '![CDATA[' property. The Drools rule contains the following sections:

import

Defines Java methods to import for use in the rule. This declaration must include the EPEvent method, which describes the event properties that the Drools engine can use.

declare EPEvent

Declares EPEvent as an event role, enabling correlation between events.

rule "test drools"

Starts the event rule.

when

Defines the rule criteria. The when clause in this example looks for the following events occurring within thirty seconds of one another:

- An event with a message that contains the text 'entered the running state'
- An event with the same alertedMdrElementID value as the 'pattrn1' event and a message that contains the text 'entered the stopped state'

Note the format of the clause, specifically how it uses the EPEvent method to retrieve and evaluate the properties. Also note the syntax of the clause that defines the time interval between events.

then

Defines the action to run when the criteria in the when clause are met. The then clause in this example creates a new event based on the properties of the correlated events.

<FormatPostN>

Sets the properties for the new event. Note that this syntax uses the Format operation to establish the new event properties, and the event class matches the name of the original rule. The Message and Severity properties have new values that reflect the new event condition, and the other properties use the values from the first event.

Note: Several properties have been omitted from this example.

For additional examples and information about the syntax and requirements of the Drools language version 5, see the following page: http://www.jboss.org/drools/documentation.html.

Filter Operation

Filter operations exclude certain events from further processing. These operations can exclude or include events with a certain event tag value or combinations of tag values. For example, you may not want to exclude the processing of events with a Normal severity.

Filter operations are evaluated in the order entered in the file. The operations adhere to the following conventions:

- If an event matches exclude criteria, the IFW immediately discards the event without evaluating ensuing filter entries.
- If an event matches include criteria, the IFW keeps the event and does not evaluate ensuring entries.
- If an event does not match an entry criteria, the IFW continues to evaluate the filter operations in order through the end of the filter operations.

Assembling ordered combinations of filter criteria offers the flexibility to create complex filters.

You can create filter operations in any of the following ways:

- Directly in connector policy files using the format described in this topic. This method applies specific filters to events received from a specific source.
- In the Operations Console for events using event policies (either on multiple domain managers using the mid-tier connector or on a single source).

Filter Property—Excludes Events from Processing

Filtering operations begin with a <Filter> property. Filter operations exclude certain events from further processing.

This property has the following format:

```
<EventClass name=>
   <Filter>
       <Field type= input= pattern=>
   </Filter>
</EventClass>
```

name

Defines the name of the event class that you are using to create the connector policy.

type

Defines if you want to exclude events matching the pattern. Specify exclude to exclude all events matching the pattern, and specify include to include all events matching the pattern.

input

Defines the event property value that is compared against the pattern attribute for filtering. You can combine multiple properties into one entry using a comma-delimited list. In this scenario, both properties must match their corresponding patterns for the filter criteria to be met.

pattern

Defines the regular expression pattern that the input property must match to trigger the filtering action. Use comma-delimited patterns to correspond to multiple input values. If the pattern is matched for an exclude filter type, the entire event is excluded from further processing and is directly sent to the core Write module for output. If matched for an include filter type, the core includes the event in processing and dispatching.

You can use this attribute on the last filter entry to create a default filter for all events not filtered by other entries. For example, you can enter the regular expression ^.*$ to exclude all events not filtered or included by the preceding entries.

Example: Exclude low severity events

The following example excludes events received from the Windows Event Log with a severity of OK or Normal:

```
<EventClass name="SYSLOG">
   <Filter>
       <Field input="internal_newseverity" pattern="^30$" type="exclude" />
   </Filter>
</EventClass>
```

Example: Include high severity events

The following example includes events received from the Windows Event Log with a severity of Critical and excludes all other severities:

```
<EventClass name="SYSLOG">
   <Filter>
       <Field input="internal_newseverity" pattern="^(70|90)$" type="include" />
       <Field input="internal_newseverity" pattern="^.*$" type="exclude" />
   </Filter>
</EventClass>
```

Example: Create a complex filter

The following example filters Windows Event Log events by including high severity events of a certain class and excluding all other events:

```
<EventClass name="SYSLOG">
   <Filter>
       <Field input="internal_newseverity,internal_resourceclass"
       pattern="^(70|90),Application$" type="include" />
       <Field input="internal_newseverity" pattern="^.*$" type="exclude" />
   </Filter>
</EventClass>
```

The <Filter> property filters events received from the Windows Event Log by doing the following in order:

- Includes all events with a Critical severity with a resourceclass of "Application"
- Excludes all events not explicitly included by a previous entry

Write Operation

Write operations define where an event is sent after processing. You can copy an event into several internal buffers to send an event to multiple destinations, one for each destination.

Write operations support inheritance. The write operation processes only a single Write element that defines where to copy the event.

For outbound from connector policy, do you not have to include a Write element if you leveraged global policy (see page 180), because global policy includes the necessary write operation.

Write Property—Defines Where an Event is Sent

Write operations begin with a <Write> property. The <Write> property defines where an event is sent after processing.

This property has the following format:

```
<Write tagfilter= />
   <Field type="file" name= properties= />
   <Field type="publishcache" properties= />
</Write>
```

tagfilter

Defines a regular expression that, if matched against an event property, removes the property from the event.

type

Defines the type of destination. The provided destinations and their syntax are as follows:

- Publishing Queue: publishcache
- A file for IFW processing: file

properties

Defines the event properties to include in the written output.

Example: Write CIs to a file

The following example writes all normalized CIs to a file called "sampleCIs.txt," which is automatically placed in a location for IFW processing:

```
<Field type="file" name="sampleCIs.txt" properties="/>
```

Connector Policy Examples

For full connector policy examples with explanations for each operation, see Outbound Policy Example (see page 155) and Inbound Policy Examples (see page 157). You can also use the Sample connector policy files as examples of complete outbound and inbound connector policy. Install the Sample connector (see page 128) to make the connector policy files available at SOI_HOME\resources\Core\Catalogpolicy.

Additionally, the policy writing process has been explained with the help of an example (see page 209), providing information about how you can build your policy for the CA Catalyst connector for SNMP.

Example: Writing Connector Policy for the CA Catalyst Connector for SNMP

This section explains the policy writing process with the help of an example. The example describes how you can write your connector policy for the CA Catalyst connector for SNMP. The SNMP connector is a generic connector (Level 2 integration) that receives SNMP traps from any SNMP-capable product, transforms the traps, and maps the generic SNMP attributes to the USM model. Each domain manager (integrating product) requires you to write a separate connector policy.

The SNMP connector ships a sample policy file that you can customize.

Integrating with CA Workload Automation through the SNMP Connector

For this example, the domain manager that sends SNMP traps to the SNMP connector is CA Workload Automation. For CA Workload Automation, any job scheduled on the specific agent host is a CI and is mapped to the USM type ITActivity.

Note: This example assumes that you have configured CA Workload Automation to forward traps to the SNMP connector system on the port that you specified during connector installation.

How to Write the Example Connector Policy

The policy writing process for this example includes the following steps:

1. Understand the SNMP trap received from the domain manager (see page 210).
2. Understand the addition of the eventtype properties (Item and Alert) to the received SNMP trap (see page 210).
3. Add the event classes (Item, Alert, and USM-Entity) to the policy file (see page 212).
4. Write appropriate policy operations for the specified event classes (see page 213):
   - Policy operations for the Item event class (see page 213)
   - Policy operations for the Alert event class (see page 219)

A special property named eventtype determines each event type in the sample policy. Connector policy matches the name attribute of each <EventClass> property in the policy file to the eventtype property in the event and performs the policy operations for that class on the event. An event class represents a container for all processing operations related to a specific type of event.

Understand the SNMP Trap Received from the Domain Manager

The structure of the SNMP trap that the SNMP connector receives from its domain manager is as follows. Note that varbinds and varbindvals are represented as comma-separated values:

```
snmp_agent="155.35.37.142"
snmp_community="public"
snmp_varbindvals="Manager,ESP_COE-NE-DSWA-D1_7500,MAIN,OrderEntry DB Backup
Workflow,23,DATA_TRANSFER_JOB,,READY,Automated Test Service,This is my
description"
snmp_ticks="0"
snmp_genericTrap="6"
snmp_varbindoid="1.3.6.1.4.1.11203.1,1.3.6.1.4.1.11203.3,1.3.6.1.4.1.11203.11
,1.3.6.1.4.1.11203.7,1.3.6.1.4.1.11203.8,1.3.6.1.4.1.11203.9,1.3.6.1.4.1.1120
3.10,1.3.6.1.4.1.11203.6,1.3.6.1.4.1.11203.5"
snmp_requestID="0"
snmp_errorStatus="Success"
snmp_enterprise="1.3.6.1.4.1.11203"
snmp_errorIndex="0"
snmp_specificTrap="1"
```

After receiving SNMP traps from the domain manager, the SNMP connector converts them as data objects and produces the output in name/value pair format. The connector policy must use these name/value pairs as its input.

Understand the Addition of the Eventtype Property to the Trap

The SNMP connector adds the eventtype property to the received trap before sending it to policy for transformation. The SNMP connector receives only traps with no CIs available for these traps. Therefore, when the connector receives a trap for the first time, it explicitly creates a CI based on the trap. The connector first sends the trap as a CI with eventtype as Item. It then sends the same trap as an alert but with eventtype as Alert. Subsequently, if the same trap is received again, the connector checks its internal cache and verifies that the trap is already available as a CI, it then sends the trap only as an alert.

For example, if CA Workload Automation sends a trap indicating that a job failed, the connector first creates a CI based on the trap for the job, and then creates an alert associated with that CI representing the job failure. The connector policy uses the trap sent by the connector with an eventtype of Item to create the CI and the trap with the eventtype of Alert to create the alert. If another trap is sent related to the same job, the connector detects the relation and only sends a trap with the Alert eventtype.

You must create a policy for both the Item (see page 211) and Alert (see page 211) eventtypes.

Item Eventtype

For the Item eventtype, the connector adds eventtype=Item to the trap information as follows:

```
snmp_agent="155.35.37.142"
snmp_community="public"
snmp_varbindvals="Manager,ESP_COE-NE-DSWA-D1_7500,MAIN,OrderEntry DB Backup
Workflow,23,DATA_TRANSFER_JOB,,READY,Automated Test Service,This is my
description"
snmp_ticks="0"
snmp_genericTrap="6"
snmp_varbindoid="1.3.6.1.4.1.11203.1,1.3.6.1.4.1.11203.3,1.3.6.1.4.1.11203.11
,1.3.6.1.4.1.11203.7,1.3.6.1.4.1.11203.8,1.3.6.1.4.1.11203.9,1.3.6.1.4.1.1120
3.10,1.3.6.1.4.1.11203.6,1.3.6.1.4.1.11203.5"
snmp_requestID="0"
snmp_errorStatus="Success"
snmp_enterprise="1.3.6.1.4.1.11203"
snmp_errorIndex="0"
snmp_specificTrap="1"
eventtype = Item
```

Alert Eventtype

For the Alert eventtype, the connector adds eventtype=Alert to the trap information as follows:

```
snmp_agent="155.35.37.142"
snmp_community="public"
snmp_varbindvals="Manager,ESP_COE-NE-DSWA-D1_7500,MAIN,OrderEntry DB Backup
Workflow,23,DATA_TRANSFER_JOB,,READY,Automated Test Service,This is my
description"
snmp_ticks="0"
snmp_genericTrap="6"
snmp_varbindoid="1.3.6.1.4.1.11203.1,1.3.6.1.4.1.11203.3,1.3.6.1.4.1.11203.11
,1.3.6.1.4.1.11203.7,1.3.6.1.4.1.11203.8,1.3.6.1.4.1.11203.9,1.3.6.1.4.1.1120
3.10,1.3.6.1.4.1.11203.6,1.3.6.1.4.1.11203.5"
snmp_requestID="0"
```

```
snmp_errorStatus="Success"
snmp_enterprise="1.3.6.1.4.1.11203"
snmp_errorIndex="0"
snmp_specificTrap="1"
eventtype = Alert
```

Add the Event Classes

Connector policy matches the name attribute of each <EventClass> property in the policy file to the eventtype property in the event and performs the policy operations for that class on the event.

For this example, you add the following event classes to the policy file:

- Item (see page 212)
- Alert (see page 212)
- USM-Entity (see page 213)

Add the Item Event Class

For the Item eventtype, add an event class with its name as Item in the policy file so the policy can create CIs based on received CA Workload Automation traps.

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!-- ======Event Class====== -->
    <EventClass name="Item">
    </EventClass>
</Catalog>
```

All applicable policy operations for the Item eventtype are explained in the Policy Operations for the Item Event Class (see page 213) section.

Add the Alert Event Class

For the Alert eventtype, you must add an event class with its name as Alert to the same policy file beneath the already defined event classes. You must define policy under the Alert eventtype to process the received traps that reach the policy defined as alerts, so that they can appear on the Operations Console associated with the CI of the trap processed using the Item eventtype policy.

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!--  ======Event Class======   -->
    <EventClass name="Alert">
    </EventClass>
</Catalog>
```

Alert is a separate event class that does not extend the previously defined Item event class. All applicable policy operations for the Alert eventtype are explained in the Policy Operations for the Alert Event Class (see page 219) section.

Add the USM-Entity Event Class

For the USM-Entity eventtype, you must add an event class with its name as USM-Entity at the bottom of the policy file. The USM-Entity eventtype publishes the transformed CIs and Alerts.

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!--  ======Event Class======   -->
    <EventClass name="USM-Entity" />
    </EventClass>
</Catalog>
```

Write Policy Operations

For this example, you write appropriate policy operations for the Item (see page 213) and Alert (see page 219) event classes. You do not need to write any policy operation for the USM-Entity event class.

Policy Operations for the Item Event Class

The following policy operations are added to the policy file for the Item eventtype:

- Classify (see page 213)
- Parse (see page 214)
- Normalize (see page 215)
- Format (see page 217)

Classify

You must classify traps that reach the policy as an Item eventtype to specific CI types. To do this, you must choose a property from the connector data output that you can use for classifying the CI. For this example, snmp_varbindoids has been chosen as the classifier. Write the Classify section of the sample policy as follows:

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!-- ======Event Class====== -->
    <EventClass name="Item">
    <!-- Classify -->
    <Classify>
        <Field input="snmp_varbindoids" pattern=".*1\.3\.6\.1\.4\.1\.11203\.9.*$"
        output="eventtype" outval="ITActivity" />
    </Classify>
    </EventClass>
</Catalog>
```

This policy classifies traps defined under the Item eventtype as belonging to the ITActivity CI type when the value of the snmp_varbindoids property contains the pattern 1.3.6.1.4.1.11203.9.

For example, if the snmp_varbindoids value is 1.3.6.1.4.1.11203.1,1.3.6.1.4.1.11203.3,1.3.6.1.4.1.11203.11,1.3.6.1.4.1.11203.7,1.3.6.1.4.1.11203.8,**1.3.6.1.4.1.11203.9**,1.3.6.1.4.1.11203.10,1.3.6.1.4.1.11203.6,1.3.6.1.4.1.11203.5, the trap is classified to ITActivity, because it contains the specified 1.3.6.1.4.1.11203.9 pattern.

By default, all received traps in this example are considered to be of type ITActivity. Most detailed integrations require you to define multiple classification rules for multiple CI types (for example, ComputerSystem, Process, Database, and so on). Each CI type requires its own specialized processing operations. Because this example classifies all traps under one CI type, you define the ITActivity class as follows:

```
    <Catalog version="1.0" globalextends="GLOBAL!">
    <!-- ======Event Class====== -->
    <EventClass name="Item">
    <!-- Classify -->
    <Classify>
        <Field input="snmp_varbindoids" pattern=".*1\.3\.6\.1\.4\.1\.11203\.9.*$"
        output="eventtype" outval="ITActivity" />
    </Classify>
    </EventClass>
    <!--  ======Event Class======   -->
    <EventClass name="ITActivity" extends="Item">
    </EventClass>
</Catalog>
```

All processing operations specific to the ITActivity CI type occur within this EventClass definition.

Parse

In the received CA Workload Automation traps, a single property, snmp_varbindvals, represents various individual properties (such as nodetype, nodename, state, and so on) as comma-separated values. To isolate all of these individual property values from the snmp_varbindvals property so that you can map them to individual USM properties, you must use the Parse operation. Parse operations split event properties into additional properties using regular expression subgroups.

For this example, the trap varbind data is parsed into meaningful properties in the policy file that can later be mapped to USM properties. Write the Parse section for the Item event class as follows:

Note: The ITActivity class inherits the same parse information from the parent Item class.

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!-- ======Event Class====== -->
    <EventClass name="Item">
    <!-- Classify -->
```

```
    <Classify>
        <Field input="snmp_varbindoids" pattern=".*1\.3\.6\.1\.4\.1\.11203\.9.*$"
        output="eventtype" outval="ITActivity" />
    </Classify>
    <Parse>
        <Field input="snmp_varbindvals"
        output="temp_nodetype,temp_nodename,temp_domain,temp_applname,temp_applge
        n,temp_jobname,temp_jobequal,temp_state,temp_status"
        pattern="^(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?)$" />
    </Parse>
    </EventClass>
    <!--  ======Event Class======   -->
    <EventClass name="ITActivity" extends="Item">
    </EventClass>
</Catalog>
```

This policy takes the received comma-separated snmp_varbindvals property values and assigns them to individual property names according to their order of occurrence. For example, the first value is assigned to the temp_nodetype property, and second value is assigned to the temp_nodename property, and so on. After the value field is parsed and available, you can use the same information for uniquely identifying a specific CI or alert.

Normalize

In the parsing (see page 214) section, the job state property temp_state that you created from a specific trap varbind value represents the activity state of the trap. The activity state for CA Workload Automation traps can be one of the following:

- Unknown
- Complete
- Monitor
- Exec
- Failed
- Premature End
- Inactive
- Overdue
- Suberror
- Agent Down
- Ready
- Abandon Submission

You must map these activity states to the CA SOI activity states using the Normalize operation. Normalize operations transform the syntax of event property values to give values from all sources a uniform nomenclature. For performing these mappings, write the Normalize section as follows:

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!-- ======Event Class====== -->
    <EventClass name="Item">
    <!-- Classify -->
    <Classify>
        <Field input="snmp_varbindoids" pattern=".*1\.3\.6\.1\.4\.1\.11203\.9.*$"
        output="eventtype" outval="ITActivity" />
    </Classify>
    <Parse>
        <Field input="snmp_varbindvals"
        output="temp_nodetype,temp_nodename,temp_domain,temp_applname,temp_applge
        n,temp_ jobname,temp_jobequal,temp_state,temp_status"
        pattern="^(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?)$" />
    </Parse>
    </EventClass>
    <!--  ======Event Class======   -->
    <EventClass name="ITActivity" extends="Item">
    <Normalize>
        <Field input="temp_state" type="map" output="activityState">
        <mapentry mapin="[Uu][Nn][Kk][Nn][Oo][Ww][Nn]" mapout="Unknown" />
        <mapentry mapin="[Cc][Oo][Mm][Pp][Ll][Ee][Tt][Ee]" mapout="Finished" />
        <mapentry mapin="[Mm][Oo][Nn][Ii][Tt][Oo][Rr]" mapout="Normal" />
        <mapentry mapin="[Ee][Xx][Ee][Cc]" mapout="Normal-Running" />
        <!--  Informational  -->
        <mapentry mapin="[Ff][Aa][Ii][Ll][Ee][Dd]" mapout="Trouble" />
        <mapentry mapin="[Pp][Rr][Ee][Mm][Aa][Tt][Uu][Rr][Ee] [Ee][Nn][Dd]"
        mapout="Obstructed" />
        <mapentry mapin="[Ii][Nn][Aa][Cc][Tt][Ii][Vv][Ee]" mapout="Normal-Waiting"
        />
        <mapentry mapin="[Oo][Vv][Ee][Rr][Dd][Uu][Ee]" mapout="Obstructed" />
        <mapentry mapin="[Ss][Uu][Bb][Ee][Rr][Rr][Oo][Rr]"
        mapout="Finished-Completed" />
        <mapentry mapin="[Aa][Gg][Ee][Nn][Tt] [Dd][Oo][Ww][Nn]" mapout="Trouble" />
        <!--  Fatal   -->
        <mapentry mapin="[Rr][Ee][Aa][Dd][Yy]" mapout="Normal" />
        <mapentry mapin="[Aa][Bb][Aa][Nn][Dd][Oo][Nn]
        [Ss][Uu][Bb][Mm][Ii][Ss][Ss][Ii][Oo][Nn]" mapout="Finished-Abandoned" />
        </Field>
    </Normalize>
    </EventClass>
</Catalog>
```

This policy maps all potential values of the temp_state property to activity states that CA SOI can understand. For example, Complete maps to Finished, Failed maps to Trouble, Inactive maps to Normal-Waiting, and so on. Now, when you map this property value to the activityState USM property, it can understand and display the supported activity state values.

Format

Format operations combine property values into a new or existing property using a specified format. You can use format operations to define information in events received from event sources using a new property and adhering to a specified format.

For this example, you transform the source properties and the temporary properties that you create using the Parse operation into USM properties as follows:

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!-- ======Event Class====== -->
    <EventClass name="Item">
    <!-- Classify -->
    <Classify>
        <Field input="snmp_varbindoids" pattern=".*1\.3\.6\.1\.4\.1\.11203\.9.*$"
        output="eventtype" outval="ITActivity" />
    </Classify>
    <Parse>
        <Field input="snmp_varbindvals"
        output="temp_nodetype,temp_nodename,temp_domain,temp_applname,temp_applge
        n,temp_ jobname,temp_jobequal,temp_state,temp_status"
        pattern="^(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?)$" />
    </Parse>
    <Format>
        <!--  Non-Correlatable properties   -->
        <Field output="MdrElementID" format="{0}" input="AlertedMdrElementID" />
    </Format>
```

This policy assigns the mandatory MdrElementID property to the CI based on the value of the AlertedMdrElementID in the alert created from the same trap. You assign the AlertedMdrElementID value in the format section for alerts. This value determines whether a CI is created for the trap. If the MdrElementID exists, the CI creation does not occur, and only an alert is created for the trap. For more information about assigning values for the AlertedMdrElementID property, see Format (see page 221) for the alert type.

```
    </EventClass>
    <!--  ======Event Class======   -->
    <EventClass name="ITActivity" extends="Item">
    <Normalize>
        <Field input="temp_state" type="map" output="activityState">
        <mapentry mapin="[Uu][Nn][Kk][Nn][Oo][Ww][Nn]" mapout="Unknown" />
        <mapentry mapin="[Cc][Oo][Mm][Pp][Ll][Ee][Tt][Ee]" mapout="Finished" />
```

```
            <mapentry mapin="[Mm][Oo][Nn][Ii][Tt][Oo][Rr]" mapout="Normal" />
            <mapentry mapin="[Ee][Xx][Ee][Cc]" mapout="Normal-Running" />
            <!--  Informational  -->
            <mapentry mapin="[Ff][Aa][Ii][Ll][Ee][Dd]" mapout="Trouble" />
            <mapentry mapin="[Pp][Rr][Ee][Mm][Aa][Tt][Uu][Rr][Ee] [Ee][Nn][Dd]"
            mapout="Obstructed" />
            <mapentry mapin="[Ii][Nn][Aa][Cc][Tt][Ii][Vv][Ee]" mapout="Normal-Waiting"
            />
            <mapentry mapin="[Oo][Vv][Ee][Rr][Dd][Uu][Ee]" mapout="Obstructed" />
            <mapentry mapin="[Ss][Uu][Bb][Ee][Rr][Rr][Oo][Rr]"
            mapout="Finished-Completed" />
            <mapentry mapin="[Aa][Gg][Ee][Nn][Tt] [Dd][Oo][Ww][Nn]" mapout="Trouble" />
            <!--  Fatal   -->
            <mapentry mapin="[Rr][Ee][Aa][Dd][Yy]" mapout="Normal" />
            <mapentry mapin="[Aa][Bb][Aa][Nn][Dd][Oo][Nn]
            [Ss][Uu][Bb][Mm][Ii][Ss][Ss][Ii][Oo][Nn]" mapout="Finished-Abandoned" />
            </Field>
        </Normalize>
        <Format>
            <!-- Correlatable properties, must populate at least one -->
            <Field output="ActivityID" format="{0}" input="temp_jobname" />
            <Field conditional="snmp_agent" output="DeviceIPV4Address" format="{0}"
            input="snmp_agent" />
            <!-- verify that at least one property is set -->
            <Field conditional="DeviceIPV4Address" output="temp_atleastoneset"
            format="{0}" input="DeviceIPV4Address" />
            <Field conditional="!temp_atleastoneset" output="Error" format="At least
            one correlatable property is not set" input="" />
            <!-- Non-Correlatable properties  -->
            <Field output="RuntimeName" format="{0}.{1}.{2}"
            input="temp_applname,temp_applgen,temp_jobname" />
            <Field output="RuntimeDiscriminator" format="{0}" input="temp_applgen" />
            <Field output="DefinitionName" format="{0}.{1}"
            input="temp_applname,temp_jobname" />
            <Field output="ActivityTypes" format="Job" input="" />
            <Field conditional="activityState" output="ActivityState" format="{0}"
            input="activityState" />
            <Field conditional="temp_state" output="StateDescription" format="{0}
            present status is {1}" input="temp_jobname,temp_state" />
        </Format>
        </EventClass>
    </Catalog>
```

This policy does the following:

- Ensures that at least one of the mandatory correlatable properties (as defined in the USM schema for the ITActivity type) is set and includes values. It maps the temp_jobname property that you created during the parse operation to the ActivityID USM property. You must ensure that at least one correlatable property is populated for every type in your policy.
- Inserts a control that creates an error if one of the correlatable properties is not populated.
- Uses the temporary properties you created in the parse section to populate key USM properties for the ITActivity type. For example, the first entry in the Non-Correlatable Properties section combines the values of three temporary properties to populate the RuntimeName USM property with each value separated by a period.
- Note the use of the conditional attribute, which only enacts the associated operation if the input property has a value.

Policy Operations for the Alert Event Class

The following operations are added to the policy file for the Alert eventtype:

- Parse (see page 219)
- Normalize (see page 220)
- Format (see page 221)

You add these operations to the same policy file where you have already included the Item eventtype information directly under the Alert event class definition.

Parse

You must parse the trap data for alerts similar to the way you did for trap-based CIs (see page 214). Separating the varbind values lets you isolate the values for individual use in other sections of policy. For the Alert eventtype, add the parse operation to the Alert event class section of the policy file as follows:

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!--  ======Event Class======   -->
    <EventClass name="Alert">
    <Parse>
        <Field input="snmp_varbindvals"
        output="temp_nodetype,temp_nodename,temp_domain,temp_applname,temp_applge
        n,temp_jobname,temp_jobequal,temp_state,temp_status"
        pattern="^(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?)$" />
    </Parse>
    </EventClass>
</Catalog>
```

This policy takes the received comma-separated snmp_varbindvals property values and assigns them to individual property names according to their order of occurrence. For example, the first value is assigned to the temp_nodetype property, and second value is assigned to the temp_nodename property, and so on. After the value field is parsed and available, you can use the same information for uniquely identifying a specific alert.

Normalize

In the parsing (see page 219) section, the job state property temp_state that you created from a specific trap varbind value represents the severity level of the trap. The severity for CA Workload Automation traps may be one of the following:

- Unknown
- Complete
- Monitor
- Exec
- Failed
- Premature End
- Inactive
- Overdue
- Suberror
- Agent Down
- Ready
- Abandon Submission

You must map these severities to the CA SOI severity property using the Normalize operation. Normalize operations transform the syntax of event property values to give values from all sources a uniform nomenclature. For performing these mappings, write the Normalize section as follows:

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!--  ======Event Class======   -->
    <EventClass name="Alert">
    <Parse>
        <Field input="snmp_varbindvals"
        output="temp_nodetype,temp_nodename,temp_domain,temp_applname,temp_applge
        n,temp_jobname,temp_jobequal,temp_state,temp_status"
        pattern="^(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?)$" />
    </Parse>
    <Normalize>
        <Field input="temp_state" type="map" output="severity">
        <mapentry mapin="[Uu][Nn][Kk][Nn][Oo][Ww][Nn]" mapout="Unknown" />
        <mapentry mapin="[Cc][Oo][Mm][Pp][Ll][Ee][Tt][Ee]" mapout="Normal" />
        <mapentry mapin="[Mm][Oo][Nn][Ii][Tt][Oo][Rr]" mapout="Normal" />
```

```
            <mapentry mapin="[Ee][Xx][Ee][Cc]" mapout="Normal" />
            <!--  Informational   -->
            <mapentry mapin="[Ff][Aa][Ii][Ll][Ee][Dd]" mapout="Critical" />
            <mapentry mapin="[Pp][Rr][Ee][Mm][Aa][Tt][Uu][Rr][Ee] [Ee][Nn][Dd]"
            mapout="Critical" />
            <mapentry mapin="[Ii][Nn][Aa][Cc][Tt][Ii][Vv][Ee]" mapout="Minor" />
            <mapentry mapin="[Oo][Vv][Ee][Rr][Dd][Uu][Ee]" mapout="Major" />
            <mapentry mapin="[Ss][Uu][Bb][Ee][Rr][Rr][Oo][Rr]" mapout="Major" />
            <mapentry mapin="[Aa][Gg][Ee][Nn][Tt] [Dd][Oo][Ww][Nn]" mapout="Critical"
            />
            <!--  Fatal   -->
            <mapentry mapin="[Rr][Ee][Aa][Dd][Yy]" mapout="Normal" />
            <mapentry mapin="[Aa][Bb][Aa][Nn][Dd][Oo][Nn]
            [Ss][Uu][Bb][Mm][Ii][Ss][Ss][Ii][Oo][Nn]" mapout="Unknown" />
            </Field>
        </Normalize>
        </EventClass>
</Catalog>
```

This policy maps all potential values of the temp_state property to severities that CA SOI can understand. For example, Complete maps to Normal, Failed maps to Critical, Inactive maps to Major, and so on. Now, when you map this property value to the Severity USM property, it can understand and display the supported severity values.

Format

Format operations combine property values into a new or existing property using a specified format. You can use format operations to define information in events received from event sources using a new property and adhering to a specified format.

For this example, you transform the source properties and the temporary properties that you create using the Parse (see page 219) operation into USM properties as follows:

Note: Alerts do not have correlatable properties.

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!--  ======Event Class======   -->
    <EventClass name="Alert">
    <Parse>
        <Field input="snmp_varbindvals"
        output="temp_nodetype,temp_nodename,temp_domain,temp_applname,temp_applge
        n,temp_jobname,temp_jobequal,temp_state,temp_status"
        pattern="^(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?)$" />
    </Parse>
    <Normalize>
        <Field input="temp_state" type="map" output="severity">
        <mapentry mapin="[Uu][Nn][Kk][Nn][Oo][Ww][Nn]" mapout="Unknown" />
```

```
        <mapentry mapin="[Cc][Oo][Mm][Pp][Ll][Ee][Tt][Ee]" mapout="Normal" />
        <mapentry mapin="[Mm][Oo][Nn][Ii][Tt][Oo][Rr]" mapout="Normal" />
        <mapentry mapin="[Ee][Xx][Ee][Cc]" mapout="Normal" />
        <!--  Informational   -->
        <mapentry mapin="[Ff][Aa][Ii][Ll][Ee][Dd]" mapout="Critical" />
        <mapentry mapin="[Pp][Rr][Ee][Mm][Aa][Tt][Uu][Rr][Ee] [Ee][Nn][Dd]"
        mapout="Critical" />
        <mapentry mapin="[Ii][Nn][Aa][Cc][Tt][Ii][Vv][Ee]" mapout="Minor" />
        <mapentry mapin="[Oo][Vv][Ee][Rr][Dd][Uu][Ee]" mapout="Major" />
        <mapentry mapin="[Ss][Uu][Bb][Ee][Rr][Rr][Oo][Rr]" mapout="Major" />
        <mapentry mapin="[Aa][Gg][Ee][Nn][Tt] [Dd][Oo][Ww][Nn]" mapout="Critical"
        />
        <!--  Fatal   -->
        <mapentry mapin="[Rr][Ee][Aa][Dd][Yy]" mapout="Normal" />
        <mapentry mapin="[Aa][Bb][Aa][Nn][Dd][Oo][Nn]
        [Ss][Uu][Bb][Mm][Ii][Ss][Ss][Ii][Oo][Nn]" mapout="Unknown" />
        </Field>
    </Normalize>
    <Format>
        <!-- Correlatable properties, must populate at least one  -->
        <!--  Non-Correlatable properties  -->
        <Field output="MdrElementID" format="alert-{0}:{1}:{2}"
        input="snmp_agent,temp_applname,temp_jobname" />
        <Field output="OccurrenceTimestamp" format="{0}" input="{xsdateTime(now)}"
        />
        <Field output="ReportTimestamp" format="{0}" input="{xsdateTime(now)}" />
        <Field output="AlertType" format="{0}" input="Risk-Fault" />
        <Field conditional="severity" output="Severity" format="{0}"
        input="severity" />
        <Field output="AlertedMdrProduct" format="CA:00036" input="" />
        <Field output="AlertedMdrProdInstance" format="{0}"
        input="{fqdn(snmp_agent)}" />
        <!--  Assign instance name   -->
        <Field conditional="snmp_agent" output="Section1" format="{0}"
        input="snmp_agent" />
        <Field conditional="!snmp_agent" output="Flag" format="false" input="" />
        <Field conditional="temp_applname" output="Section2" format="{0}"
        input="temp_applname" />
        <Field conditional="!temp_applname" output="Flag" format="false" input=""
        />
        <Field conditional="temp_jobname" output="Section3" format="{0}"
        input="temp_jobname" />
        <Field conditional="!temp_jobname" output="Flag" format="false" input="" />
        <Field conditional="Flag" output="AlertedMdrElementID" format="" input=""
        />
        <Field conditional="!Flag" output="AlertedMdrElementID"
        format="{0}:{1}:{2}" input="Section1,Section2,Section3" />
        <Field output="Summary" format="{0}" input="temp_status" />
```

```
        <Field conditional="temp_state" output="Message" format="{0} alert on {1}
        scheduled on host {2}" input="temp_state,temp_jobname,snmp_agent" />
        <Field output="MetricName" format="{0}" input="Job Status" />
        <Field output="MetricType" format="{0}" input="Unknown" />
        <Field output="MetricUnitDefinition" format="{0}" input="Number" />
        <Field output="MetricDataType" format="{0}" input="String" />
    </Format>
    </EventClass>
</Catalog>
```

This policy does the following:

- Populates all mandatory alert USM properties as defined in the USM schema with values from source trap properties or temporary properties you created in the parse section.
- Note the use of functions to populate several mandatory properties. For example, a function populates the OccurrenceTimestamp property with the current time, and the AlertedMdrProdInstance value is derived by converting the value of the snmp_agent property to a fully qualified domain name.
- Note how some properties are assigned hard-coded values instead of the value of source properties. For example, the AlertType property is assigned a static value of Risk-Fault.
- Populates the AlertedMdrElementID value, or instance name, using agent IP address, application name, and job name. The policy uses conditional attributes to ensure that each source property has a value before using it to determine the AlertedMdrElementID. AlertedMdrElementID serves as the key value that uniquely identifies an alert and associated CI by its associated agent, application, and job. This value becomes the MdrElementID value for the CI created by the trap, and if the same MdrElementID value exists, the policy does not create a CI, and instead creates only the alert and assigns it to the existing CI with the same MdrElementID value.

  Note: The key values that you use to define the AlertedMdrElementID (and by extension, MdrElementID) value differ depending on the trap source you are integrating. The property must contain a value or combination of values that can uniquely identify a CI and alert from your trap source.
- Populates the alert Summary property with the value of the temporary temp_status property created in the parse section.

Completed Example Policy File

This section includes all the policy sections that you created for this example:

```
<Catalog version="1.0" globalextends="GLOBAL!">
    <!-- ======Event Class====== -->
    <EventClass name="Item">
    <!-- Classify -->
```

```
<Classify>
    <Field input="snmp_varbindoids" pattern=".*1\.3\.6\.1\.4\.1\.11203\.9.*$"
    output="eventtype" outval="ITActivity" />
</Classify>
<Parse>
    <Field input="snmp_varbindvals"
    output="temp_nodetype,temp_nodename,temp_domain,temp_applname,temp_applge
    n,temp_ jobname,temp_jobequal,temp_state,temp_status"
    pattern="^(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?)$" />
</Parse>
<Format>
    <!--  Non-Correlatable properties   -->
    <Field output="MdrElementID" format="{0}" input="AlertedMdrElementID" />
</Format>
</EventClass>
<!--  ======Event Class======   -->
<EventClass name="ITActivity" extends="Item">
<Normalize>
    <Field input="temp_state" type="map" output="activityState">
    <mapentry mapin="[Uu][Nn][Kk][Nn][Oo][Ww][Nn]" mapout="Unknown" />
    <mapentry mapin="[Cc][Oo][Mm][Pp][Ll][Ee][Tt][Ee]" mapout="Finished" />
    <mapentry mapin="[Mm][Oo][Nn][Ii][Tt][Oo][Rr]" mapout="Normal" />
    <mapentry mapin="[Ee][Xx][Ee][Cc]" mapout="Normal-Running" />
    <!--  Informational  -->
    <mapentry mapin="[Ff][Aa][Ii][Ll][Ee][Dd]" mapout="Trouble" />
    <mapentry mapin="[Pp][Rr][Ee][Mm][Aa][Tt][Uu][Rr][Ee] [Ee][Nn][Dd]"
    mapout="Obstructed" />
    <mapentry mapin="[Ii][Nn][Aa][Cc][Tt][Ii][Vv][Ee]" mapout="Normal-Waiting"
    />
    <mapentry mapin="[Oo][Vv][Ee][Rr][Dd][Uu][Ee]" mapout="Obstructed" />
    <mapentry mapin="[Ss][Uu][Bb][Ee][Rr][Rr][Oo][Rr]"
    mapout="Finished-Completed" />
    <mapentry mapin="[Aa][Gg][Ee][Nn][Tt] [Dd][Oo][Ww][Nn]" mapout="Trouble" />
    <!--  Fatal   -->
    <mapentry mapin="[Rr][Ee][Aa][Dd][Yy]" mapout="Normal" />
    <mapentry mapin="[Aa][Bb][Aa][Nn][Dd][Oo][Nn]
    [Ss][Uu][Bb][Mm][Ii][Ss][Ss][Ii][Oo][Nn]" mapout="Finished-Abandoned" />
    </Field>
</Normalize>
<Format>
    <!-- Correlatable properties, must populate at least one -->
    <Field output="ActivityID" format="{0}" input="temp_jobname" />
    <Field conditional="snmp_agent" output="DeviceIPV4Address" format="{0}"
    input="snmp_agent" />
    <!-- verify that at least one property is set -->
```

```
    <Field conditional="DeviceIPV4Address" output="temp_atleastoneset"
    format="{0}" input="DeviceIPV4Address" />
    <Field conditional="!temp_atleastoneset" output="Error" format="At least
    one correlatable property is not set" input="" />
    <!-- Non-Correlatable properties  -->
    <Field output="RuntimeName" format="{0}.{1}.{2}"
    input="temp_applname,temp_applgen,temp_jobname" />
    <Field output="RuntimeDiscriminator" format="{0}" input="temp_applgen" />
    <Field output="DefinitionName" format="{0}.{1}"
    input="temp_applname,temp_jobname" />
    <Field output="ActivityTypes" format="Job" input="" />
    <Field conditional="activityState" output="ActivityState" format="{0}"
    input="activityState" />
    <Field conditional="temp_state" output="StateDescription" format="{0}
    present status is {1}" input="temp_jobname,temp_state" />
</Format>
</EventClass>
<!--  ======Event Class======   -->
<EventClass name="Alert">
<Parse>
    <Field input="snmp_varbindvals"
    output="temp_nodetype,temp_nodename,temp_domain,temp_applname,temp_applge
    n,temp_jobname,temp_jobequal,temp_state,temp_status"
    pattern="^(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?),(.*?)$" />
</Parse>
<Normalize>
    <Field input="temp_state" type="map" output="severity">
    <mapentry mapin="[Uu][Nn][Kk][Nn][Oo][Ww][Nn]" mapout="Unknown" />
    <mapentry mapin="[Cc][Oo][Mm][Pp][Ll][Ee][Tt][Ee]" mapout="Normal" />
    <mapentry mapin="[Mm][Oo][Nn][Ii][Tt][Oo][Rr]" mapout="Normal" />
    <mapentry mapin="[Ee][Xx][Ee][Cc]" mapout="Normal" />
    <!--  Informational   -->
    <mapentry mapin="[Ff][Aa][Ii][Ll][Ee][Dd]" mapout="Critical" />
    <mapentry mapin="[Pp][Rr][Ee][Mm][Aa][Tt][Uu][Rr][Ee] [Ee][Nn][Dd]"
    mapout="Critical" />
    <mapentry mapin="[Ii][Nn][Aa][Cc][Tt][Ii][Vv][Ee]" mapout="Minor" />
    <mapentry mapin="[Oo][Vv][Ee][Rr][Dd][Uu][Ee]" mapout="Major" />
    <mapentry mapin="[Ss][Uu][Bb][Ee][Rr][Rr][Oo][Rr]" mapout="Major" />
    <mapentry mapin="[Aa][Gg][Ee][Nn][Tt] [Dd][Oo][Ww][Nn]" mapout="Critical"
    />
    <!--  Fatal   -->
    <mapentry mapin="[Rr][Ee][Aa][Dd][Yy]" mapout="Normal" />
    <mapentry mapin="[Aa][Bb][Aa][Nn][Dd][Oo][Nn]
    [Ss][Uu][Bb][Mm][Ii][Ss][Ss][Ii][Oo][Nn]" mapout="Unknown" />
    </Field>
</Normalize>
```

```
    <Format>
        <!-- Correlatable properties, must populate at least one  -->
        <!--  Non-Correlatable properties  -->
        <Field output="MdrElementID" format="alert-{0}:{1}:{2}"
        input="snmp_agent,temp_applname,temp_jobname" />
        <Field output="OccurrenceTimestamp" format="{0}" input="{xsdateTime(now)}"
        />
        <Field output="ReportTimestamp" format="{0}" input="{xsdateTime(now)}" />
        <Field output="AlertType" format="{0}" input="Risk-Fault" />
        <Field conditional="severity" output="Severity" format="{0}"
        input="severity" />
        <Field output="AlertedMdrProduct" format="CA:00036" input="" />
        <Field output="AlertedMdrProdInstance" format="{0}"
        input="{fqdn(snmp_agent)}" />
        <!--  Assign instance name   -->
        <Field conditional="snmp_agent" output="Section1" format="{0}"
        input="snmp_agent" />
        <Field conditional="!snmp_agent" output="Flag" format="false" input="" />
        <Field conditional="temp_applname" output="Section2" format="{0}"
        input="temp_applname" />
        <Field conditional="!temp_applname" output="Flag" format="false" input=""
        />
        <Field conditional="temp_jobname" output="Section3" format="{0}"
        input="temp_jobname" />
        <Field conditional="!temp_jobname" output="Flag" format="false" input="" />
        <Field conditional="Flag" output="AlertedMdrElementID" format="" input=""
        />
        <Field conditional="!Flag" output="AlertedMdrElementID"
        format="{0}:{1}:{2}" input="Section1,Section2,Section3" />
        <Field output="Summary" format="{0}" input="temp_status" />
        <Field conditional="temp_state" output="Message" format="{0} alert on {1}
        scheduled on host {2}" input="temp_state,temp_jobname,snmp_agent" />
        <Field output="MetricName" format="{0}" input="Job Status" />
        <Field output="MetricType" format="{0}" input="Unknown" />
        <Field output="MetricUnitDefinition" format="{0}" input="Number" />
        <Field output="MetricDataType" format="{0}" input="String" />
    </Format>
    </EventClass>
    <!--  ======Event Class======   -->
    <EventClass name="USM-Entity" />
    </EventClass>
</Catalog>
```

Appendix A: Connector Identification Numbers

The USM schema assigns an ID value to each connector defined by enumerations of the MdrProduct property. These five-digit values appear as a part of the connector (or CA SOI plugin) name in the CA SOI interfaces. The following table shows the MdrProduct values for all available connectors and plugins:

Note: To find values for connectors not listed on this table, see the USM schema documentation.

For more information about plugins, see CA SOI Plugins.

MdrProduct	Connector/CA SOI Plugin
CA:00001	CA Application Performance Management
CA:00002	CA eHealth Performance Manager
CA:00003	CA NSM DSM
CA:00004	CA SOI
CA:00005	CA Spectrum Infrastructure Manager
CA:00006	Automation Manager Platform (includes CA Spectrum Automation Manager, CA Virtual Assurance, and CA Virtual Automation)
CA:00007	CA Access Control
CA:00013	CA SiteMinder
CA:00015	CA Workload Automation AE
CA:00018	CA Clarity PPM
CA:00019	CA Service Catalog
CA:00020	CA Service Desk Manager/CA CMDB
CA:00022	CA IT Client Manager
CA:00030	CA Catalyst
CA:00031	Microsoft SCOM
CA:00033	CA Application Configuration Manager
CA:00034	CA NetQoS Performance Center

MdrProduct	Connector/CA SOI Plugin
CA:00035	CA SYSVIEW Performance Manager
CA:00036	Generic SNMP traps
CA:00037	Microsoft Windows Event Log
CA:00038	HP Business Availability Center
CA:00039	Text log files
CA:00040	CA OPS-MVS Event Management and Automation
CA:00041	Web services events
CA:00042	IBM Tivoli Netcool
CA:00043	IBM Tivoli Enterprise Console
CA:00044	IBM Service Request Manager
CA:00045	BMC Atrium
CA:00046	BMC Remedy
CA:00047	CA SOI Modeler
CA:00048	CA Insight Database Performance Manager
CA:00049	CA NSM WorldView
CA:00051	BMC Remedy and BMC Atrium connector
CA:00052	CA NSM Event Management
CA:00056	Service Discovery plugin
CA: 00062	CA SOI Domain connector plugin
CA:09993	Mid-tier connector plugin
CA:09995	Manual CA Catalyst Persistence Store updates
CA:09996	CA SOI web services
CA:09997	Universal Connector Plugin
CA:09998	Sample connector
CA:09999	DB Test Application

The MdrProduct values from CA:00035 to CA:00041 and CA:00052 define sources sent from the Event connector.

The CA SOI MdrProduct values define CA SOI updates as follows:

CA:00004

Defines CIs published by CA SOI into the Persistence Store, through migration or otherwise.

CA:00047

Defines CIs manually created from the Service Modeler.

CA:09995

Defines a manual update of the Persistence Store.

CA:09996

Defines CIs created using CA SOI web services through the USM Web View.

Appendix B: USM Specifics

This section provides details about how to access the USM schema documentation and the parts included in the USM schema.

This section contains the following topics:

How to Access the USM Schema Documentation

The USM schema documentation is available on the CA SOI bookshelf https://support.ca.com/cadocs/0/CA%20Service%20Operations%20Insight%20r3%202-ENU/Bookshelf_Files/HTML/USM%20Schema/USM-Schema-out.html.

USM Parts

The Unified Service Model (USM) USM Schema files and documentation link in the Document Library of the CA USM-Catalyst Global User Community page provides information about the USM parts.

The USM includes the following parts:

- XSD files
- XML instance files
- WSDL files
- Other USM files

XSD Files

The following are XSD files in the USM schema:

- usm-core-200907
- usm-extensions-201001
- usm-metadata-200907
- usm-metrics-200907

Schema contents are defined by the usm-core-200907.xsd file. This file contains the type declarations for all root elements in USM.

The usm-extensions-2001001.xsd file contains properties that you can add to specific USM types, but are not expected to be included in most instances of these types. For example, the USM BinaryRelationship type is instantiated to associate instances and the type's semantic property defines the specific meanings to ascribe to the association. In some cases, this type defines an ordering of the relationships and you use the usm-exts:Order property. This type is inserted into the BinaryRelationship instance at the location of the xs:any declaration.

Information about the new property and meta-data that describes the type extended are included in the usm-extensions XSD.

Based on this example, the following is the definition for the Order property:

```
<xs:element name="Order" type="usm-meta:NonNegativeInteger">
<xs:annotation>
  <xs:appinfo>
    <usm-meta:NewElementFor>usm-core:BinaryRelationship
       </usm-meta:NewElementFor>
    </xs:appinfo>
    <xs:documentation>An integer value indicating the ordering of members in a
relationship. This element extends all semantics of BinaryRelationship.
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

The usm-metadata-200907.xsd file specifies the semantics and formats of USM meta-information. You define meta-data using the following techniques:

- As a global attribute, such as isReserved that indicates an element is set by the infrastructure
- As type data, such as MultiValuedString that indicates a string property holds a list of data values; the meta-data defines the ordering of the values in the list and their delimiters
- As root elements for separate XML instance files such as the usm-correl-query.xml file

The xs:appinfo element of the type definition holds meta-data that is defined as type data. An example is the usm-exts:Order element as indicated in the example.

The usm-metrics-200907.xsd file specifies the format of metrics that you can collect from domain managers.

XML Instance Files

The following XML instance files are in the USM schema:

- usm-query
- usm-openenums
- usm-infradefaults

The usm-query file defines meta-data specifying type elements that you can use in a query. For performance queries, the CA Catalyst infrastructure assumes you can query some elements, but not all (some properties cannot be queried to select particular instances returned). Elements you can query are specified as QueriableElements in the usm-correl-query file.

The usm-openenums.xml file defines the values for enumerations that are expected to evolve over time and are hierarchical. These types of enumerations are difficult to define as opposed to static enumerations, which are not expected to change. Static enumerations are defined within an XSD, using the xs:restriction of a string and the xs:enumeration facet. However, taking this approach means that any change to the enumeration is a revision of the schema. Schema revisions that update one enumeration value are not desirable.

Using xs:enumeration works well when defining closed sets of values such as the days of the week. It does not work well when defining value lists for country, vendor, or operating system names, which change over time, sometimes daily.

Static (closed) enumerations are an exception. Most enumerations are open (evolving). Enumerations can constantly grow, be subject to interpretation or perspective, or can be hierarchical. Another example of an open enumeration is a US region value list that specifies a country by quadrants, versus a different customer's perspective that defines smaller groupings of selected states. USM can define enumerations in an XML instance file where a property's value can be checked against the instances in the usm-openenums file. An error or warning (depending on the needs of the application) can report when a new value is encountered and alert needs to spell-check, correct the value, or add to the enumeration meta-data.

The following illustration shows an example of an open enumeration from the usm-openenums.xml file for the AlertTypeEnum:

```
<usm-meta:EnumDetails enumName="AlertTypeEnum">
  <usm-meta:Documentation>A categorization of Alerts along the lines of risk, quality, compliance
  and cost.</usm-meta:Documentation>
  <usm-meta:HierarchicalValue value="Risk">
    <usm-meta:Description>Indicates that the Alert deals with errors, exceptions, changes in state, etc.
    such as out of memory conditions or application exceptions.</usm-meta:Description>
    <usm-meta:HierarchicalValue value="Risk-Capacity"/>
    <usm-meta:HierarchicalValue value="Risk-Change"/>
    <usm-meta:HierarchicalValue value="Risk-Fault"/>
    <usm-meta:HierarchicalValue value="Risk-Security"/>
    <usm-meta:HierarchicalValue value="Risk-Performance"/>
    <usm-meta:HierarchicalValue value="Risk-RootCause"/>
  </usm-meta:HierarchicalValue>
  <usm-meta:HierarchicalValue value="Quality">
    <usm-meta:Description>Indicates that the Alert deals with perceived quality by the user/consumer.
    An example of this type of Alert is an SLA violation based on time to respond to a query.
    </usm-meta:Description>
  </usm-meta:HierarchicalValue>
  <usm-meta:HierarchicalValue value="Compliance"/>
  <usm-meta:HierarchicalValue value="Cost">
    <usm-meta:Description>Indicates that the Alert deals with cost - either increased or decreased.
    An example of this type of Alert is a necessary switch to a different cloud provider that results in
    a change in cost.</usm-meta:Description>
  </usm-meta:HierarchicalValue>
</usm-meta:EnumDetails>
```

The CA Catalyst infrastructure and CA Technologies products require specific data. The USM schema is not directly tagged with this data as it is stored in a separate XML instance file called usm-infraadefaults.xml (infrastructure defaults).

WSDL Files

The usm-metrics-200907.wsdl file specifies the operations that can be invoked against USM instances.

The following are valid operations:

GetAvailableMetrics

Returns a list of metrics for a specified instance. The list is an aggregation of the metric data from all products that have published an instance that was correlated.

GetMetricsValues

Returns the values of one or more requested metrics for a specified instance. The data returned can be live data or historical (time-stamped).

As metrics are complex, USM includes basic metric operations.

Other USM Files

The following information is available from the CA Technologies USM-Catalyst Global User Community page:

USM Metrics Dictionary

A list of the key metrics and performance indicators available from the CA Technologies products for many USM types. This dictionary is an Excel spreadsheet.

USM Schema Documentation

An HTML document defining the USM XML Schema constructs.

USM Schema Overview

A Word document providing an overview of the USM schema. The document also includes information about the structure, goals, and components of the schema and how it was designed.

USM Areas of Work

A presentation on the current areas of work for future USM extensions.

USM Schema Files

A compressed file including the schema XSD, XML, and WSDL files.

Advanced USM Features (Additional Information)

This section provides more information about the advanced USM features:

- Examples of supported custom operations (see page 236)
- Examples of supported metrics (see page 237)
- Details about the Incident Profile (see page 238)

Examples of Supported Custom Operations

This section includes two examples that list the supported custom operations:

- Example 1 lists the custom operations (see page 56) that the BMC Atrium/Remedy connector supports
- Example 2 lists the custom operations that the IBM Tivoli Service Request Manager connector supports

Example 1: Custom Operations Supported by the BMC Atrium/Remedy Connector

The BMC Atrium/Remedy connector supports the following custom operations as part of the Incident Profile, which contains a common definition for incident management for use across incident management products:

- CreateIncident
- UpdateIncident
- GetComments
- GetAttachments
- GetIncidents

Note: For more information about the BMC Atrium/Remedy connector and how these custom operations work, see the *BMC Atrium/Remedy Connector Guide*.

Example 2: Custom Operations Supported by the IBM TSRM Connector

The IBM Tivoli Service Request Manager connector supports the following custom operations as part of the Incident Profile, which contains a common definition for incident management for use across incident management products:

- CreateIncident
- UpdateIncident
- GetComments
- GetIncidents

Note: For more information about the IBM Tivoli Service Request Manager connector and how these custom operations work, see the *IBM Tivoli Service Request Manager Connector Guide*.

Examples of Supported Metrics

This section includes two examples that list the supported metrics:

- Example 1 lists the metrics (see page 56) that the BMC Atrium/Remedy connector supports
- Example 2 lists the metrics that the IBM Tivoli Service Request Manager connector supports

Example 1: Metrics Derived by the BMC Atrium/Remedy Connector

The BMC Atrium/Remedy connector derives the following metrics from imported data:

- AvgNumberOfIncidentsPerUser
- AvgResolutionTimeForPriorityXxx IncidentsInMinutes
- AvgResponseTimeOfPriorityXxx IncidentsInMinutes
- PercentMajorIncidents
- NumberOfIncidentsCreated
- UnassignedIncidents

Note: For more information about the BMC Atrium/Remedy connector, see the *BMC Atrium/Remedy Connector Guide*.

Example 2: Metrics Derived by the IBM TSRM Connector

The IBM Tivoli Service Request Manager connector derives the following metrics from imported data:

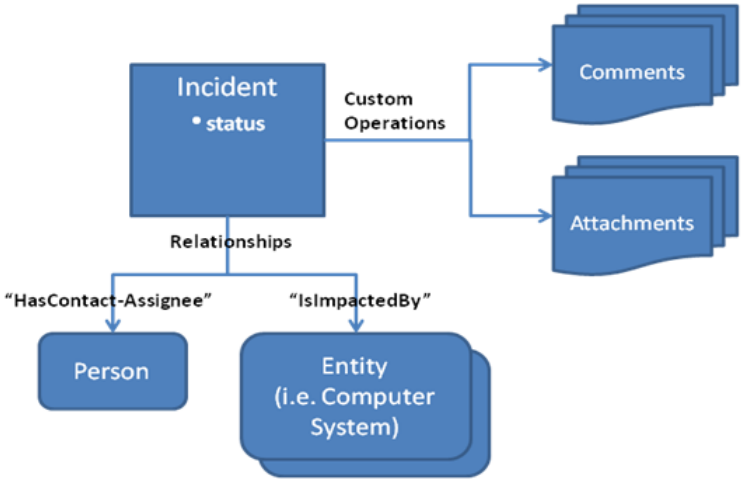
- AvgNumberOfIncidentsPerUser
- PercentMajorIncidents
- NumberOfIncidentsCreated
- UnassignedIncidents
- AvgDailyIncidentsPerAgent

Note: For more information about the IBM Tivoli Service Request Manager connector, see the *IBM Tivoli Service Request Manager Connector Guide*.

Incident Profile

The Incident Profile (see page 56) contains a common definition for incident management that you can apply to connectors, CA Technologies and third-party incident management products. A CA Catalyst connector that complies with the Incident Profile supports the CI types, custom operations, query filters, and metrics defined in the profile.

The following illustration shows the Object Model for incidents:



Incident objects are accessible using the query custom operation, with optional filters. Incident comments and attachments are accessible through custom operations; the related persons and impacted entities are exposed with relationships. Metrics about incidents are accessible through the USM metrics operations.

A CA Catalyst connector indicates support for the Incident Profile by returning the Incident Profile namespace value using the ConnectorDescriptor.getManagementCats() interface.

USM Types

A connector that is compliant with the Incident Profile supports the USM Incident, Person, and any impacted entity types (for example, ComputerSystem, Router, and so on).

The connector enumerates its supported CI types using the ConnectorDescriptor.getManagedEntityTypes() interface.

Supported Custom Operations

The Incident Profile supports the following custom operations:

- CreateIncident
- UpdateIncident
- GetComments
- GetAttachments
- GetIncidents

Supported Metrics

The Incident Profile supports the following metrics:

Note: For more information about the USM metrics dictionary, see the USM documentation.

- AvgResolutionTimeForPriorityXxx IncidentsInMinutes
- PercentPriXxxIncidentsResolved OnTime
- AvgResponseTimeOfPriorityXxx IncidentsInMinutes
- PercentPriorityXxxIncidents RespondedOnTime
- AvgNumberOfIncidentsPerUser
- PercentMajorIncidents
- NumberOfInstallIncidents
- NumberOfLearnabilityIncidents

- NumberOfOperationIncidents
- NumberOfUnderstandability Incidents
- PercentIncidentsResolvedInTarget
- PercentXxxIncidentsResolvedInTarget
- AvgIncidentCreateToResolveInMinutes
- AvgXxxIncidentCreateToResolveInMinutes
- PercentIncidentsAutoGenerated
- PercentIncidentsDueToChanges
- PercentIncidentsDueToDataIntegrity
- PercentIncidentsDueToVirus
- PercentIncidentsDispatched
- PercentIncidentsDueToLackOfTraining
- AvgDailyIncidentsPerAgent
- NumberOfIncidentsCreated
- UnassignedIncidents
- IncidentSLACount

Glossary

alert

An *alert* is a message on the Operations Console that reports a fault condition that is associated with a resource or service.

Apache ActiveMQ

Apache ActiveMQ is an open source (Apache 2.0 licensed) message broker that fully implements the Java Message Service 1.1.

CA Catalyst

CA Catalyst is a platform for federating, correlating, reconciling, and storing high-level, business-relevant data from a wide variety of management products.

configuration item (CI)

A *configuration item* (*CI*) is a managed resource such as a printer, software application, or database. Configuration items support services. A synonym for a configuration item is a *resource.*

Connector

A *connector* is software that provides the interface for the data exchange between the CA Catalyst infrastructure and a domain manager.

consuming product

A *consuming product* leverages the infrastructure by consuming information that connectors retrieve from integrated products.

correlation

Correlation is the act of comparing CIs to determine equivalencies—whether the CIs represent the same underlying entity.

enrichment

An *enrichment* adds information to an event or alert from an outside source. An outside source can include a database, method, or script. You can add an enrichment to event policies to enrich matching events with important information.

event

An *event* is a message that indicates an occurrence or change in your enterprise. Events can indicate a negative occurrence or object state change. CA SOI Event Management lets you view and manage events that are received from all connectors.

Event Management

Event Management provides a processing layer between raw connector USM alert data and alert management. You can influence the data that makes it into the Operations Console as alerts and how those alerts appear. Available Event Management actions include event correlation, filtering, creation, and enrichment.

event policy

An *event policy* is a combination of event search patterns and an action to perform when the patterns match. You can deploy event policies on all or specific sources, and available actions include filtering, event creation, and enrichment.

eXtensible Markup Language (XML)

eXtensible Markup Language (XML) is a syntax for creating structured data files for electronic transmission and consumption. XML files typically have the extension .xml.

federation

Federation is the act of joining data from various sources.

FIPS 140-2 (Federal Information Processing Standard)

FIPS 140-2 (Federal Information Processing Standard) describes US Federal government requirements that IT products should meet for sensitive but unclassified use.

high availability

High availability helps maintain a business continuity during an IT resource interruption. Also known as fault tolerance or failover.

integration framework (IFW)

The *integration framework (IFW)* is the mechanism that CA SOI uses to connect to domain managers and gather CI, service, topology, and state information.

metadata

metadata is information about data, specifically information further explaining the meaning, structure, value set, and other items of the data elements of the USM schema.

Mid-tier connector

The *Mid-tier connector* is an intermediate processing layer through which you can deploy cross-domain event policy for automated processing on events from all connectors. All events flow through the Mid-tier connector before reaching the Operations Console as alerts.

reconciliation

*Reconciliation c*reates a unified set of properties and values from instances of a single entity that multiple connectors retrieve.

resource

A *resource* is a managed resource such as a printer, software application, or database. Resources support services. A synonym for a resource is a configuration item (CI).

SA Manager

The *SA Manager* (also called SAM Application Server) is the primary management component in CA SOI. The SA Manager monitors the health and availability of managed resources and services. The SA Manager also processes events from connected applications and performs service impact and risk analysis.

service

A *service* typically consists of several CIs, which are grouped to represent entities like web server farms or clusters. Services can also contain *subservices*, which are subordinate service models. Service models typically represent high-level abstract entities like a web-based retail transaction service, an application server service, or a source control service. You can define any service type with CA SOI as long as one of the integrated domain managers monitors the service components.

service model

A *service model* is a definition of a service or other entity in your enterprise. It is a logical grouping of resources, associations, dependencies, and policies.

Unified Service Model (USM)

The *Unified Service Model (USM)* is the semantic schema that is used as the CA Catalyst and CA SOI infrastructure.

USM schema

The *USM schem*a defines the internal format for all CA Catalyst data, which is transmitted for display to CA SOI.

Web Service Definition Language (WSDL)

Web Service Definition Language (WSDL) is an XML-based language to describe operations such as their inputs, outputs, faults, and protocol bindings, which are addressed to specific endpoints. Files created using WSDL typically have the extension .wsdl.

XSD

XSD is the XML schema, which is an XML-based language that defines the contents and constraints on data in an XML document. XSD files typically have the extension .xsd.