Red Hat JBoss Fuse Service Works 6.2.1

Governance Development Guide

A guide to implementing Governance for JBoss Fuse.

Last Updated: 2017-11-12

Red Hat JBoss Fuse Service Works 6.2.1 Governance Development Guide

A guide to implementing Governance for JBoss Fuse.

JBoss A-MQ Docs Team
Content Services
fuse-docs-support@redhat.com

Legal Notice

Abstract

Use this guide to help implement Governance on Red Hat JBoss Fuse

Table of Contents

CHAPTER 1. GOVERNANCE

1.1. WHAT IS SOA GOVERNANCE?

A distributed system involves many components that must collaborate well in order to deliver high performance. Monitoring and managing a large distributed system is a critical and complex activity. It becomes more challenging with Service Oriented Architecture (SOA), as there is no control over infrastructure and there may be indeterminate delays and failures.

In order to achieve optimum quality of service, predictability, consistency, performance, and control over its resources, SwitchYard makes use of SOA Governance. SOA Governance is nothing but exercising control over services in a Service Oriented Architecture. It helps with the adoption, implementation, and sustainability of SOA. SOA Governance covers people, processes, and technologies for the entire SOA lifecycle by exercising defined policies, access control, and service monitoring. It keeps track of what services are running, what are their contracts and SLAs, and whether they are being violated. SOA Governance brings these policies and access control into the way in which services are used within a business process.

SwitchYard provides two types of service governance:

- Design Time Governance: Design Time Governance provides support for design, creation, and implementation of services. In addition to that, it also supports service testing, deployment, and versioning. It uses S-RAMP as the central repository for services including policies and documentation.

- Runtime Governance: Runtime Governance provides monitoring, control, auditing, and policy enforcement capabilities (both business and service level) to the services at runtime.

1.2. FEATURES OF SOA GOVERNANCE

SOA Governance in SwitchYard provides the following features:

- Controls all the phases of a service lifecycle, not just the run-time management.

- Is extensible and caters to change requirements, as what needs to be monitored may change with time.

- Leverages existing monitoring capabilities, so you need not employ a separate approach for fault tolerance and reliability.

- Is SOA standard complaint and therefore enables you to replace components with other compatible implementations.

1.3. SOA GOVERNANCE ACTIVITIES

- Lifecycle Management, including planning, creation, and deployment of services

- Change Management of services

- Policy use to restrict service behavior to enforce conformance and compliance

- Monitoring services for uptime, performance, reliability, and security

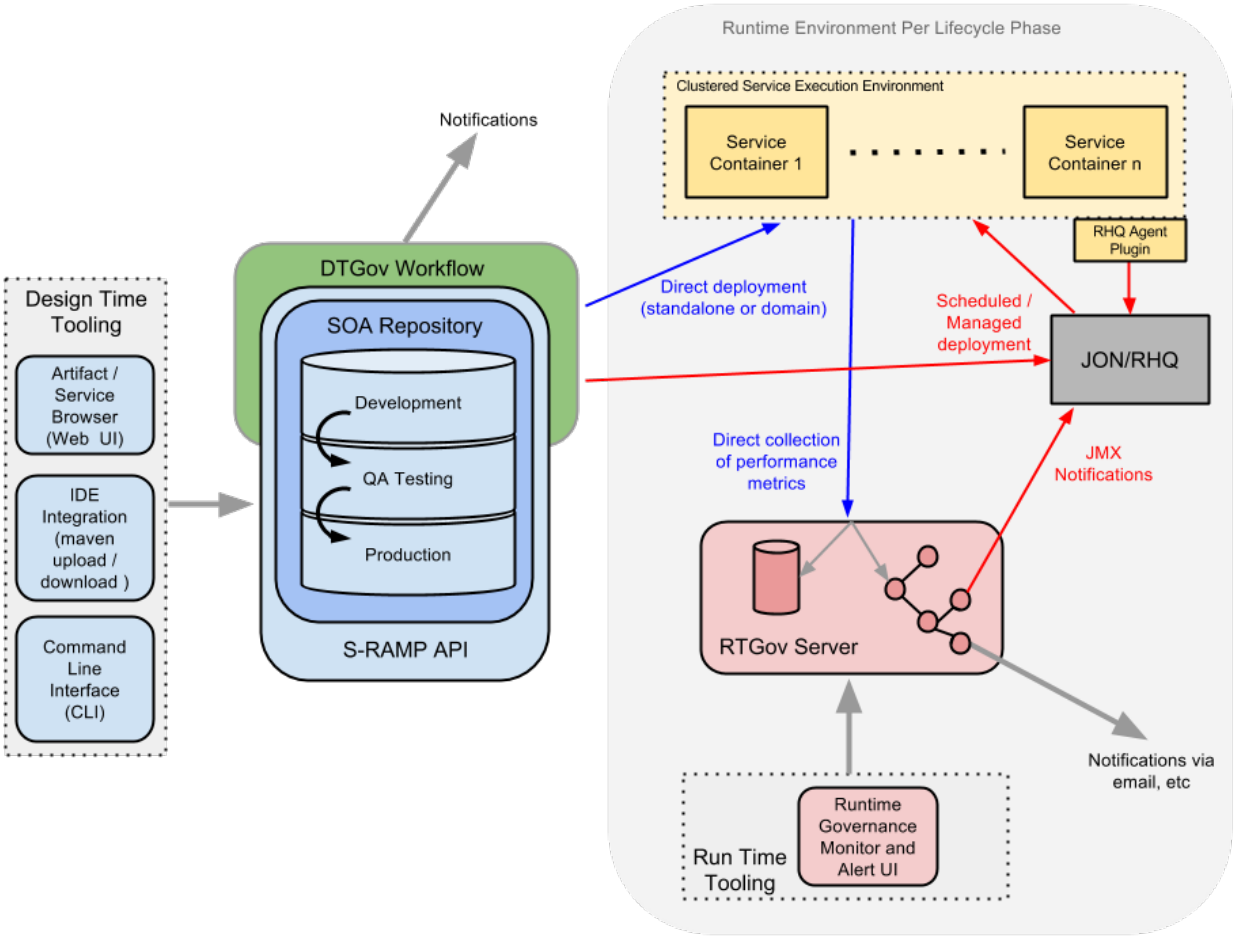
- Impact analysis, service composition for complex scenarios

- Access Management to specify who can use a service for how many times
- Documentation, for both business and technical owners

1.4. SOA GOVERNANCE ARCHITECTURE

The following architectural diagram illustrates how the Design Time Governance, Runtime Governance, and the S-RAMP repository work together to provide an integrated SOA solution:

Figure 1.1. SOA Governance Architecture



The web-based Design Time Tooling provides support for creating service artifacts in an Integrated Development Environment (IDE). You can access SOA repository (S-RAMP) and reuse existing resources to further build their services and test them. The runtime Environment enables running multiple services as a clustered service execution for both direct and scheduled deployments. The Business Activity Monitor that has access to the latest policies monitors the executing services, creates performance metrics and further sends notifications to the user.

1.5. SOA GOVERNANCE USE CASES

SOA Governance provides visibility and control in every stage of the SOA lifecycle for organizations with a wide range of artifacts. This section lists some of the use-cases of SOA Governance implementation in an insurance company:

Runtime Governance

An insurance company implementing Runtime Governance can monitor service response times and based on that, make business decisions in real-time. For example, based on the service response

time, resources can be removed or added to the data center. Runtime Governance enables them to easily customize the analysis and presentation or real-time information. They can monitor load on the infrastructure and conclude on how much load can they handle. They can monitor service fulfillment times and find out how long does it take to process a new insurance policy application. Using policy enforcement, an insurance company can block a customer from placing any more orders until they have reduced their debt.

Also, Runtime Governance may be applied in an insurance scenario to ensure regulations are being followed. For example, if a customer cancels a policy within 14 days, then it can ensure that the policy is actually canceled, as otherwise regulatory penalties may apply.

S-RAMP

The S-RAMP repository can be used by an insurance company implementing SOA Governance to store, extract, and derive content (artifacts) and Metadata used by the company. The S-RAMP repository also classifies the artifacts and provides rich query support.

1.6. RED HAT JBOSS GOVERNANCE PACKAGE

This section lists the two major components within Red Hat JBoss Governance:

S-RAMP

SOA Repository Artifact Model and Protocol (based on the OASIS standard)

- Artifact repository
  - Built on top of ModeShape
  - Stores artifact content and meta-data including:
    - Core properties (name, description, version, etc)
    - Custom ad-hoc properties (name-value)
    - Relationships between artifacts
    - Hierarchical tagging system called Classifiers
  - Rich XPath-like query language to search the repository
  - Atom based REST API
  - Automatic decomposition of artifacts based on type
    - Built in support for common S-RAMP types (XSD, WSDL, etc)
    - Extensible to support additional types (Teiid VDB, switchyard.xml, etc)
- Command Line Interface
  - Supports interactive and batch modes
  - Built-in commands to manage and search for artifacts in the repository
  - Extensible to allow contributing custom commands

- Maven integration
  - Can "mvn deploy" a deployment artifact directly into S-RAMP
  - Able to use S-RAMP artifacts as dependencies in maven projects
- Web UI
  - Based on Errai 2.4
  - Allows granular management of content in the repository
  - Can be used to manipulate all artifact meta-data

RTGov

Runtime Governance

- Activity Collector
  - Activity events can be reported directly to the Activity Server via REST service, OR
  - Embedded collector to intercept activities, pre-process the information to extract correlation details and business relevant properties, before batching events for distribution to the server.
    - Events can be 'validated' using pluggable event processors, which can be used to block business transactions
- Activity Server
  - Provides a REST interface to enable events to be reported
  - Persists activity information
  - Distributes reported events to subscribed components
- Event Processor Network
  - Subscribes to the Activity Server, to receive reported activity events
  - Distributes activity events through a series of registered networks, that consist of event processors to validate, transform and retransmit events. Processors can be specified using Java, Drools CEP rules and MVEL
- Maintaining Derived Results
  - Information derived by the Event Processor Network can be presented as a set of 'active' collections
  - Components can directly access these collections, or issue queries through a REST API
- Web UI
  - Runtime governance information is provided through a set of gadgets
    - Service Response Time Gadget - displays response time information per service and optionally operation

- Situations Gadget - table for displaying situations detected by the event processing (for example, SLA violation)
- Call Trace Gadget - provides a tree based visual display of the call stack associated with a business transaction
- Service Overview - graphical representation of the services and their dependencies, showing response time and invocation details

CHAPTER 2. DESIGN TIME GOVERNANCE

Design Time Governance has been deprecated in JBoss Fuse 6.2.1. It is not available for this version of JBoss Fuse.

CHAPTER 3. RUNTIME GOVERNANCE

In SwitchYard, the applications are exposed as shared business processes or services. In a shared environment, the most common challenges revolve around access control, access rights, security issues, and service authorizations. SwitchYard uses Runtime Governance to facilitate complete control over the shared services by following a policy based approach. SwitchYard classifies Runtime Governance into two aspects:

- Policy definition and enforcement
- Collection and exposure of runtime metrics for services and service references

To provide you with a deeper understanding of Runtime Governance and how it works, this chapter covers the following information:

- The Runtime Governance architecture, its components, and details on how these components integrate and process business information from the activity events.
- The activity events, and details on the types of activity events and their significance.
- The role of Activity Collector and Activity Server in collecting, validating, processing, and recording activity event information.
- The role of Event Processor Network and Event Processors in processing activity events.
- The Active Collection mechanism, its implementation, and details on how you can use Active Collection to store processed events and notify the end-users or applications.

For information on setting up the Runtime Governance server, see the Red Hat JBoss Fuse *Administration and Configuration Guide*.

For information on using the user interface and gadgets, see the Red Hat JBoss Fuse *User Guide*.

You can enable or disable Runtime Governance through the collectionEnabled property in the **`$JBOSS_HOME/standalone/configuration/overlord-rtgov.properties`** file. By default, the value of the collectionEnabled is set to false.

This property will determine whether activity information is collected when the server is initially started.

3.1. ARCHITECTURE OVERVIEW

The Runtime Governance architecture provides a modular and loose-coupled solution for processing business activity information in real-time. The architecture is classified into four distinct areas:

- Activity Collector: Used to collect activity events.
- Activity Server and Store: Used for central activity event storage and querying.
- Event Processor Network: Used for generic event analysis to process the activity events.
- Active Collections: Used for active information management to post-process and cache information for end-user applications.

You can configure these components to work together to build a Runtime Governance solution for real-time monitoring of business transactions. Out of these, the Activity Server is the only mandatory

component, as it provides the central hub for storing and querying activity events. This means that you can replace the way in which events are processed, or presented to end users and applications, with any other more appropriate technology for a particular target environment. You can use the Event Processor Network and Active Collection mechanisms to process and manage the presentation of any type of information.

Also, a collection of REST services is provided to support remote access to the information. You can use REST API to develop new custom code to access Runtime Governance information.

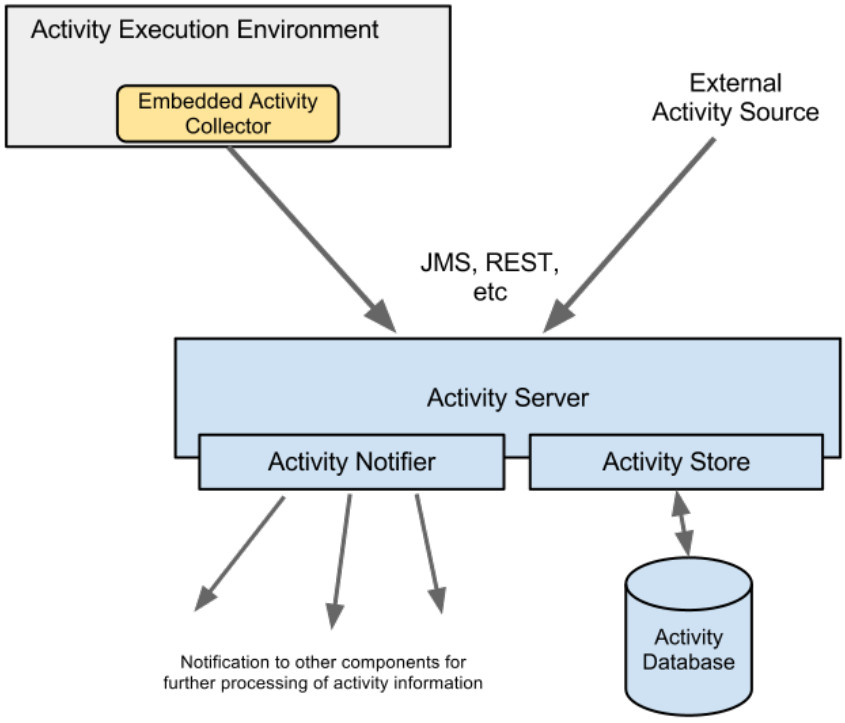
3.1.1. Activity Collection and Reporting

Runtime Governance supports the following public APIs:

- The Activity Server API: This is used by Runtime Governance itself to communicate with a remote Runtime Governance server.
- The Active Collection API: This is used to query active collections.

The Embedded Activity Collector collects events and sends them to the Activity Server in batches. Activity Server also receives events from external activity sources. The Activity Server saves these events in its data store and also notifies other components about the activity information. The diagram below illustrates the functions performed by the first stage of the Runtime Governance architecture:

Figure 3.1. Activity Collection, Reporting, and Storage



- ***`Collection`***: The Activity Collector component of the architecture is optional and is responsible for collecting information from the execution infrastructure as efficiently as possible. It collects the activity events associated with a particular thread as a group,

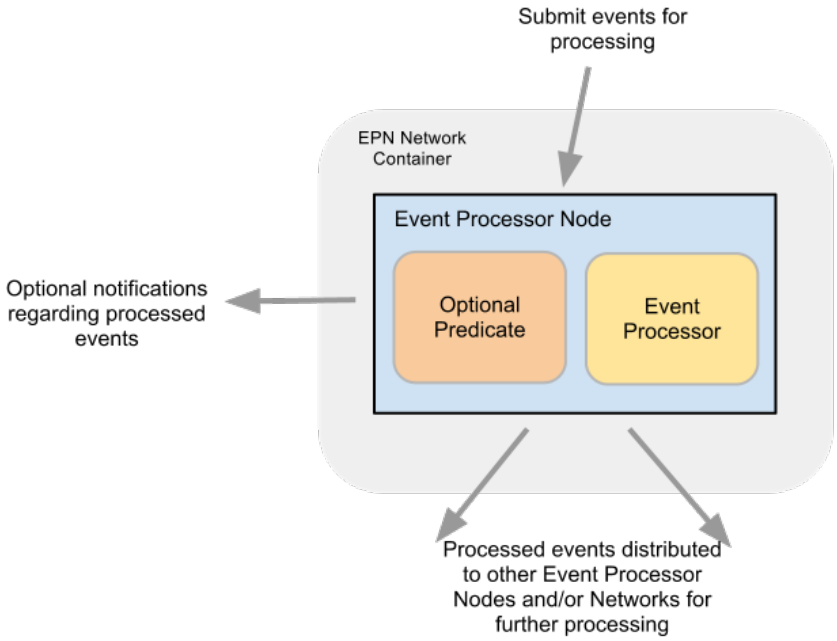
contained within an Activity Unit. This provides an implicit correlation of the activities that are associated with the same business transaction. You may also pre-process the activity events to extract relevant context and property information before it is reported to the Activity Server. Activity Units are then batched into further groups, and reported to the Activity Server at regular time intervals or when the batch gets too large. If the Activity Collector and Activity Server are co-located within the same execution environment, the Activity Units are reported directly. If the Activity Server is running remotely, then you can use suitable connectors to report the information via REST API.

- ***`Activity Server`***: The Activity Server provides a public API for reporting a list of Activity Units. Remote components can access this API either directly (for example, as a CDI component), or remotely via REST. The Activity Server has the following main responsibilities:
  - To ensure Ids are set and consistent.
  - To store the events in a repository.
  - To notify other interested modules.
- ***`Storage`***:The Storage component records the activity events in a persistent data store.
- ***`Notification`***: The Notification component is an API and keeps the other modules notified as and when the activity events are reported.

3.1.2. Event Processor Network

The Activity Server notifies the Event Processor Node about the activity events through the Event Processor Network. The Runtime Governance infrastructure notifies one or more Event Processor Networks (specifically Event Processor Networks that have a subscription for the subject ***`ActivityUnits`***) about activity events reported to the Activity Server. The following diagram illustrates how a node within an Event Processor Network functions to process the inbound event information.

Figure 3.2. Runtime Governance Event Processor Network and Nodes



The Event Processor Network is a graph based mechanism for processing a series of events. You can register one or more networks to receive the activity information as notifications from the Activity Server and process it. Processing involves filtering, transforming and analysing the activity events. Each network defines a graph of Event Processor Nodes connected by links that transfer the results from the source node to the target node. The graphs can subscribe to event subjects in order to identify the information they are interested in. The graphs then identifies and nominates the node(s) within the network that processes the information on that subject. The nodes can also publish their results to event subjects, for other networks to further process it. This provides a de-coupled way for networks to exchange information.

Each Event Processor Node defines the following:

- An optional predicate to determine whether the event is of interest, and
- An event processor to perform the actual task of event processing.

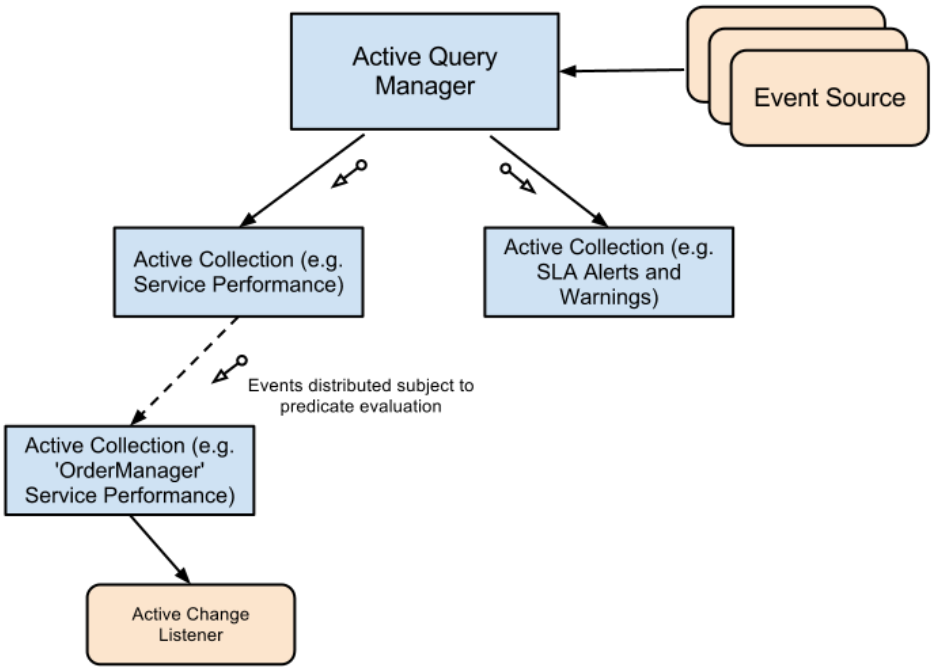
You can have multiple versions of the Event Processor Network. When you deploy a new version of a network, the events that are being processed by the old version continue to be processed using that network, while new events being dispatched to the network are handled by the newer version.

Additionally, you can configure the nodes to generate different levels of notification. other applications or components may use these notifications to check the information being processed through the network. These notifications are distributed to named notification subjects. This enables observing components to remain de-coupled from the details of which networks or nodes are producing results for that subject.

3.1.3. Active Collections

The Active Collection mechanism provides a capability for storing processed events and derived analysis information in a manner that the end users can access. It also actively notifies the end users or applications when any changes occur. The following diagram illustrates the Active Collection mechanism:

Figure 3.3. Active Collections



The Active Collection mechanism differs with the standard collection concept, where interested components or applications can register interest in changes that occur to the contents of a collection such as a list or a map. The Active Collection mechanism is used to maintain information that is presented to the users, for example via the Gadget Server.

An Active Collection Source manages the information within a particular Active Collection. The Active Collection Source acts as an adapter between the actual source of the information and the Active Collection. For example, an out of the box implementation of an Active Collection Source checks for different types of information produced by an Event Processor Network.

The generic Active Collection Source implementation includes:

- The ability to aggregate information which is then stored as a summary within the Active Collection.
- The ability to perform routine maintenance tasks.
- The ability to clear the collection entries based on configured criteria, such as maximum size of the collection, or maximum duration an item should exist in the collection.

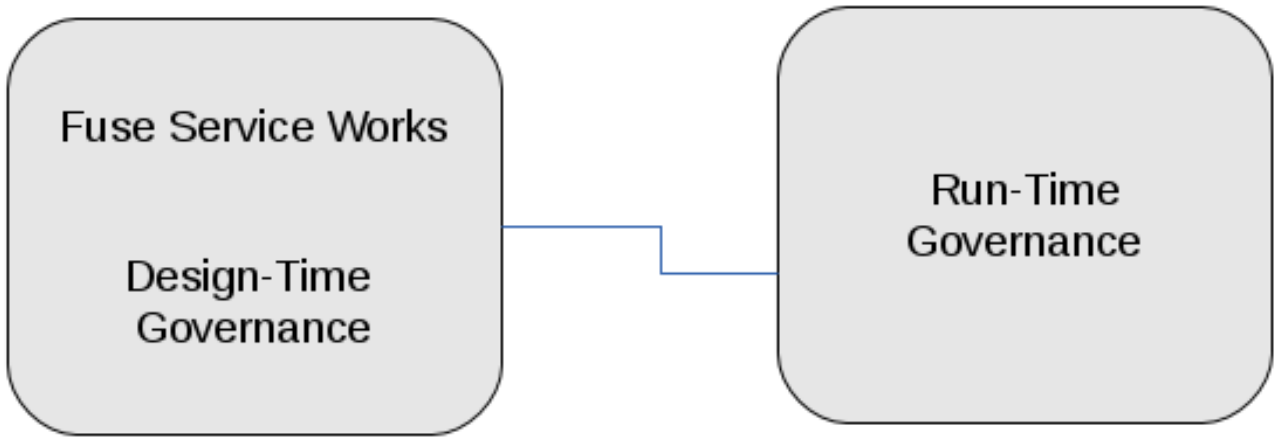
Additionally, you can also create derived (child) collections from these top-level collections. The derived collections have a predicate that determines whether an entry in the parent collection is relevant to the child collection. This can be used to manage specific sub-sets, and provide an active query mechanism, enabling interested clients to observe changes to the child collection.

3.1.4. Runtime Governance Topologies

You can install and run Runtime Governance either on the same application server as Design Time Governance, S-RAMP, and SwitchYard, or on a separate server.

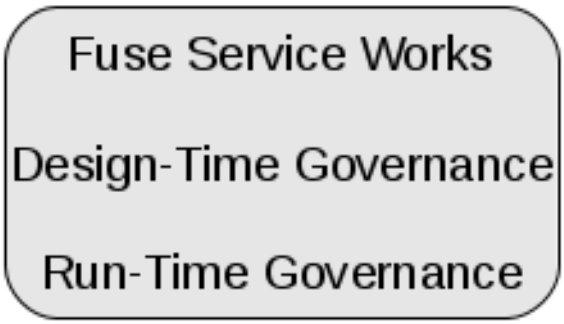
Using Runtime Governance client in the SwitchYard environment to talk to a remote Runtime Governance server, enables the Runtime Governance server to run in its own isolated cluster.

Figure 3.4. Runtime Governance on a Different Server



If co-located, then although you are saving on the REST communication and serialization, it means that all event processing will be done on the same servers as the business logic. And if the business application uses JMS, then the EPN event distribution also uses the same JMS infrastructure.

Figure 3.5. Runtime Governance on the Same Server as JBoss Fuse



If you are working on testing and development environments without high loads, it is possible to run one JVM with the server, Runtime Governance, and JMS enabled as the performance hit is not dramatic. However, if you need Runtime Governance in high performance environment, Red Hat recommends running Runtime Governance in a separate JVM. The JVM instance running Runtime Governance must have JMS enabled, whereas other application server instances can run without JMS.

The Runtime Governance server configuration is same in both situations. The only difference is that, in the Runtime Governance client configuration, there are configuration details in overlord-rtgov.properties file to provide the server URL for reporting activity events via a REST API. Also, you need to install the appropriate war file depending upon the Runtime Governance client and server topology you choose during installation.

3.1.4.1. Runtime Governance and S-RAMP in a Clustered Environment

When you start the server using the HA profile, runtime governance is automatically clustered based on the clustered Infinispan, HornetQ, and a shared database. S-RAMP only requires a shared database. There is no dependency between runtime governance and S-RAMP as they are installed and configured independently.

In a clustered environment, it is recommended to have a dedicated runtime governance server, separate from the execution servers. In this case, all servers connect to this server via the runtime governance client. Depending upon the load of the runtime governance server, it may also be necessary to set up a cluster of governance servers as well. If you have a cluster of runtime governance servers, then they only need to connect to the same database instance. However, it is possible to also configure each runtime governance server to have its own S-RAMP database. The JBoss Fuse installer configures all components to use the same database by default so that runtime governance and S-RAMP share the same database instance. However this is not mandatory.

3.2. RUNTIME GOVERNANCE MANAGED APPLICATIONS

3.2.1. Building a Runtime Governance Managed Application

When you deploy and use a SwitchYard application out-of-the-box, the activity information is reported and stored to the Runtime Governance Activity Server, provided activity collection is enabled. Additionally, the activity events are submitted to the out-of-the-box event processor networks (EPNs). The event processor networks then derive some default information, such as response time data and service dependency information. This information is available through the gadgets in the Runtime Governance user interface.

Procedure 3.1. To build a Runtime Governance managed application:

1. Deploy your SwitchYard application and enable Runtime Governance activity collection.
2. To record any additional context (correlation) or properties with the activity information, define an Information Processor with the configuration details required to extract the information.
3. To enforce policy decisions immediately (synchronously), define:
    a. An activity validator to contact the policy you wish to enforce.
    b. An interceptor in the SwitchYard application to trigger the validation.

NOTE

Activity Validators are only intended to validate the activities and raise exceptions where necessary to block the business transaction.

4. If policy decisions can be processed asynchronously, define an Event Processor Network which executes in the server.

NOTE

You can use the Event Processor Networks to derive further information (including Situations), enabling creation of multiple networks or nodes and building on previously processed information.

This section outlines a basic flow of building and customizing your Runtime Governance managed application. More details on event collection and processing are discussed later in this chapter.

3.2.2. Customizing Your Runtime Governance Managed Application

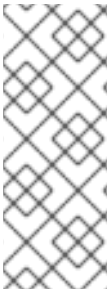
This section discusses how you can further enhance your Runtime Governance managed application. You can implement one or more of these basic flows, depending on the type of Runtime Governance you are interested in:

Create rules or policies based on general interaction information or response times

Based on the out-of-the-box configuration, you have an option to define your own Event Processor Networks (EPN). You can use your EPNs to observe the basic activity information or derived response times to perform required actions (such as deriving further information, and identifying situations to create an alert). For more information, refer the policy-async quickstart.

Create server side (asynchronous) policies based on business specific information

To define intelligent policies, it is necessary to extract information from the message content or headers. This requires the definition of an Information Processor to define how each message type should be processed to extract the relevant context and property information. For more information, refer the order management quickstart with an ip artifact defining the information processor for the order management application.



NOTE

Creating the ip and EPN modules associated with the SwitchYard application is useful as there may be different users developing the SwitchYard application and defining the governance aspects. Once the relevant information is extracted and associated with the activity events, the server side EPNs can access this information to perform more intelligent tasks, such as assess a customer's credit limit. For more information, refer the policy-async quickstarts.

Create client side (synchronous) activity validators

Activity validators are similar to EPNs, except that they operate in the SwitchYard environment, and can be used to block business transactions based on evaluation of a business policy. As the SwitchYard event collection is asynchronous, blocking a business transaction can only be achieved by performing the activity validation from within the SwitchYard application, with the use of an interceptor. For more information, refer the ordermgmt/app quickstart and the policy-sync quickstart, which defines the validator policy.

NOTE

Creation of an Information Processor configuration is optional. You can use it to extract additional relevant information that may be required by the activity validator policies.

3.3. ACTIVITY EVENTS COLLECTION

Before going in detail about how the Runtime Governance collects and processes the events, let us first understand what these activity events are. The Activity Model defines a set of events that are reported to Red Hat JBoss Fuse, in order to identify what is happening during the execution of a business transaction.

3.3.1. Activity Model

The Activity Model comprises the following components:

Activity Unit

The Activity Unit is the top-level and most important component of the Activity Model. The Activity Unit provides grouping capability. It aggregates a set of activities that relate to a particular transaction. The Activity Unit includes:

- Id: This uniquely identifies the activity unit for historical retrieval purposes.
- Origin: This information identifies the environment in which the activities were recorded.
- A set of contexts: This provides contextual information to help relate the activities with other activities recorded in other units.
- A group of activity types: This is the list of the actual activities or situations that occurred.

Origin

As mentioned above, Origin is a component of the Activity Unit. It represents information about the source of the activities associated with the Activity Unit. Origin provides the following information about an activity:

- Principal: This represents the user (if any) associated with the activities.
- Thread: This represents the execution path and is useful in diagnostic situations when used in conjunction with the host information.
- Host: This is the host name.
- Node: This is the name of the node, in the case when the server is part of a cluster.

Context

The Context items represent information used for correlating the activities within the unit against other Activity Units. It also provides identify information that may be useful when attempting to retrieve the unit. The Context contains the following information:

- type: This is the context type.
- value: This is the value of the context information.
- timeframe: This is an optional value used with a ***`Link`*** context type, to identify the time period in which the context is valid.

Here is the list of various context types you can define:

Table 3.1. Context Types

Type	Description
Context.Type.Conversation	The conversation id, which can be used to correlate activities across service boundaries. It is unique to a particular business transaction instance.

Type	Description
Context.Type.Endpoint	The endpoint id, which can be used to correlate activities within a service boundary such as a BPM process instance id. It is unique to a particular business transaction instance.
Context.Type.Message	The unique id for a message being sent or received. The message id may only be valid within the scope of an endpoint, as it's value may not be carried with the message contents to the recipient. A common usage is to correlate a response against the originating request within the same endpoint.
Context.Type.Link	This type represents a correlation between two activity events based on information that is unique and valid only for a limited time period.

Activity Type

All activity events are derived from an Activity Type superclass. This class has the following information:

- Activity unit id
- Activity unit index
- Timestamp
- Principal
- A set of contexts
- A set of properties

Out of these, the only piece of information that the reporting component needs provide is the Timestamp, and optionally some activity type specific contexts. The other information are initialized by the infrastructure before persisting the Activity Unit, in order to make the specific Activity Type instance locatable.

3.3.2. BPM and SOA Specific Activity Events

The BPM (Business Process Management) specific activity events are used to record the lifecycle and state transitions that occur when a business process (associated with a description language, such as BPMN2 or WS-BPEL) is executed within a runtime engine, in support of a business transaction. These business processes are long running, and they handle multiple requests and responses over a period of time, all being correlated to the same process instance. This means that the activities generated as a result of this execution must also correlate to:

- The specific XA transaction in which they are performed,
- The process instance that holds their state information in the BPM engine, and

- The conversation associated with the particular business transaction.

However, this does not mean that all Activity Units that contain activity information from the BPM engine need to have these three types of correlation information. For example, the initial Activity Unit for a business process instance may identify the first two and establish a unique process instance id. A subsequent Activity Unit may then define the same process id for the second one, as well as the third one, that can then be used to tie any Activity Unit related with the process instance id to that conversation. Therefore, all Activity Units with the same process instance id become directly or indirectly correlated to the conversation id that may only be declared in some of the Activity Units.

Here is a list of BPM and SOA specific activity types:

Table 3.2. BPM Activity Types

Activity Type	Description
ProcessStarted	This activity type is recorded when a process instance is initially started. Attributes include: process type, instance id and version.
ProcessCompleted	This activity type is recorded when a process instance completes. Attributes include: process type, instance id and status (either success or fail)
ProcessVariableSet	This activity type is recorded when a process variable value is set or modified. Attributes include: process type, instance id and variable name/type/value.

Table 3.3. SOA Activity Types

Activity Type	Description
RequestReceived and RequestSent	This activity type is recorded when a service invocation (request) is received or sent. Attributes include: message type, content and message id.
ResponseReceived and ResponseSent	This activity type is recorded when a service invocation returns. Attributes include: message type, content, message id and replyTo id (used to correlate the response to the original request)

3.3.3. Collecting Activity Events

The Runtime Governance Server collects activity information from events in the following two ways:

- Integrating an Activity Collector into the execution environment.
- Manually report the activity information to the Runtime Governance Server through a publicly available API such as REST service.

3.3.4. Integrating an Activity Collector

In this approach, the Runtime Governance Server integrates an Activity Collector into the execution environment. The Activity Collector intercepts activities and automatically reports them to the Runtime Governance Server. The Activity Collector makes use of Information Processors to enable information associated with the activity event to be pre-processed. The Activity Collector makes use of Activity Validators to install event processing capabilities. This section provides details on how to define and register Information Processors and Activity Validators.

3.3.4.1. Defining Information Processors

For integrating an Activity Collector inside the execution environment, you can use Information processors to extract relevant context and property values from activity events. Here is an example that illustrates the configuration of a single Information Processor:

```
[{
        "name":"OrderManagementIP",
        "version":"1",
        "typeProcessors":{
                "{urn:switchyard-quickstart-demo:orders:1.0}submitOrder":
{
                        "contexts":[{
                                "type":"Conversation",
                                "evaluator":{
                                        "type":"xpath",
                                        "expression":"order/orderId"
                                }
                        }],
                        "properties":[{
                                "name":"customer",
                                "evaluator":{
                                        "type":"xpath",
                                        "expression":"order/customer"
                                }
                        },{
                                "name":"item",
                                "evaluator":{
                                        "type":"xpath",
                                        "expression":"order/itemId"
                                }
                        }]
                },
                "{urn:switchyard-quickstart-
demo:orders:1.0}submitOrderResponse":{
                        "contexts":[{
                                "type":"Conversation",
                                "evaluator":{
                                        "type":"xpath",
                                        "expression":"orderAck/orderId"
                                }
                        }],
                        "properties":[{
                                "name":"customer",
                                "evaluator":{
                                        "type":"xpath",
                                        "expression":"orderAck/customer"
                                }
                        },{
```

```
                        "name":"total",
                        "evaluator":{
                                "type":"xpath",
                                "expression":"orderAck/total"
                        }
                }]
        },
        "org.switchyard.quickstarts.demos.orders.Order":{
                "contexts":[{
                        "type":"Conversation",
                        "evaluator":{
                                "type":"mvel",
                                "expression":"orderId"
                        }
                }],
                "properties":[{
                        "name":"customer",
                        "evaluator":{
                                "type":"mvel",
                                "expression":"customer"
                        }
                },{
                        "name":"itemId",
                        "evaluator":{
                                "type":"mvel",
                                "expression":"itemId"
                        }
                }]
        },
        "org.switchyard.quickstarts.demos.orders.OrderAck":{
                "contexts":[{
                        "type":"Conversation",
                        "evaluator":{
                                "type":"mvel",
                                "expression":"orderId"
                        }
                }],
                "properties":[{
                        "name":"customer",
                        "evaluator":{
                                "type":"mvel",
                                "expression":"customer"
                        }
                },{
                        "name":"total",
                        "evaluator":{
                                "type":"mvel",
                                "expression":"total"
                        }
                }]
        },
        "{urn:switchyard-quickstart-demo:orders:1.0}makePayment":
{
                "properties":[{
                        "name":"customer",
                        "evaluator":{
```

```
                                        "type":"xpath",
                                        "expression":"payment/customer"
                                }
                        },{
                                "name":"amount",
                                "evaluator":{
                                        "type":"xpath",
                                        "expression":"payment/amount"
                                }
                        }]
                },
                "{urn:switchyard-quickstart-
demo:orders:1.0}makePaymentResponse":{
                        "properties":[{
                                "name":"customer",
                                "evaluator":{
                                        "type":"xpath",
                                        "expression":"receipt/customer"
                                }
                        },{
                                "name":"amount",
                                "evaluator":{
                                        "type":"xpath",
                                        "expression":"receipt/amount"
                                }
                        }]
                },
                "org.switchyard.quickstarts.demos.orders.Receipt":{
                        "properties":[{
                                "name":"customer",
                                "evaluator":{
                                        "type":"mvel",
                                        "expression":"customer"
                                }
                        },{
                                "name":"amount",
                                "evaluator":{
                                        "type":"mvel",
                                        "expression":"amount"
                                }
                        }]
                },

"org.switchyard.quickstarts.demos.orders.ItemNotFoundException":{
                        "script":{
                                "type":"mvel",
                                "expression":"activity.fault =
\"ItemNotFound\""
                        }
                }
        }
}]
```

Table 3.4. Information Processor Example

Field	Description
name	Name of the Information Processor.
version	Version of the Information Processor. If multiple versions Information Processor are installed with the same name, only the newest version is used.
typeProcesors	The map of type processors . There is one map per type, with the type name being the map key.

The type processor element in the example is associated with a particular information type. The fields associated with this component are:

Table 3.5. Type Processor

Field	Description
contexts	List of context evaluators.
properties	List of property evaluators.
script	An optional script evaluator that is used to do any other processing that may be required, such as setting additional properties in the activity event that are not necessarily derived from message content information.
transformer	An optional transformer that determines how this information type is represented in an activity event.

Table 3.6. Context Evaluator

Field	Description
type	The context type, such as Conversation, Endpoint, Message, or Link.
timeframe	The number of milliseconds associated with a ***`Link`*** context type. If not specified, then the context is assumed to represent the destination of the link, so the source of the link must define the timeframe.
header	The optional header name. If not defined, then the expression is applied to the information content to obtain the context value.
evaluator	The expression evaluator used to derive the context value.

The context types represent different ways in which the activity events can be related to each other or to a logical grouping (for example, a business transaction). Not all activity events need to be associated directly with a global business transaction id. They can be indirectly associated based on transitive correlation. For example, if activity 1 is associated with the global business transaction id, activity 2 is associated with activity 1 by a message context type, and activity 3 is associated with activity 2 based on an endpoint correlation id, then all the three activity events will be collectively correlated to the business transaction id.

Table 3.7. Context Type

Context Type	Description
Conversation	A conversation identifier can be used to correlate activity events to a business transaction associated with a globally unique identifer such as an order id.
Endpoint	A globally unique identifier associated with one endpoint in a business transaction. For example, a process instance id associated with the business process executing within a service playing a particular role in the business transaction.
Message	A globally unique identifier of a message sent from one part to another.
Link	A temporal link between a source and a destination activity. The temporal nature of the association is intended to enable non-globally unique details to be used to correlate activities, where the id is considered unique within the defined timeframe.

Table 3.8. Property Evaluator

Field	Description
name	The property name being initialized.
header	The optional header name. If not defined, then the expression is applied to the information content to obtain the property value.
evaluator	The expression evaluator used to derive the property value.

In the context and property evaluator components, you can reference an expression evaluator to derive their value. The expression evaluator has the following fields:

Table 3.9. Expression Evaluator

Field	Description
type	The type of expression evaluator to use. Only mvel or xpath are supported.
expression	The expression to evaluate.
optional	It indicates whether the value being extracted by the expression is optional. The default is false. If a value is not optional, but the expression fails to locate a value, then an error is reported.

These expressions operate on the information being processed, to return a string value to be applied to the appropriate context or property.

The Type processor also has a Script field, which has the following fields:

Table 3.10. Script

Field	Description
type	The type of expression evaluator to use. Currently only mvel is supported.
expression	The expression to evaluate.

You can use the following variables with the MVEL script evaluator:

- information: The information being processed.
- activity: - The activity event.

  The transformer field of the Type Processor has the following fields:

Table 3.11. Transformer

Field	Descrition
type	The type of transformer to use. Currently only serialize and mvel are supported. .

The serialize transformer does not take any other properties. It attempts to convert the representation of the information into a textual form for inclusion in the activity event. So this transformer type can be used where the complete information content is required.

You can use the following variables with the MVEL transformer script:

Table 3.12. MVEL Transformer

Field	Descrition
expression	The mvel expression to transform the supplied information.

3.3.4.2. Registering the Information Processors

The Information Processors are deployed within the JEE container as a WAR file with the following structure:

```
warfile
|
|-META-INF
|    |- beans.xml
|
|-WEB-INF
|    |-classes
|    |    |-ip.json
|    |    |-<custom classes/resources>
|    |
|    |-lib
|       |-ip-loader-jee.jar
|       |-<additional libraries>
```

The **ip.json** file contains the JSON representation of the Information Processor configuration. The **ip-loader-jee.jar** acts as a bootstrapper to load and register the Information Processors. If custom classes are defined, then the associated classes and resources can be defined in the **WEB-INF/classes** folder or within additional libraries located in the **WEB-INF/lib** folder. A maven **pom.xml** that creates this structure is shown below:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>....</groupId>
        <artifactId>....</artifactId>
        <version>....</version>
        <packaging>war</packaging>
        <name>....</name>

        <properties>
                <rtgov.version>....</rtgov.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.overlord.rtgov.activity-
management</groupId>
                        <artifactId>activity</artifactId>
                        <version>${rtgov.version}</version>
                        <scope>provided</scope>
                </dependency>
```

```
                <dependency>
                        <groupId>org.overlord.rtgov.activity-
management</groupId>
                        <artifactId>ip-loader-jee</artifactId>
                        <version>${rtgov.version}</version>
                </dependency>
                ....
        </dependencies>

</project>
```

If you are deploying in JBoss Application Server, then you also need to include the following fragment to define the dependency on the core Runtime Governance modules:

```
.....
        <build>
                <finalName>....</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-war-
plugin</artifactId>
                                <configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>
                                        <archive>
                                                <manifestEntries>

<Dependencies>deployment.overlord-rtgov.war</Dependencies>
                                                </manifestEntries>
                                        </archive>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
        .....
```

3.3.4.3. Defining the Activity Validators

The Activity Validator mechanism provides the means to install event validation capabilities within the activity collection environment. In some execution environments these validators can be implicitly called as part of collecting the activity events. However, in some environments (such as, SwitchYard) these validators need to be explicitly invoked, as they may need to impact the execution behavior (that is, block the transaction). The Activity Validator then invokes the validation capability and reacts to any issues it detects.

You can define an Activity Validator as an object model or specified as a JSON representation for packaging in a suitable form, and subsequently de-serialized when deployed to the governed execution environment. The following is an example of the JSON representation of a list of Activity Validators:

```
[{
  "name" : "RestrictUsage",
  "version" : "1",
  "predicate" : {
    "@class" : "org.overlord.rtgov.ep.mvel.MVELPredicate",
    "expression" : "event instanceof
```

```
org.overlord.rtgov.activity.model.soa.RequestReceived && event.serviceType
== \"{urn:switchyard-quickstart-demo:orders:0.1.0}OrderService\""
  },
  "eventProcessor" : {
    "@class" : "org.overlord.rtgov.ep.mvel.MVELEventProcessor",
    "script" : "VerifyLastUsage.mvel",
    "services" : {
      "CacheManager" : {
        "@class" :
"org.overlord.rtgov.common.infinispan.service.InfinispanCacheManager"
      }
    }
  }
}]
```

This example illustrates the configuration of a single Activity Validator with the following top level elements:

Table 3.13. Activity Validator Example

Field	Description
name	The name of the Activity Validator.
version	The version of the Activity Validator. If multiple versions of the same named Activity Validator are installed, only the newest version is used. Versions can be expressed using Numeric, Dot Format, or any alpha numeric and symbols scheme.
predicate	The optional implementation of the **`org.overlord.rtgov.ep.Predicate`** interface, used to determine if the activity event is relevant. If relevant, the event can be supplied to the event processor.
eventProcessor	Defines the details for the event processor implementation being used. At a minimum, the value for this field should define a @class property to specify the Java class name for the event process implementation to use. Another general field that can be configured is the map of services that can be used by the event processor. Depending upon which implementation is selected, the other fields within the value apply to the event processor implementation.

When comparing versions, for example, when determining whether a newly deployed Activity Validator has a higher version than an existing one with the same name, then initially the versions are compared as numeric values. If either are not numeric, then they are compared using dot format, with each field being compared first as numeric values, and if not based on lexical comparison. If both fields do not have a dot, then they are compared lexically.

3.3.4.4. Registering the Activity Validators

The Activity Validators are deployed within the JEE container as a WAR file with the following structure:

```
warfile
|
|-META-INF
|    |- beans.xml
|
|-WEB-INF
|    |-classes
|    |    |-av.json
|    |    |-<custom classes/resources>
|    |
|    |-lib
|       |-av-loader-jee.jar
|       |-<additional libraries>
```

The **av.json** file contains the JSON representation of the Activity Validator configuration. The **av-loader-jee.jar** acts as a bootstrapper to load and register the Activity Validators. If custom classes are defined, then the associated classes and resources can be defined in the **WEB-INF/classes** folder or within additional libraries located in the **WEB-INF/lib** folder. A maven **pom.xml** that creates this structure is as shown below:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>....</groupId>
        <artifactId>....</artifactId>
        <version>....</version>
        <packaging>war</packaging>
        <name>....</name>

        <properties>
                <rtgov.version>....</rtgov.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.overlord.rtgov.activity-
management</groupId>
                        <artifactId>activity</artifactId>
                        <version>${rtgov.version}</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.overlord.rtgov.activity-
management</groupId>
                        <artifactId>av-loader-jee</artifactId>
                        <version>${rtgov.version}</version>
                </dependency>
                ....
```

```
        </dependencies>

</project>
```

If deploying in JBoss Application Server, then you also need to include the following fragment to define the dependency on the core Runtime Governance modules:

```
.....
        <build>
                <finalName>....</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-war-
plugin</artifactId>
                                <configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>
                                        <archive>
                                                <manifestEntries>

<Dependencies>deployment.overlord-rtgov.war</Dependencies>
                                                </manifestEntries>
                                        </archive>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
        .....
```

3.3.5. Reporting and Querying Activity Events via Rest

In this approach, the Runtime Governance Server manually reports the activity information from the activity events to the Runtime Governance Server through a publicly available API such as REST service. This section provides details on how the Runtime Governance Server reports the activity event information, queries the events for relevant information, retrieves an Activity Unit, and retrieves events with a context value.

3.3.5.1. Reporting Activity Information

You can report Activity Information by using the POST request to the following URL:

```
<host>/overlord-rtgov/activity/store
```

The request contains the list of ActivityUnit objects encoded in JSON. Here is an example:

```
[{
    "id":"TestId1",
    "activityTypes":[{
        "type":"RequestSent",
        "context":[{
            "value":"12345"
        },{
            "value":"abc123",
            "type":"Endpoint"
```

```
        },{
            "value":"ABC123",
            "type":"Message"
        }],
        "content":"....",
        "serviceType":"{http://service}OrderService",
        "operation":"buy",
        "fault":"MyFault",
        "messageType":"{http://message}OrderRequest",
        "timestamp":1347028592880
    },{
        "type":"ResponseReceived",
        "context":[{
            "value":"12345"
        },{
            "value":"ABC124",
            "type":"Message"
        }],
        "content":"....",
        "serviceType":"{http://service}OrderService",
        "operation":"buy",
        "fault":"OutOfStock",
        "messageType":"{http://message}OutOfStock",
        "replyToId":"ABC123",
        "timestamp":1347028593010
    }],
    "origin":{
        "host":"Saturn",
        "principal":"Fred",
        "node":"Saturn1",
        "thread":"Thread-1"
    }
},{
    .....
}]
```

3.3.5.2. Querying Activity Events Using an Expression

You can query activity events using an expression from a POST request to the following URL:

```
<host>/overlord-rtgov/activity/query
```

The request contains the JSON encoding of the Query Specification, which has the following properties:

Table 3.14. JSON Encoding Properties

Property	Description
fromTimestamp	Specifies (optionally) the start date/time for the activity units required. If not specified, then the query applies to the first recorded activity unit.

Property	Description
toTimestamp	Specifies (optionally) the end date/time for the activity units required. If not specified, then the query relates to the most recently recorded activity units.
expression	An expression (optional) that can be used to specify the activity events of interest.
format	Specifies (optionally) the format of the expression. The configured activity store must support this value.

3.3.5.3. Retrieving an Activity Unit

You can retrieve an Activity Unit from a GET request to the following URL:

```
<host>/overlord-rtgov/activity/unit?id=<unitId>
```

Here, the **`unitId`** represents the identifier associated with the retrieved ActivityUnit.

3.3.5.4. Retrieve Activity Events Associated with a Context Value

You can retrieve activity events associated with a context value from a GET request to the following URL:

```
<host>/overlord-rtgov/activity/events?type=<contextType>&value=
<identifier>
```

The service uses basic authentication by default, using a valid JBoss EAP Application Realm user (You can use the Governance user you configured when you installed Fuse 6).

Here,

- The <contextType> represents the context type, such as Conversation, Endpoint, Message or Link. For more information, see the API documentation for **`org.overlord.rtgov.activity.model.Context.Type`**.
- The <identifier> represents the correlation value associated with the ActivityType(s) that are being retrieved.

You can provide two additional optional query parameters, ***`start`*** for the start timestamp, and ***end*** for the end timestamp. You can use these parameters to scope the time period of the query.

The response is a list of ActivityType objects encoded in JSON. For more information, see the API documentation for **`org.overlord.rtgov.activity.model.ActivityType`**.

3.4. REPORTING ACTIVITY EVENT INFORMATION

The Activity Collector accumulates information from the Activity events and provides this information to the Activity Server. The Activity Server then records it and provides support for querying this information. This section describes the role of Activity Collector and Activity Server in detail.

3.4.1. Activity Collector

The Activity Collector component is embedded in the activity execution environment. It accumulates activity information from the infrastructure used in the execution of a business transaction. It then reports the activity information to the Activity Server implicitly, using an appropriate Activity Logger implementation. The default Activity Logger implementation operates efficiently by providing a batching capability to send activity information to the server based either on a regular time interval, or a maximum number of activity units, whichever occurs first.

Locating the Activity Collector depends upon your run-time environment. For example, in a JEE environment, you can locate the Activity Collector using the following code:

```
import org.overlord.rtgov.activity.collector.ActivityCollector;
import org.overlord.rtgov.activity.collector.ActivityCollectorAccessor;
....

ActivityCollector activityCollector =
ActivityCollectorAccessor.getActivityCollector();
```

The accessor is initialized with an instance of the ActivityCollector when it is instantiated by the system such as a CDI. If an instance is not initialized when this method is invoked, then the client is blocked for a short period of time as it waits for the instance. If the instance is not initialized after this period, then a null is returned.

3.4.1.1. Processing the Activity Event Information

It is necessary to process certain events in order to enable the Runtime Governance infrastructure, and the user policies and rules that are defined within it, to make most effective use of activities that are reported. The relevant information extracted from these activity events are used in:

- Correlating events to a particular business transaction instance.
- Highlighting important properties that may be useful for business policies.

The Activity Collector has an **`ActivityCollector`** API that provides a method to enable information associated with the activity event to be preprocessed. The **`ActivityCollector`** API does this with the help of information processors.

These extracted properties can be used in further event analysis, to correlate the events and enable business relevant queries. The signature for this method is:

```
public String processInformation(String processor, String type,
        Object info, java.util.Map<String, Object> headers,
                ActivityType actType);
```

Here,

- The ***processor*** parameter is an optional value that you can use to explicitly name the information processor. If not specified, then Activity Collector checks all registered information processors to determine if they are relevant for the supplied information type.
- The ***type*** parameter represents the information type. This can be in any form, as long as it matches the registered type defined in the information processor configuration.
- The ***info*** parameter represents the actual information that the Activity Collector will process.
- The ***headers*** parameter represents any header information that may have accompanied the information.
- The ***actType*** parameter represents the activity event that any extracted properties must be recorded against.

3.4.1.2. Validating the Activity Events

The Activity Collector provides a validate method for pre-processing the activity event, using configured Activity Validators, before it is submitted to the activity server. This mechanism can be used to process activity events in the execution environment, prior to it being distributed to the activity server which may be located on a separate server. It can also be used to identify invalid situations, resulting in an exception. The execution environment can handle these exceptions and use them to block the business transaction associated with the activity event.

3.4.1.3. Managing the Activity Scope

An Activity Scope is a way of grouping a range of different activity types, that are reported to the activity server, into a single logical unit. It represents the same scope as an eXtended Architecture (XA) transaction, which means, it encompasses all the activity types within the range or discards it completely if the transaction is rolled back.

When the first activity is reported within the scope of a XA transaction, then the scope automatically starts. When that transaction subsequently commits, the Activity Unit (that is, the collection of activities accumulated during that scope) is reported to the Activity Server. However, if the activities are performed outside the scope of an XA transaction, then the component reporting the activity information can either explicitly start a scope, or just report the activity information. If no scope exists, and an activity type is reported, then it is reported to the activity server as a single event. The disadvantage of this approach is that it is less efficient, both in terms of reporting due to the duplication of certain header information, and for subsequent analysis. Having multiple activity events defined in a single unit related to the transaction, provides added value to interrelating the different event. This also provides some implied correlation that would not exist if the events were independently reported to the Activity Server.

3.4.1.4. Starting and Ending the Activity Scope

1. **Starting an Activity Scope**
   To start the scope, invoke the **`startScope`** method on the Activity Collector as shown below:

   ```
   activityCollector.startScope();
   ```

   If the application does not know whether a scope has already been started, and only wishes to start a single scope, then use the following condition:

   ```
   boolean started=false;
   if (!activityCollector.isScopeActive()) {
   ```

```
    activityCollector.startScope();
    started = true;
}
```

Here, the **isScopeActive** method returns a boolean value to indicate whether the scope was previously started. If it returns true, then this component is also responsible for stopping the scope. If it returns false, then it means the scope has already been started, and therefore the component must not invoke the **endScope** method.

2. **Ending an Activity Scope**
   To stop the scope, invoke the **endScope** method on the Activity Collector as shown below:

```
if (started) {
    activityCollector.endScope();
}
```

3.4.1.5. Reporting an Activity Type

The Activity Collector reports the activity event information to the server as an Activity Unit. An Activity Unit contains one or more actual activity events. The activity event is generically known as an Activity Type.

The Activity Collector mechanism removes the need for each component to report general information associated with the Activity Unit, and instead is only responsible for reporting the specific details associated with the situation that has occurred.

Procedure 3.2. Reporting an Activity Event

1. To report an event, create the specific Activity Type and invoke the record method as shown below:

```
org.overlord.rtgov.activity.model.RequestSent sentreq=new
org.overlord.rtgov.activity.model.soa.RequestSent();

sentreq.setServiceType(serviceType);
sentreq.setOperation(opName);
sentreq.setContent(content);
sentreq.setMessageType(mesgType);
sentreq.setMessageId(messageId);

activityCollector.record(sentreq);
```

2. For certain types of event, it may also be appropriate to invoke an information processor(s) to extract relevant context and property information, that can then be associated with the activity event. To achieve this, use the **processInformation**() method as shown below:

```
Object modifiedContent=_activityCollector.processInformation(null,
mesgType, content, headers, sentreq);

sentreq.setContent(modifiedContent);
```

3.4.1.6. Configuring an Activity Unit Logger

When the Activity Collector invokes the **endScope** method, the Activity Unit Logger component logs

the activity unit that is generated as a result. The Activity Unit Logger interface has three methods:

- **`init`**: This method initializes the activity unit logger implementation
- **`log`**: This method is used for the Activity Unit logging.
- **`close`**: This method closes the activity unit logger implementation.

The Batched Activity Unit Logger abstract base class implements the Activity Unit Logger interface. It provides the functionality to batch Activity Unit instances, and forwards them based on two properties:

- Maximum Time Interval: If the time interval expires, then it sends the set of Activity Units to the server.
- Maximum Unit Count: If the number of Activity Units reaches this maximum value, then it sends the batch to the server.

You can initialize this implementation explicitly in an embedded environment. In a JEE environment, use the PostConstruct and PreDestroy annotations to enable it for implicit initialization.

Additionally, Activity Unit Logger interface implements the Activity Server Logger, which is derived from the Batched Activity Unit Logger. The Activity Server Logger, therefore, sends activity information in a batch periodically based on the configured properties. When the batch of Activity Units are sent, this implementation forwards them to an implementation of the Activity Server interface, injected explicitly or implicitly into the logger. This can be used to either send the events directly to the Activity Server component, if it is located within the same server, or via a remote binding.

The following example shows a situation where an embedded Activity Collector is being initialized with an Activity Server Logger, which uses the REST Activity Server client implementation:

```
import org.overlord.rtgov.activity.collector.ActivityCollector;
import
org.overlord.rtgov.activity.collector.activity.server.ActivityServerLogger
;
import org.overlord.rtgov.activity.server.rest.client.RESTActivityServer;

.....

        RESTActivityServer restc=new RESTActivityServer();
        restc.setServerURL(_activityServerURL);

        ActivityServerLogger activityUnitLogger=new
ActivityServerLogger();
        activityUnitLogger.setActivityServer(restc);

        activityUnitLogger.init();

        _collector.setActivityUnitLogger(activityUnitLogger);
```

3.4.1.7. Collector Context Configuration

The Collector Context component of the Collector architecture is an interface that provides the Activity Collector with information about the environment such as principal, host, node, and port. This information can be used to complete the Origin information within an Activity Unit, as well as providing access to capabilities required such as the Transaction Manager, from the environment. Each type of

environment in which the collector may be used provides an implementation of this interface. Depending upon the environment, this is either implicitly injected into the Activity Collector, or set explicitly using the setter method.

3.4.1.8. ActivityReporter Component for Simplified Reporting

Apart from using the Activity Collector mechanism for collective activity events, SwitchYard also provides an injectable component called ActivityReporter. The ActivityReporter component enables applications to perform simple activity reporting tasks. If injection is not possible, you can instantiate a default implementation of the interface. For example, the sample SwitchYard order management application uses this approach as shown below:

```
@Service(InventoryService.class)
public class InventoryServiceBean implements InventoryService {

    private final Map<String, Item> _inventory = new HashMap<String, Item>
();

    private org.overlord.rtgov.client.ActivityReporter _reporter=
                new org.overlord.rtgov.client.DefaultActivityReporter();

    public InventoryServiceBean() {
        ....
    }

    @Override
    public Item lookupItem(String itemId) throws ItemNotFoundException {
        Item item = _inventory.get(itemId);

        if (item == null) {

            if (_reporter != null) {
                _reporter.logError("No item found for id '"+itemId+"'");
            }

            throws new ItemNotFoundException("No item found for id " +
itemId);
        }

        ....

        return item;
    }
}
```

The ActivityReporter enables the application to perform the following tasks:

Table 3.15. ActivityReporter Methods

Method	Description
logInfo(String mesg)	Log some information
logWarning(String meg)	Log a warning

Method	Description
logError(String mesg)	Log an error
report(String type, Map</String,String> props)	Record a custom activity with a particular type and associated properties
report(ActivityType activity)	Record an activity

Note that, you can not use this API to control the scope of an ActivityUnit. This API is purely intended to simplify the approach used for reporting additional incidental activities from within an application and you must use other parts of the infrastructure to handle the scope control of ActivityUnit.

Here is the maven dependency required to access the ActivityReporter:

```
<dependency>
      <groupId>org.overlord.rtgov.integration</groupId>
      <artifactId>rtgov-client</artifactId>
      <version>${rtgov.version}</version>
</dependency>
```

3.4.2. Activity Server

Activity Units describe the activities that occur during the execution of business transactions in a distributed environment. An Activity Server records these Activity Units and also provides query support to retrieve previously recorded Activity Units.

3.4.2.1. Recording Activity Units

The Activity Units represent the logical grouping of individual situations that occur within a transaction boundary. The Activity Server records a list of Activity Units using the following approaches:

- Direct Injection

  This is the simplest approach is to leverage CDI (for example within a JEE container) to directly inject the Activity Server implementation. Here is how you can use direct injection to get reference to the Activity Server:

  ```
  import org.overlord.rtgov.activity.server.ActivityServer;

  ....

  @Inject
  private ActivityServer _activityServer=null;
  ```

  Once you obtain the reference to the Activity Server, you can call the store method to record a list of Activity Units as shown below:

  ```
  import org.overlord.rtgov.activity.model.soa.RequestSent;
  import org.overlord.rtgov.activity.model.ActivityUnit;
  ```

```
....

java.util.List<ActivityUnit> list=new .....;

RequestSent act=new RequestSent();
act.setServiceType(...);
...

list.add(act);

_activityServer.store(list);
```

- REST Service

  You can access the Activity Server as RESTful service. here is an example:

```
import org.codehaus.jackson.map.ObjectMapper;
import org.overlord.rtgov.activity.model.ActivityUnit;


.....

        java.util.List<ActivityUnit> activities=........
        java.net.URL storeUrl = new java.net.URL(....);
        // <host>/overlord-rtgov/activity/store

        java.net.HttpURLConnection connection =
(java.net.HttpURLConnection) storeUrl.openConnection();

        String userPassword = username + ":" + password;
        String encoding =
org.apache.commons.codec.binary.Base64.encodeBase64String(userPasswo
rd.getBytes());

        connection.setRequestProperty("Authorization", "Basic " +
encoding);

        connection.setRequestMethod("POST");
        connection.setDoOutput(true);
        connection.setDoInput(true);
        connection.setUseCaches(false);
        connection.setAllowUserInteraction(false);
        connection.setRequestProperty("Content-Type",
"application/json");

        java.io.OutputStream os=connection.getOutputStream();

        ObjectMapper mapper=new ObjectMapper();
        // Use jackson to serialize the activity units
        mapper.writeValue(os, activities);

        os.flush();
        os.close();

        java.io.InputStream is=connection.getInputStream();
```

```
        byte[] result=new byte[is.available()];

        is.read(result);
        is.close();
```

NOTE

Use the Activity Collector mechanism to aggregate and record the activity information. It is very efficient as each system individually reports events to the server.

3.4.2.2. Querying Activity Units

You can use the Activity Server to query a list of Activity Units that meet a supplied query specification. The Activity Server queries Activity Units using the following approaches:

- Direct Injection

  The is the simplest approach is to leverage CDI to obtain a reference to the Activity Server. Once you obtain a reference to the Activity Server, then build the query specification based on the relevant criteria, and call the query method to retrieve the list of Activity Units as shown below:

```
import org.overlord.rtgov.activity.model.ActivityUnit;
import org.overlord.rtgov.activity.model.Context;
import org.overlord.rtgov.activity.server.QuerySpec;

....

QuerySpec qs=new QuerySpec()
                .setFromTimestamp(...)
                .setToTimestamp(...)
                .addContext(new
Context(Context.CONVERSATION_ID,"txnId","123"));


java.util.List<ActivityUnit> list=_activityServer.query(qs);
```

- REST Service

  You can access the Activity Server as RESTful service. here is an example:

```
import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.type.TypeReference;
import org.overlord.rtgov.activity.server.QuerySpec;
import org.overlord.rtgov.activity.model.ActivityUnit;


.....

        QuerySpec qs=........
        java.net.URL queryUrl = new java.net.URL(....);
        // <host>/overlord-rtgov/activity/query
```

```
        java.net.HttpURLConnection connection =
(java.net.HttpURLConnection) queryUrl.openConnection();

        String userPassword = username + ":" + password;
        String encoding =
org.apache.commons.codec.binary.Base64.encodeBase64String(userPasswo
rd.getBytes());

        connection.setRequestProperty("Authorization", "Basic " +
encoding);

        connection.setRequestMethod("POST");
        connection.setDoOutput(true);
        connection.setDoInput(true);
        connection.setUseCaches(false);
        connection.setAllowUserInteraction(false);
        connection.setRequestProperty("Content-Type",
"application/json");

        java.io.OutputStream os=connection.getOutputStream();

        ObjectMapper mapper=new ObjectMapper();
        // Use jackson to serialize the query spec
        mapper.writeValue(os, qs);

        os.flush();
        os.close();

        java.io.InputStream is=connection.getInputStream();

        java.util.List<ActivityUnit> activities =
mapper.readValue(is, new TypeReference<java.util.List<ActivityUnit>>
() {});

        is.close();
```

3.5. PROCESSING ACTIVITY EVENTS

You can use the Event Processor and supporting components directly within the Activity Collection mechanism or from nodes within an Event Processor Network. This section discusses Event Processors, Event Processor Network, Predicates and how you can define custom processors and predicates.

3.5.1. Configuring an Event Processor Network

3.5.1.1. Defining the Event Processor Network

An EPN (Event Processor Network) processes a stream of events through a network of linked nodes established to perform specific filtering, transformation or analysis tasks. You can define this network as an object model or specify it as a JSON representation for packaging in a suitable form, and subsequently de-serialized when deployed to the Runtime Governance server.

The following is an example of the JSON representation of an Event Processor Network. This example defines the out of the box Event Processor Network installed with the distribution:

```
{
  "name" : "Overlord-RTGov-EPN",
  "version" : "1.0.0.Final",
  "subscriptions" : [ {
    "nodeName" : "SOAEvents",
    "subject" : "ActivityUnits"
  },
  {
    "nodeName" : "ServiceDefinitions",
    "subject" : "ActivityUnits"
  },
  {
    "nodeName" : "SituationsStore",
    "subject" : "Situations"
  } ],
  "nodes" : [
    {
      "name" : "SOAEvents",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ "SOAEvents" ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" :
"org.overlord.rtgov.content.epn.SOAActivityTypeEventSplitter"
      },
      "predicate" : null,
      "notifications" : [ ]
    },{
      "name" : "ServiceDefinitions",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" :
"org.overlord.rtgov.content.epn.ServiceDefinitionProcessor"
      },
      "predicate" : null,
      "notifications" : [ {
        "type" : "Results",
        "subject" : "ServiceDefinitions"
      } ]
    },{
      "name" : "ServiceResponseTimes",
      "sourceNodes" : [ "ServiceDefinitions" ],
      "destinationSubjects" : [ "ServiceResponseTimes" ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" :
"org.overlord.rtgov.content.epn.ServiceResponseTimeProcessor"
      },
      "predicate" : null,
      "notifications" : [ {
        "type" : "Results",
```

```
        "subject" : "ServiceResponseTimes"
      } ]
    },{
      "name" : "SituationsStore",
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" : "org.overlord.rtgov.ep.jpa.JPAEventProcessor",
        "entityManager" : "overlord-rtgov-epn-non-jta"
      }
    }
  ]
}
```

3.5.1.2. Event Processor Network Example

This example illustrates the configuration of a service associate with the event processor, as well as a predicate:

```
{
  "name" : "AssessCreditPolicyEPN",
  "version" : "1",
  "subscriptions" : [ {
    "nodeName" : "AssessCredit",
    "subject" : "SOAEvents"
  } ],
  "nodes" : [
    {
      "name" : "AssessCredit",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "predicate" : {
        "@class" : "org.overlord.rtgov.ep.mvel.MVELPredicate",
        "expression" : "event.serviceProvider && !event.request &&
event.serviceType == \"{urn:switchyard-quickstart-
demo:orders:0.1.0}OrderService\""
      },
      "eventProcessor" : {
        "@class" : "org.overlord.rtgov.ep.mvel.MVELEventProcessor",
        "script" : "AssessCredit.mvel",
        "services" : {
          "CacheManager" : {
            "@class" :
"org.overlord.rtgov.common.infinispan.service.InfinispanCacheManager"
          }
        }
      }
    }
  ]
}
```

Here,

Table 3.16. Top Level Elements

Filed	Description
name	The name of the network.
subscriptions	The list of subscriptions associated with the network.
nodes	The nodes that form the connected graph within the network.
version	The version of the network.

Subscriptions

The subscription element is used to define a subject that the network is interested in, and the name of the node to which the events from that subject should be routed. This decoupled subscription approach enables multiple networks to register their interest in events from the same subject. Multiple nodes within the same network can subscribe to the same subject. The fields associated with this component are:

Table 3.17. Fields Associated with Subscriptions

Field	Description
subject	The subject to subscribe to.
nodeName	The name of the node within the network to route the events to.

This is a list of the subjects that are reserved for Red Hat JBoss Governance's use:

Table 3.18. Subjects Reserved for Governance

Subject	Purpose
ActivityUnits	This subject is used to publish events of the type **`org.overlord.rtgov.activity.model.ActivityUnit`**, produced when activity information is recorded with the Activity Server.

Node

This element is used to define a particular node in the graph that forms the network. The fields associated with this component are:

Table 3.19. Node Fields

Field	Description
name	The name of the node.
sourceNodes	A list of node names that represent the source nodes, within the same network that this node receives its events from. Therefore, if this list is empty, it means that the node is a root node and should be the target of a subscription.
destinationSubjects	A list of inter-EPN subjects to publish any resulting events to. These subjects are only of relevance to other networks.
maxRetries	The maximum number of times an event should be retried, following a failure, before giving up on the event.
retryInterval	The delay that should occur between retry attempts. This may not be supported in some environments.
eventProcessor	Defines the details for the event processor implementation being used. At a minimum, the value for this field should define a @class property to specify the Java class name for the event process implementation to use. Another general field that can be configured is the map of services that can be used by the event processor. Depending upon which implementation is selected, the other fields within the value apply to the event processor implementation.
predicate	This field is optional, but if specified, it defines a predicate implementation. As with the event processor, it must at a minimum define a @class field that specifies the Java class name for the implementation, with any additional fields be used to initialize the predicate implementation.
notifications	A list of notifications. A notification entry defines its type and the notification subject upon which the information should be published. Unlike the destinationSubjects, which are subjects for inter-EPN communication, these notification subjects are the mechanism for distribution information out of the EPN capability, for presentation to end-users through various means.

The notify types field defines what type of notifications should be emitted from a node when processing an event. The notifications are the mechanism used by potentially interested applications to observe what information each node is processing, and the results they produce.

The possible values for this field are:

Table 3.20. Notification Values

Field	Description
Processed	This type indicates that a notification should be created when an event is considered suitable for processing by the node. An event is suitable either if no predicate is defined, or if the predicate indicates the event is valid.
Results	This type indicates that a notification should be created for any information produced as the result of the event processor processing the event.

NOTE

Notification is the mechanism for making information processed by the Event Processor Network accessible by interested parties. If a notification is not defined for a node, then it will only be used for internal processing, potentially supplying the processed event to other nodes in the network (or other networks, if destination subject(s) are specified).

3.5.1.3. Registering the Event Processor Network

The Event Processor Network is deployed within the JEE container as a WAR file with the following structure:

```
warfile
|
|-META-INF
|    |- beans.xml
|
|-WEB-INF
|    |-classes
|    |    |-epn.json
|    |    |-<custom classes/resources>
|    |
|    |-lib
|       |-epn-loader-jee.jar
|       |-<additional libraries>
```

The **epn.json** file contains the JSON representation of the Event Processor Network configuration. The **epn-loader-jee.jar** acts as a bootstrapper to load and register the Event Processor Network. If custom predicates or event processors are defined, then the associated classes and resources can be defined in the **WEB-INF/classes** folder or within additional libraries located in the **WEB-INF/lib** folder. A maven **pom.xml** that creates this structure is shown below:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
```

```
        <modelVersion>4.0.0</modelVersion>
        <groupId>....</groupId>
        <artifactId>....</artifactId>
        <version>....</version>
        <packaging>war</packaging>
        <name>....</name>

        <properties>
                <rtgov.version>....</rtgov.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.overlord.rtgov.event-processor-
network</groupId>
                        <artifactId>epn-core</artifactId>
                        <version>${rtgov.version}</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.overlord.rtgov.event-processor-
network</groupId>
                        <artifactId>epn-loader-jee</artifactId>
                        <version>${rtgov.version}</version>
                </dependency>
                ....
        </dependencies>

</project>
```

If you are deploying in JBoss Application Server, then you also need to include the following fragment to define the dependency on the core Runtime Governance modules:

```
.....
        <build>
                <finalName>slamonitor-epn</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-war-
plugin</artifactId>
                                <configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>
                                        <archive>
                                                <manifestEntries>

<Dependencies>deployment.overlord-rtgov.war</Dependencies>
                                                </manifestEntries>
                                        </archive>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
        .....
```

3.5.1.4. Supporting Multiple Versions of the Event Processor Network

Event Processor Networks define a version number that can be used to keep track of the evolution of changes in a network. When a network is deployed to a container, and used to process events, a newer version of the network can be deployed along side the existing version to ensure there is continuity in the processing of the event stream. New events presented to the network are processed by the most recent version, while events still being processed by a particular version of the network, continue to be processed by the same version, thus ensuring that changes to the internal structure of the network do not impact events that are mid-way through being processed by the network. You can determine when an older version of the network last processed an event, and therefore when an older version has been inactive for a suitable amount of time, it can be unregistered.

3.5.2. Event Processor Implementations

3.5.2.1. Introduction

SwitchYard provides the following out of the box Event Processor implementations:

- Drools Event Processor
- JPA Event Processor
- Mail Event Processor
- MVEL Event Processor

Details about these out of the box implementations and how you can configure custom event processors are described in this section.

3.5.2.2. Drools Event Processor

The Drools Event Processor implementation (**`org.overlord.rtgov.ep.drools.DroolsEventProcessor`**) enables events to be processed by a Complex Event Processing (CEP) rule. This implementation defines the following additional fields:

Table 3.21. Drools Event Processor

Field	Description
ruleName	The name of the rule, used to locate the rule definition in a file called **`<ruleName>.drl`**

Here is an example of such a rule:

```
import org.overlord.rtgov.activity.model.soa.RequestReceived
import org.overlord.rtgov.activity.model.soa.ResponseSent

global org.overlord.rtgov.ep.EPContext epc

declare RequestReceived
    @role( event )
    @timestamp( timestamp )
    @expires( 2m20s )
end
```

```
declare ResponseSent
    @role( event )
    @timestamp( timestamp )
    @expires( 2m20s )
end

rule "correlate request and response"
when
    $req : RequestReceived( $id : messageId ) from entry-point
"Purchasing"
    $resp : ResponseSent( replyToId == $id, this after[0,2m20s] $req )
from entry-point "Purchasing"
then

    epc.logInfo("REQUEST: "+$req+" RESPONSE: "+$resp);

    java.util.Properties props=new java.util.Properties();
    props.put("requestId", $req.getMessageId());
    props.put("responseId", $resp.getMessageId());

    long responseTime=$resp.getTimestamp()-$req.getTimestamp();

    epc.logDebug("CORRELATION on id '"+$id+"' response time
"+responseTime);

    props.put("responseTime", responseTime);

    epc.handle(props);

end
```

This is an example of a rule used to correlate request and response events. When a correlation is found, then a ResponseTime object is created and forwarded to the Event Processor Network for further processing using the handle method. The source of the events into the rule are called entry points, where the name relates to the source node or subject that supplies the events. The rule has access to external capabilities through the EPContext , which is defined in the statements such as:

```
global org.overlord.rtgov.ep.EPContext epc
```

EPContext is used at the end of the above example to handle the result of the event processing (that is, to forward a derived event back into the network.

If an error occurs, that requires the event to be retried (within the Event Processor Network), or the business transaction blocked (when used as a synchronous policy), then the rule can either throw an exception or return the exception as the result using the **`handle()`** method.

WARNING

Temporal rules do not currently work in a clustered environment. This is because correlation between events occurs in working memory, which is not shared across servers. Therefore, for the correlation to work, all relevant events must be received by a single server.

3.5.2.3. JPA Event Processor

A JPA based Event Processor implementation (`org.overlord.rtgov.ep.jpa.JPAEventProcessor`) enables events to be persisted. This implementation defines the following additional fields:

Table 3.22. Additional Fields for JPA Event Processor Implementation

Field	Description
entityManager	The name of the entity manager to be used.

3.5.2.4. Mail Event Processor

A mail based Event Processor implementation (`org.overlord.rtgov.ep.mail.MailEventProcessor`) enables events to be transformed and sent as an email. This implementation defines the following additional fields:

Table 3.23. Additional Fields for Mail Event Processor Implementation

Field	Description
from	The from email address.
to	The list of to email addresses.
subjectScript	The location of the MVEL script, which may be relative to the classpath, used to define the email subject.
contentScript	The location of the MVEL script, which may be relative to the classpath, used to define the email content.
contentType	The optional type of the email content. By default it will be "text/plain".
jndiName	The optional JNDI name locating the JavaMail session.

3.5.2.5. MVEL Event Processor

A MVEL based Event Processor implementation (`org.overlord.rtgov.ep.mvel.MVELEventProcessor`) enables events to be processed by a MVEL script. This implementation defines the following additional fields:

Table 3.24. MVEL Event Processor Fields

Field	Description
script	The location of the MVEL script, which may be relative to the classpath.

The script will have access to the following variables:

Table 3.25. Script Variables

Variable	Description
source	The name of the source node or subject upon which the event was received.
event	The event to be processed.
retriesLeft	The number of retries remaining.
epc	The EP context (org.overlord.rtgov.ep.EPContext), providing some utility functions for use by the script, including the handle method for pushing the result back into the network.

If an error occurs, that requires the event to be retried (within the Event Processor Network), or the business transaction blocked (when used as a synchronous policy), then the script can return the exception as the result using the `handle()` method.

3.5.2.6. Creating Custom Event Processors

The `org.overlord.rtgov.ep.EventProcessor` abstract class is responsible for processing an event routed to a particular node within the Event Processor Network. To create a custom implementation, derive a class from the `EventProcessor` abstract class. This class provides the following methods:

Table 3.26. Custom Event Processors

Method	Description
`java.util.Map<String,Service> getServices()`	This method returns the map of services available to the Event Processor.

Method	Description
`void setServices(java.util.Map<String,Service> services)`	This method sets the map of services available to the Event Processor.
`void init()`	This method is called when the event processor is first initialized as part of the Event Processor Network. A custom implementation does not need to override this method if not required.
`Serializable process(String source, Serializable event, int retriesLeft) throws Exception`	This method processes the supplied event, indicating the source of the event and how many retries are left (so that suitable error handling can be performed in no more retries remain.

3.5.2.7. Support Service by Cache Manager

This section describes the Cache Manager that enables event processors to store and retrieve information in named caches. The Cache Manager provides the following methods:

Table 3.27. API

Method	Description
`<K,V> Map<K,V> getCache(String name)`	This method returns the cache associated with the supplied name. If the cache does not exist, then a null will be returned.
`boolean lock(String cacheName, Object key)`	This method locks the item, associated with the supplied key, in the named cache.

Table 3.28. Implementations

Implementation	Class Name	Description
Infinispan	org.overlord.rtgov.common.infinispan.service.InfinispanCacheManager	This class provides an implementation based on Infinispan. It has a property called container which represents the optional JNDI name for the infinispan container defined in the `standalone-full.xml` or `standalone-full-ha.xml` file.

The Infinispan Container is obtained in one of these three possible ways:

- If the container is explicitly defined, then it is used.

- If the container is not defined, then a default container is obtained from the `$JBOSS_HOME/standalone/configuration/overlord-rtgov.properties` file for the `infinispan.container` property.
- if no default container is defined, then a default cache manager is created.

3.5.2.8. Creating Custom Services

The `org.overlord.rtgov.common.service.Service` abstract class is used to provide services for use by event processors, such as a CacheManager. To create a custom implementation, derive a class from the `Service` abstract class. This class provides the following methods:

Table 3.29. Custom Service Methods

Method	Description
`void init()`	This method is called when the service is first initialized. A custom implementation does not need to override this method if it is not required.

3.5.3. Predicates

3.5.3.1. MVEL Predicate

An MVEL based Predicate implementation (`org.overlord.rtgov.ep.mvel.MVELPredicate`) enables events to be evaluated by a MVEL expression or script. This implementation defines the following additional fields:

Table 3.30. MVEL Predicate

Field	Description
expression	The MVEL expression used to evaluate the event.
script	The location of the MVEL script, which may be relative to the classpath.

WARNING

You can define only one of these fields in your implementation.

3.5.3.2. Creating Custom Predicates

The `org.overlord.rtgov.ep.Predicate` abstract class is responsible for determining whether an event is suitable to be processed by a particular node within the Event Processor Network. To create a custom implementation, derive a class from the Predicate abstract class. This class provides the following methods:

Table 3.31. Methods of Predicate Class

Method	Description
`void init()`	This method is called when the predicate is first initialized as part of the Event Processor Network. A custom implementation does not need to override this method if not required.
`boolean evaluate(Object event)`	This method determines whether the supplied event must be processed by the node.

3.5.4. Packaging

The custom predicate or event processor implementations must be available to the classloader when an Event Processor Network or Activity Validator referencing the implementations is loaded. You can achieve this either by packaging the implementations with the Event Processor Network or Activity Validator configuration, or by installing them in a common location used by the container in which the Event Processor Network or Activity Validator is being loaded.

3.6. ACCESSING DERIVED INFORMATION THROUGH ACTIVE COLLECTIONS

An Active Collection is similar to a standard collection, with an additional ability to report change notifications when items are inserted, updated or removed. The other main difference is that you can not directly update them. An Active Collection Source manages their contents and acts as an adapter between the collection and the originating source of the information.

The Active Collection mechanism provides a means of actively managing a collection of information. This section explains how to:

- Configure Active Collections
- Present Results from an Event Processor Network
- Publish Active Collection Contents as JMX Notifications
- Query Active Collections via REST
- Use Pre-Defined Active Collections
- Implement an Active Collection Source, which can be used to subscribe to a source of information which can result in data being inserted, updated and removed from an associated active collection.
- Implement an Active Change Listener that can associated with an Active Collection Source, and automatically notified of changes to an associated Active Collection.
- Write a custom application for accessing Active Collections.

3.6.1. Configuring Active Collections

3.6.1.1. Defining the Active Collection Source

You can define an Active Collection Source as an object model or specified as a JSON representation for packaging in a suitable form, and subsequently de-serialized when deployed to the Runtime Governance server. The following is an example of the JSON representation that defines a list of Active Collection Sources, so that more than one source can be specified with a single configuration:

```
[
  {
    "@class" :
"org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "ServiceResponseTimes",
    "type" : "List",
    "itemExpiration" : 0,
    "maxItems" : 100,
    "subject" : "ServiceResponseTimes",
    "aggregationDuration" : 1000,
    "groupBy" : "serviceType + \":\" + operation + \":\" + fault",
    "aggregationScript" : "AggregateServiceResponseTime.mvel"
  },{
    "@class" :
"org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "ServiceDefinitions",
    "type" : "Map",
    "itemExpiration" : 0,
    "maxItems" : 100,
    "subject" : "ServiceDefinitions",
    "scheduledScript" : "TidyServiceDefinitions.mvel",
    "scheduledInterval" : 60000,
    "properties" : {
        "maxSnapshots" : 5
    },
    "maintenanceScript" : "MaintainServiceDefinitions.mvel"
  },{
    "@class" :
"org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "Situations",
    "type" : "List",
    "itemExpiration" : 40000,
    "maxItems" : 0,
    "subject" : "Situations",
    "activeChangeListeners" : [ {
        "@class" : "org.overlord.rtgov.active.collection.jmx.JMXNotifier",
        "objectName" : "overlord.rtgov.services:name=Situations",
        "descriptionScript" : "SituationDescription.mvel",
        "insertTypeScript" : "SituationType.mvel"
        } ],
        "derived": [ {
                "name": "FilteredSituations",
                "predicate": {
                        "type": "MVEL",
                        "expression": "map =
context.getMap(\"IgnoredSituationSubjects\"); if (map == null) { return
false; } return !map.containsKey(subject);"
                },
            "properties" : {
                "active" : false
            }
```

```
        } ]
  },{
    "@class" :
"org.overlord.rtgov.active.collection.ActiveCollectionSource",
    "name" : "IgnoredSituationSubjects",
    "type" : "Map",
    "lazy" : true,
    "factory" : {
        "@class" :
"org.overlord.rtgov.active.collection.infinispan.InfinispanActiveCollectio
nFactory",
        "cache" : "IgnoredSituationSubjects"
    }
  },{
    "@class" :
"org.overlord.rtgov.active.collection.ActiveCollectionSource",
    "name" : "Principals",
    "type" : "Map",
    "lazy" : true,
    "visibility" : "Private",
    "factory" : {
        "@class" :
"org.overlord.rtgov.active.collection.infinispan.InfinispanActiveCollectio
nFactory",
        "cache" : "Principals"
    }
  }
]
```

This configuration shows the definition of multiple Active Collection Sources. The top level elements for a source, that are common to all active collection sources, are:

Table 3.32. Elements of Active Collection Sources

Field	Description
@class	This attribute defines the Java class implementing the Active Collection Source. This class must be directly or indirectly derived from **`org.overlord.rtgov.active.collection.ActiveCollectionSource`**.
name	The name of the Active Collection that will be created and associated with this source.
type	The type of active collection. The currently supported values (as defined in the **`org.overlord.rtgov.active.collection.ActiveCollectionType`** enum are List (default) and Map.

Field	Description
visibility	This value specifies the visibility of active collection, that is, whether an active collection is accessible via the remote access mechanisms such as REST or not. The currently supported values (as defined in the `org.overlord.rtgov.active.collection.ActiveCollectionVisibility` enum are Public (default) and Private.
lazy	This value specifies whether active collection must be created on startup, or instantiated upon first use. The default value is false.
itemExpiration	If not zero, then defines the number of milliseconds until an item in the collection expires and is removed.
maxItems	If not zero, defines the maximum number of items that the collection must hold. If an insertion causes the size of the collection to increase above this value, then the oldest item must be removed.
aggregationDuration	The duration (in milliseconds) over which the information is aggregated.
groupBy	An expression defining the key to be used to categorize the information being aggregated. The expression can use properties associated with the information being aggregated.
aggregationScript	The MVEL script used for aggregating the information.
scheduledInterval	The interval (in milliseconds) between the invocation of the scheduled script.
scheduledScript	The MVEL script invoked at a fixed interval to perform routine tasks on the collection.
maintenanceScript	By default, events received by the active collection source are inserted into the associated active collection. If a MVEL maintenance script is specified, then it is invoked to manage the way in which the received information is applied to the active collection.
properties	A set of properties that can be accessed by the various scripts.

Field	Description
derived	An optional list of definitions for derived collections that will be created with the top level active collection, and retained regardless of whether any users are currently accessing them. Normally when a derived collection is created dynamically on demand, once it has served its purpose, it is cleaned up.
activeChangeListeners	The list of active change listeners that must be instantiated and automatically registered with the Active Collection. The listeners must be derived from the Java class **`org.overlord.rtgov.active.collection.AbstractActiveChangeListener`**.
factory	The optional factory for creating the active collection, derived from the class **`org.overlord.rtgov.active.collection.ActiveCollectionFactory`**.

Scripts

The aggregation script is used to aggregate information being provided by the source, before being applied to the collection. The values available to the MVEL script are:

Table 3.33. Aggregate Script Values

Variable	Description
events	The list of events to be aggregated.

The scheduled script is used to perform regular tasks on the active collection, independent of any information being applied to the collection. The values available to the MVEL script are:

Table 3.34. Scheduled Script Values

Variable	Description
acs	The active collection source.
acs.properties	The properties configured for the active collection source.
variables	A map associated with the active collection source that can be used by the scripts to cache information.

The maintenance script is used to manage how new information presented to the source is applied to the active collection. If no script is defined, then the information is inserted by default. The values available to the MVEL script are:

Table 3.35. Scheduled Script Values

Variable	Description
acs	The active collection source.
acs.properties	The properties configured for the active collection source.
key	The key for the information being inserted. You can have null value for key.
value	The value for the information being inserted.
variables	A map associated with the active collection source that can be used by the scripts to cache information.

Here is an example script, showing how you can use these variables :

```
int maxSnapshots=acs.properties.get("maxSnapshots");

snapshots = variables.get("snapshots");

if (snapshots == null) {
        snapshots = new java.util.ArrayList();
        variables.put("snapshots", snapshots);
}

// Update the current snapshot
currentSnapshot = variables.get("currentSnapshot");

if (currentSnapshot == null) {
        currentSnapshot = new java.util.HashMap();
}

snapshots.add(new java.util.HashMap(currentSnapshot));

currentSnapshot.clear();

// Remove any snapshots above the number configured
while (snapshots.size() > maxSnapshots) {
        snapshot = snapshots.remove(0);
}

// Merge snapshots
merged =
org.overlord.rtgov.analytics.util.ServiceDefinitionUtil.mergeSnapshots(s
napshots);
```

```
// Update existing, and remove definitions no longer relevant
foreach (entry : acs.activeCollection) {
        org.overlord.rtgov.analytics.service.ServiceDefinition sd=null;

        if (merged.containsKey(entry.key)) {
                acs.update(entry.key, merged.get(entry.key));
        } else {
                acs.remove(entry.key, entry.value);
        }

        merged.remove(entry.key);
}

// Add new definitions
for (key : merged.keySet()) {
        acs.insert(key, merged.get(key));
}
```

This example shows the script accessing the Active Collection Source and its properties, as well as accessing (and updating) the variables cache associated with the source.

Active Change Listeners

The activeChangeListeners element defines a list of Active Change Listener implementations that is instantiated and registered with the active collection. The fields associated with this component are:

Table 3.36. Active Change Listener Fields

Field	Description
@class	The Java class that provides the listener implementation and is directly or indirectly derived from **`org.overlord.rtgov.active.collection.AbstractActiveChange`**.

Factory

The factory element defines an Active Collection Factory implementation that will be used to create the active collection. The fields associated with this component are:

Table 3.37. Active Collection Factory Fields

Field	Description
@class	The Java class that provides the factory implementation and is directly or indirectly derived from **`org.overlord.rtgov.active.collection.ActiveCollectionFactory`**.

Factory implementation include Infinispan. The fields associated with the **`org.overlord.rtgov.active.collection.infinispan.InfinispanActiveCollectionFactory`** component are:

Table 3.38. Infinispan Fields

Field	Description
cache	The name of the cache to be presented as an Active Map.
container	The optional JNDI name used to obtain the cache container. If not defined, then the default container is obtained from the infinispan.container property from **`overlord-rtgov.properties`** file in the **`$JBOSS_HOME/standalone/configuration`** folder. If the default container is not defined, then a default cache manager is instantiated.

3.6.1.2. Registering the Active Collection Source

JEE Container

The Active Collection Source is deployed within the JEE container as a WAR file with the following structure:

```
warfile
|
|-META-INF
|    |- beans.xml
|
|-WEB-INF
|    |-classes
|    |    |-acs.json
|    |    |-<custom classes/resources>
|    |
|    |-lib
|       |-acs-loader-jee.jar
|       |-<additional libraries>
```

The **`acs.json`** file contains the JSON representation of the Active Collection Source configuration. The **`acs-loader-jee.jar`** acts as a bootstrapper to load and register the Active Collection Source.

If custom active collection source or active change listeners are defined, then the associated classes and resources can be defined in the **`WEB-INF/classes`** folder or within additional libraries located in the **`WEB-INF/lib`** folder. A **`maven pom.xml`** that creates this structure is:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
```

```
        <modelVersion>4.0.0</modelVersion>
        <groupId>....</groupId>
        <artifactId>....</artifactId>
        <version>....</version>
        <packaging>war</packaging>
        <name>....</name>

        <properties>
                <rtgov.version>....</rtgov.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.overlord.rtgov.active-
queries</groupId>
                        <artifactId>active-collection</artifactId>
                        <version>${rtgov.version}</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.overlord.rtgov.active-
queries</groupId>
                        <artifactId>acs-loader-jee</artifactId>
                        <version>${rtgov.version}</version>
                </dependency>
                ....
        </dependencies>

</project>
```

If deploying in JBoss Application Server, then include the following fragment to define the dependency on the core Runtime Governance modules:

```
.....
        <build>
                <finalName>....</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-war-
plugin</artifactId>
                                <configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>
                                        <archive>
                                                <manifestEntries>

<Dependencies>deployment.overlord-rtgov.war</Dependencies>
                                                </manifestEntries>
                                        </archive>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
        .....
```

3.6.2. Presenting Results from an Event Processor Network

An Active Collection Source can be configured to obtain information from an Event Processor Network, which is then placed in the associated Active Collection. Here is an example of an Active Collection Source configured using the **`org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource`** implementation:

```
[[
  {
    "@class" :
"org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "Situations",
    "type" : "List",
    "itemExpiration" : 40000,
    "maxItems" : 0,
    "subject" : "Situations",
    "activeChangeListeners" : [ {
        "@class" : "org.overlord.rtgov.active.collection.jmx.JMXNotifier",
        "objectName" : "overlord.rtgov.services:name=Situations",
        "descriptionScript" : "SituationDescription.mvel",
        "insertTypeScript" : "SituationType.mvel"
    } ],
    "derived": [ {
        "name": "FilteredSituations",
        "predicate": {
            "type": "MVEL",
            "expression": "map =
context.getMap(\"IgnoredSituationSubjects\"); if (map == null) { return
false; } return !map.containsKey(subject);"
        },
        "properties" : {
            "active" : false
        }
    } ]
  }
]
```

The additional fields associated with this implementation are:

Table 3.39. Active Collection Source Fields for EPN

Field	Description
subject	The EPN subject upon which the information has been published.

Here is an example Event Processor Network configuration that publishes information on the subject (such as,Situations) specified in the Active Collection Source configuration above:

```
{
  "name" : "SLAMonitorEPN",
  "subscriptions" : [ {
    "nodeName" : "SLAViolations",
```

```
    "subject" : "ServiceResponseTimes"
  } ],
  "nodes" : [
    {
      "name" : "SLAViolations",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "eventProcessor" : {
        "@class" : "org.overlord.rtgov.ep.drools.DroolsEventProcessor",
        "ruleName" : "SLAViolation"
      },
      "predicate" : null,
      "notifications" : [ {
         "type" : "Processed",
         "subject" : "SituationsProcessed"
      },{
         "type" : "Results",
         "subject" : "Situations"
      } ]
    }
  ],
  "version" : "1"
}
```

3.6.3. Publishing Active Collection Contents as JMX Notifications

You can use JMXNotifier active change listener implementation for Publishing Active Collection Contents. Here is an example of this configuration:

```
[
  .....
 {
   .....
   "activeChangeListeners" : [ {
     "@class" : "org.overlord.rtgov.active.collection.jmx.JMXNotifier",
     "objectName" : "overlord.sample.slamonitor:name=SLAViolations",
     "insertType" : "SLAViolation"
   } ],
   .....
 }
]
```

This implementation has the following additional fields:

Table 3.40. JMXNotifier Listener Implementation Fields

Field	Description
objectName	The MBean (JMX) object name to be used to report the notification.

Field	Description
descriptionScript	The MVEL script that can be used to derive the description field on the notification. If not defined, then the information's `toString()` value is used.
insertType	The type field for the notification when performing an insert.
insertTypeScript	An optional MVEL script that can be used to derive the type field for an insert.
updateType	The optional type field for the notification when performing an update.
updateTypeScript	An optional MVEL script that can be used to derive the type field for an update.
removeType	The optional type field for the notification when performing a removal.
removeTypeScript	An optional MVEL script that can be used to derive the type field for a removal.

3.6.4. Querying Active Collections via REST

The Active Collections configured within the Runtime Governance server can be accessed via a REST service, by POSTing the JSON representation of a query specification to the following URL:

```
<host>/overlord-rtgov/acm/query
```

This service uses basic authentication, by default, using a valid JBoss EAP Application Realm user (you can use the Governance user you configured when you installed Fuse 6).The query Specification comprises the following information:

Table 3.41. Attributes for Active Collections Query

Attribute	Description
collection	The active collection name.
predicate	Optional. If defined with the parent name, then can be used to derive a child collection that filters its parent's content (and notifications) based on the predicate.
parent	Optional. If deriving a child collection, this field defines the parent active collection from which it will be derived.

Attribute	Description
maxItems	Defines the maximum number of items that should be returned in the result, or 0 if unrestricted.
truncate	If a maximum number of items is specified, then this field can be used to indicate whether the Start or End of the collection should be truncated.
style	Allows control over how the results are returned. The value Normal means as it appears in the collection. The value Reversed means the order of the contents should be reversed.

The collection field defines the name of the collection. It can either be an existing collection name, or if defining the predicate and parent fields, then this field defines the name of the derived collection to be created. The predicate field refers to a component that implements a predicate interface. The implementation is defined based on the type field. Currently only a MVEL based implementation exists, with a single field expression defining the predicate as a string. Here is an example:

```
{
    "parent" : "ServiceResponseTime",
    "maxItems" : 5000,
    "collection" : "OrderService",
    "predicate" : {
        "type" : "MVEL",
        "expression" : "serviceType == \"{urn:switchyard-quickstart-
demo:orders:0.1.0}OrderService\" && operation == \"submitOrder\""
    },
    "truncate" : "End",
    "style" : "Reversed"
}
```

If the Active Collection Manager (ACM) does not have a collection named OrderService , then it uses the supplied defaults to create the derived collection. If the collection already exists, then the contents will be returned, allowing multiple users to share the same collection.

3.6.5. Pre-Defined Active Collections

3.6.5.1. ServiceResponseTimes Active Collection

The response times represent an aggregation of the metrics for a particular service, operation and response or fault, over a configured period. For more details please see the API documentation.

3.6.5.2. Situations Active Collection

This active collection is a list of `org.overlord.rtgov.analytics.situation.Situation` objects.

This active collection configuration also publishes it contents via a JMX notifier, based on the following configuration details:

```
[
  {
    ........
  },{
    "@class" :
"org.overlord.rtgov.active.collection.epn.EPNActiveCollectionSource",
    "name" : "Situations",
    "type" : "List",
    "itemExpiration" : 40000,
    "maxItems" : 0,
    "subject" : "Situations",
    "activeChangeListeners" : [ {
        "@class" : "org.overlord.rtgov.active.collection.jmx.JMXNotifier",
        "objectName" : "overlord.rtgov:name=Situations",
        "descriptionScript" : "SituationDescription.mvel",
        "insertTypeScript" : "SituationType.mvel"
    } ],
    .......
  }
]
```

3.6.5.3. ServiceDefinitions Active Collection

This active collection is a map of Service Type name to **`org.overlord.rtgov.analytics.service.ServiceDefinition`** objects. For more details on this class, see the API documentation.

An example of a service definition, represented in JSON is:

```
{
    "serviceType":"{http://www.jboss.org/examples}OrderService",
    "operations":[{
        "name":"buy",
        "metrics":{
            "count":30,
            "average":1666,
            "min":500,
            "max":2500
        },
        "requestResponse":{
            "metrics":{
                "count":10,
                "average":1000,
                "min":500,
                "max":1500
            },
            "invocations":[{
                "serviceType":"
{http://www.jboss.org/examples}CreditAgencyService",
                "metrics":{
                    "count":10,
                    "average":500,
                    "min":250,
                    "max":750
                },
```

```
                "operation":"checkCredit"
            }]
        },
        "requestFaults":[{
            "fault":"UnknownCustomer",
            "metrics":{
                "count":20,
                "average":2000,
                "min":1500,
                "max":2500
             }
        }]
    }],
    "metrics":{
        "count":30,
        "average":1666,
        "min":500,
        "max":2500
    }
}
```


The list of service definitions returned from this active collection, and the information they represent (such as consumed services), represents a near term view of the service activity based on the configuration details defined in the collection's active collection source. Therefore, if (for example) a service has not invoked one of its consumed services within the time period of interest, then its details do not show in the service definition. This information is intended to show the service activity that has occurred in the recent history, as a means of monitoring the real-time situation to deal with emerging problems. The duration over which the information is retained is determined by the following two properties in the ServiceDefinitions active collection source configuration:

- scheduledInterval (in milliseconds): It dictates how often a snapshot of the current service definition information is stored.
- maxSnapshots: It defines the maximum number of snapshots that must be used.

So the duration of information retained can be calculated as the scheduled interval multiplied by the maximum number of snapshots.

3.6.5.4. Principals Active Collection

This active collection is a map of Principal name to a map of named properties. This information is used to convey details captured (or derived) regarding a principal . A principal can represent a user, group or organization.

3.6.6. Implementing an Active Collection Source

The Active Collection Source can be considered as an adapter between the actual source of events and the Active Collection. The Active Collection Source is responsible for managing the insertion, update and deletion of the objects within the associated Active Collection, based on situations that occur in the source. An example of a derived Active Collection Source implementation, that is packaged with the infrastructure, can be used to listen for events produced by nodes in an Event Processor Network and insert these events in the Active Collection. To create a new type of Active Collection Source, derive a class from the **`org.overlord.rtgov.active.collection.ActiveCollectionSource`** class and implement the following methods:

Table 3.42. ActiveCollectionSource Methods for New Active Collection Source

Method	Description
`void init()`	This method is invoked when the Active Collection Source is registered, and should be used to create the subscription to the relevant source of information. The implementation of this method MUST call the `init()` method on the super class first.
`void close()`	This method is invoked when the Active Collection Source is unregistered, and should be used to unsubscribe from the source of information. The implementation of this method MUST call the `close()` method on the super class first.

When a situation occurs on the source, that requires a change in the associated Active Collection, then the derived implementation can call one of the following methods on the Active Collection Source:

Table 3.43. ActiveCollectionSource Methods

Method	Description
`public void insert(Object key, Object value)`	This method is called to insert a new element into the collection. The value is the information to be inserted. The key is potentially optional, depending on the nature of the active collection (List or Map). For a List the key is optional. If specified, then it MUST be an integer representing the index where the value must be inserted. The Map key represents the map key to be associated with the value, and is therefore not optional.
`public void update(Object key, Object value)`	This method is called to update an existing element within the collection. The value is the information to be updated. The key is potentially optional, depending on the nature of the active collection (List or Map). For a List the key is optional. If specified, then it MUST be an integer representing the index of the value to be updated. If not specified, then the value will be used to locate the index within the list. The Map key represents the map key associated with the value, and is therefore not optional.

Method	Description
`public void remove(Object key, Object value)`	This method is called to remove an element from the collection. The value is the information to be updated. The key is potentially optional, depending on the nature of the active collection (List or Map). For a List the key is optional. If specified, then it MUST be an integer representing the index of the value to be removed. If not specified, then the value will be used to locate the index within the list. The Map key represents the map key associated with the value, and is therefore not optional. However in this situation, the value is optional.

3.6.7. Implement an Active Change Listener

You can implement a listener to deal with changes that occur within an Active Collection. Here is how you perform general implementations of this interface, which may be used within custom applications:

Any component that is interested in being informed when a change occurs to an associated Active Collection can implement the `org.overlord.rtgov.active.collection.ActiveChangeListener` interface. The Active Collection API supports add and remove methods to register and unregister these active change listeners. The methods that need to be implemented for an active change listener are:

Table 3.44. Active Change Listener Methods

Method	Description
`void inserted(Object key, Object value)`	This method is called when a new value is inserted into the collection, with the key being dependent upon the type of collection (List or Map). The optional List key is the index and the mandatory Map key. The List key is the index and the Map key is the key information used in the map's key/value pair.
`void updated(Object key, Object value)`	This method is called when an existing value is updated within the collection, with the key being dependent upon the type of collection (List or Map). The optional List key is the index and the mandatory Map key. The List key is the index and the Map key is the key information used in the map's key/value pair.
`void removed(Object key, Object value)`	This method is called when an existing value is removed from the collection, with the key being dependent upon the type of collection (List or Map). The optional List key is the index and the mandatory Map key. The List key is the index, and the Map key is the key information used in the map's key/value pair.

Here is how you deal with a specific type of listener that can be configured with an Active Change Source, and automatically initialized when the Active Change Source is registered:

If the active change listener implementation is derived from the **`org.overlord.rtgov.active.collection.AbstractActiveChangeListener`** abstract class, then it can be registered with the Active Collection Source configuration, and automatically initialized when the source is registered. The benefit of this approach is that it does not require the user to write custom code to register the Active Collection Listener against the Active Collection. An example of this type of implementation is the **`org.overlord.rtgov.active.collection.jmx.JMXNotifier`**, which automatically generates JMX notifications when an object is added to the associated active collection. The implementations derived from this abstract active change listener implementation are no different from other active change listener implementations, with the exception that they can be serialized as part of the Active Collection Source configuration. Also, they support the following lifecycle methods for initialization and closing:

Table 3.45. Abstract Implementation Methods

Method	Description
`void init()`	This method can be overridden to initialize the active change listener implementation. The super class init() method MUST be called first.
`void close()`	This method can be overridden to close the active change listener implementation. The super class close() method MUST be called first.

3.6.8. Accessing Active Collections

3.6.8.1. Retrieving an Active Collection

There are two ways to retrieve an active collection:

- Directly accessing the ActiveCollectionManager

  Active Collections are created as a bi-product of registering an Active Collection Source. The Active Collection Source is registered with an Active Collection Manager, which creates the collection to be updated from the source. This Active Collection then becomes available for applications to retrieve from the manager. here is an example:

```
import org.overlord.rtgov.active.collection.ActiveCollectionManager;
import
org.overlord.rtgov.active.collection.ActiveCollectionManagerAccessor
;
import org.overlord.rtgov.active.collection.ActiveList;

.....

ActiveCollectionManager
acmManager=ActiveCollectionManagerAccessor.getActiveCollectionManage
r();
```

```
ActiveList list = (ActiveList)
          acmManager.getActiveCollection(listName);
```

This approach is used to retrieve the top level active collections. These are the collections directly maintained by the Active Collection Manager, each with an associated Active Collection Source defining the origin of the collection changes. The maven dependency required to access the ActiveCollectionManager and active collections is:

```
  <dependency>
              <groupId>org.overlord.rtgov.active-queries</groupId>
              <artifactId>active-collection</artifactId>
              <version>${rtgov.version}</version>
              <scope>provided</scope>
  </dependency>
```

- Using Injectable Collection Manager

  Using Injectable Collection Manager approach is aimed at simplifying the use of active collections from within a client application. It offers a simple API, and associated default implementation, that can be injected using CDI. Here is an example:

```
    @Inject
    private org.overlord.rtgov.client.CollectionManager
_collectionManager=null;

    private org.overlord.rtgov.active.collection.ActiveMap
_principals=null;

    protected void init() {

        if (_collectionManager != null) {
            _principals = _collectionManager.getMap(PRINCIPALS);
        }

        .......
    }
```

If injection is not possible (for example, when using SwitchYard Interceptors), then a default implementation can be directly instantiated with the class **`org.overlord.rtgov.client.DefaultCollectionManager`**. The maven dependencies required to access the CollectionManager, and the subsequent active collections are:

```
                <dependency>

<groupId>org.overlord.rtgov.integration</groupId>
                        <artifactId>rtgov-client</artifactId>
                        <version>${rtgov.version}</version>
                </dependency>
                <dependency>
                        <groupId>org.overlord.rtgov.active-
queries</groupId>
                        <artifactId>active-collection</artifactId>
```

```
                        <version>${rtgov.version}</version>
                        <scope>provided</scope>
                    </dependency>
```

3.6.8.2. Creating a Derived Active Collection

The top level active collections reflect the information changes as identified by their associated Active Collection Source. However in some situations, only a subset of the information is of interest to an application. For these situations, it is possible to derive a child active collection by specifying the following:

- parent: The parent collection from which the child may be derived. Although this is generally the name of a top level collection, it is possible to derive a collection from another child collection, forming a tree.
- predicate: A predicate is specified to determine whether information in a parent collection (and subsequently its changes), are relevant to the child collection or not.
- properties: Properties are used to initialize the derived collection.

Currently the only property that can be set is a boolean named active, which defaults to true.

If the active property is true, then when a child collection is initially created, the predicate is used to filter the contents of the parent collection. The predicate identifies the initial subset of values that are relevant for the child collection. Once initialized, the child collection effectively subscribes to the change notifications of the parent collection, and uses the predicate to determine whether the change is applicable, and if so, applies the change to the child collection.

If the active property is false, then whenever the derived collection is queried, the predicate is applied to the parent collection to obtain the current set of results. Use this configuration only where the predicate is based on volatile information. The results in the derived collection changes independently of changes applied to the parent collection. Here is an example:

```
import org.overlord.rtgov.active.collection.predicate.Predicate;
import org.overlord.rtgov.active.collection.ActiveCollectionManager;
import org.overlord.rtgov.active.collection.ActiveList;

.....

Predicate predicate=.....;

ActiveList parent =
(ActiveList)acmManager.getActiveCollection(parentName);

if (parent != null) {
        java.util.Map<String,Object> properties=.....;

        alist = (ActiveList)acmManager.create(childName,
                    parent, predicate, properties);
}}
```

3.6.8.3. Registering for Active Change Notifications

Once an Active Collection has been retrieved (or created in the case of a child collection), you can access the information using methods appropriate to the collection type, such as List or Map. In active

collections, an important source of information is the change notifications. They enable the application to understand when and what changes are occurring. To receive change notifications, the application needs to register an Active Change Listener. You can achieve this using the **`addActiveChangeListener`** method on the collection, and similarly use the **`removeActiveChangeListener`** method to unregister for change notifications. Here is an example:

```
import org.overlord.rtgov.active.collection.ActiveList;
import org.overlord.rtgov.active.collection.ActiveChangeListener;

.....

ActiveList list=.....;

list.addActiveChangeListener(new ActiveChangeListener() {
    public void inserted(Object key, Object value) {
        ....
    }
    public void updated(Object key, Object value) {
        ....
    }
    public void removed(Object key, Object value) {
        ....
    }
});
```

3.7. REPORT SERVER

The Report Server service is used to generate instances of a report whose definition has previously been deployed to the server. This section explains how to configure and deploy a report definition, and then how to generate the report instances.

3.7.1. Creating and Deploying a Report Definition

1. **Specify a JSON representation of the `org.overlord.rtgov.reports.ReportDefinition` class**
   The report definition only contains the name of the report, and the definition of the generator. Here is an example:

   ```
   [  {
       "name" : "SLAReport",
       "generator" : {
         "@class" : "org.overlord.rtgov.reports.MVELReportGenerator",
          "scriptLocation" : "SLAReport.mvel"
       }
     }
   ]
   ```

   In this case, the **`org.overlord.rtgov.reports.MVELReportGenerator`** implementation of the report generator is used, which also includes a property to define the location of the report script (for example, SLAReport.mvel). This MVEL SLA report script is located at **`samples/sla/report`** folder.

   For detals on **`org.overlord.rtgov.reports.ReportDefinition`** class, see API documentation.

2. **Register the Report**
   The Report Definition is deployed within the JEE container as a WAR file with the following structure:

```
warfile
|
|-META-INF
|    |- beans.xml
|
|-WEB-INF
|    |-classes
|    |    |-reports.json
|    |    |-<custom classes/resources>
|    |
|    |-lib
|       |-reports-loader-jee.jar
|       |-<additional libraries>
```

   The **`reports.json`** file contains the JSON representation of the report definition configuration. The **`reports-loader-jee.jar`** acts as a bootstrapper to load and register the Report Definition. If custom report generators or scripts are defined, then the associated classes and resources can be defined in the **`WEB-INF/classes`** folder or within additional libraries located in the **`WEB-INF/lib`** folder. Here is an example of the maven **`pom.xml`** that creates this structure is:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>....</groupId>
        <artifactId>....</artifactId>
        <version>....</version>
        <packaging>war</packaging>
        <name>....</name>

        <properties>
                <rtgov.version>....</rtgov.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.overlord.rtgov.activity-
analysis</groupId>
                        <artifactId>reports-loader-jee</artifactId>
                        <version>${project.version}</version>
                </dependency>
                <dependency>
                        <groupId>org.overlord.rtgov.activity-
analysis</groupId>
                        <artifactId>reports</artifactId>
                        <version>${project.version}</version>
                        <scope>test</scope>
                </dependency>
                ....
```

```
        </dependencies>

</project>
```

If deploying in JBoss Application Server, then include the following fragment to define the dependency on the core Runtime Governance modules:

```
.....
        <build>
                <finalName>slamonitor-epn</finalName>
                <plugins>
                        <plugin>
                                <artifactId>maven-war-
plugin</artifactId>
                                <configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>
                                        <archive>
                                                <manifestEntries>

<Dependencies>deployment.overlord-rtgov.war</Dependencies>

</manifestEntries>
                                        </archive>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
        .....
```

3.7.2. Generating an Instance of the Report

- **Use the following URL for the service's REST GET request:**

```
<host>/overlord-rtgov/report/generate?<parameters>
```

The service uses basic authentication by default, using a valid JBoss EAP Application Realm user (you can use the Governance user you configured when you installed Fuse 6). This service has the following query parameters:

Table 3.46. Query Parameters

Parameter	Description
report	The name of the report to be generated. This must match the previously deployed report definition name.
startDay/Month/Year	The optional start date for the report. If not defined, then the report will use all activities stored up until the end date.

Parameter	Description
endDay/Month/Year	The optional end date for the report. If not defined, then the report will use all activities up until the current date.
timezone	The optional timezone.
calendar	The optional business calendar name. A default called exists called Default which represents a working week of Monday to Friday, 9am to 5pm, excluding Christmas Day.

The query parameters tare specific to the report definition being generated. The operation returns a JSON representation of the `org.overlord.rtgov.reports.model.Report` class. See the API documentation for more details on the object model.

3.7.3. Providing a Custom Business Calendar

1. **Define a Business Calendar**
   A custom Business Calendar can be defined as a JSON representation of the `org.overlord.rtgov.reports.mode.Calendar` class. See API documentation for details.

2. **Store it in the properties file**
   Store the calender in a file whose location is referenced using a property called calendar.<CalendarName> in the `overlord-rtgov.properties` file.

3.8. SITUATION MANAGER

The Situation Manager service is used to determine whether situations associated with a particular service must be displayed to users via the Situations gadget or not. The service supports two operations:

- ignore operation
- observe operation

The service uses basic authentication by default, using a valid JBoss EAP Application Realm user (you can use the Governance user you configured when you installed Fuse 6).

3.8.1. Ignoring Situations Related to a Service

The ignore operation is used to indicate that situations for a particular service type must not be presented to users via the REST service, and therefore the Situations gadget.

3.8.2. Observing Situations Related to a Service

The observe operation is used to reverse the actions performed by a previous ignore operation. This makes the situations for a particular a service type visible again to users via the REST service and therefore the Situations gadget.

3.9. POLICY ENFORCEMENT

This section provides policy enforcement features that are distributed with the Runtime Governance project. The example located in the **samples/policy** folder, demonstrates two approaches that can be used to provide "policy enforcement". This example makes uses of the example in Switchyard application, located in the **samples/ordermgmt** folder.

3.9.1. Synchronous Enforcement

The Synchronous Enforcement approach shows how a business policy can be implemented in a synchronous (or inline) manner, where the decision is taken immediate, and can therefore be used to influence the current business transaction. The benefit of this approach is that it can ensure only valid transactions are permitted, as decisions can be immediately enforced, however the disadvantage is the potential performance impact this may have on the business transaction. This example shows activity event analysis, using the Activity Validator mechanism, to implement the business policy.

3.9.1.1. The Policy

The runtime governance infrastructure analyses the activity events generated by an executing business transaction using one or more Activity Validators. By default, Activity Validators are not invoked from within the SwitchYard environment. The specific SwitchYard applications need to be configured to include an auditor that will invoke the validation. In the Order Management quickstart, this is achieved using the class **org.overlord.rtgov.quickstarts.demos.orders.interceptors.ExchangeValidator**. This class is derived from an abstract base class that provides most of the required functionality for converting an Exchange message into an activity event. For example:

```
@Audit({Processors.TRANSFORMATION})
@Named("ExchangeValidator")
public class ExchangeValidator extends AbstractExchangeValidator
implements Auditor {

    /**
     * {@inheritDoc}
     */
    public void afterCall(Processors processor, org.apache.camel.Exchange
exch) {

        ExchangePhase
phase=exch.getProperty("org.switchyard.bus.camel.phase",
ExchangePhase.class);

        if (phase == ExchangePhase.OUT) {
            handleExchange(exch, phase);
        }
    }

    /**
     * {@inheritDoc}
     */
    public void beforeCall(Processors processor, org.apache.camel.Exchange
exch) {

        ExchangePhase
phase=exch.getProperty("org.switchyard.bus.camel.phase",
```

```
ExchangePhase.class);

        if (phase == ExchangePhase.IN) {
            handleExchange(exch, phase);
        }
    }
}
```

The following Activity Validator configuration is deployed in the environment responsible for executing the business transaction, and gets registered with the Activity Collector mechanism:

```
[{
  "name" : "RestrictUsage",
  "version" : "1",
  "predicate" : {
    "@class" : "org.overlord.rtgov.ep.mvel.MVELPredicate",
    "expression" : "event instanceof
org.overlord.rtgov.activity.model.soa.RequestReceived && event.serviceType
== \"{urn:switchyard-quickstart-demo:orders:0.1.0}OrderService\""
  },
  "eventProcessor" : {
    "@class" : "org.overlord.rtgov.ep.mvel.MVELEventProcessor",
    "script" : "VerifyLastUsage.mvel",
    "services" : {
      "CacheManager" : {
        "@class" :
"org.overlord.rtgov.common.infinispan.service.InfinispanCacheManager"
      }
    }
  }
}]
```

This Activity Validator receives activity events generated from the executing environment and applies the optional predicate to determine if they are of interest to the defined event processor. In this case, the predicate is checking for received requests for the OrderService service.

For events that pass this predicate, they are submitted to the business policy, defined using the MVEL script **`VerifyLastUsage.mvel`**, which is:

```
String customer=event.properties.get("customer");

if (customer == null) {
        return;
}

cm = epc.getService("CacheManager");

// Attempt to lock the entry
if (!cm.lock("Principals", customer)) {
        epc.handle(new java.lang.RuntimeException("Unable to lock entry
for principal '"+customer+"'"));

        return;
}

// Access the cache of principals
```

```
principals = cm.getCache("Principals");

principal = principals.get(customer);

if (principal == null) {
        principal = new java.util.HashMap();
}

java.util.Date current=principal.get(event.serviceType+"-lastaccess");
java.util.Date now=new java.util.Date();

if (current != null && (now.getTime()-current.getTime()) < 2000) {
        epc.handle(new java.lang.RuntimeException("Customer '"+customer+"'
cannot perform more than one request every 2 seconds"));

        return;
}

principal.put(event.serviceType+"-lastaccess", now);
principals.put(customer, principal);

epc.logDebug("Updated principal '"+customer+"':
"+principals.get(customer));
```

This script uses the CacheManager service, configured within the EventProcessor component, to obtain a cache called "Principals". This cache is used to store information about Principals as a map of properties. The implementation uses Infinispan, to enable the cache to be shared between other applications, as well as in a distributed/cluster environment (based on the infinispan configuration).

If a policy violation is detected, the script returns an exception using the **`handle()`** method on the EventProcessor context. This results in the exception being thrown back to the execution environment, interrupting the execution of the business transaction.

3.9.1.2. Quickstart Example

The quickstart example in this section demonstrates how a policy enforcement mechanism can be provided using the Activity Validator mechanism, to immediately evaluate the business policy and (if appropriate) block the business transaction.

3.9.1.3. Installing the Example

1. Start the Switchyard server using the following command from the bin folder:

   ```
   ./standalone.sh -c standalone-full.xml
   ```

2. Install the example Switchyard application by running the following command from the **`${rtgov}/samples/ordermgmt`** folder:

   ```
   mvn jboss-as:deploy
   ```

3. Change to the **`${rtgov}/samples/policy/sync`** folder and run the following command again:

   ```
   mvn jboss-as:deploy
   ```

3.9.1.4. Running the Example

To demonstrate the synchronous policy, send the following message twice in less than 2 seconds, to the example Switchyard application at http://localhost:8080/demo-orders/OrderService:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <orders:submitOrder xmlns:orders="urn:switchyard-quickstart-
demo:orders:1.0">
            <order>
                <orderId>1</orderId>
                <itemId>BUTTER</itemId>
                <quantity>100</quantity>
                <customer>Fred</customer>
            </order>
        </orders:submitOrder>
    </soap:Body>
</soap:Envelope>
```

The messages can be sent using an appropriate SOAP client (e.g. SOAP-UI) or by running the test client available with the Switchyard application, by running the following command from the ${rtgov}/samples/ordermgmt/app folder:

```
mvn exec:java -Dreq=order1 -Dcount=2
```

Result

If the two requests are received within two seconds of each other, it results in the following response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>org.switchyard.exception.SwitchYardException:
Customer 'Fred' cannot perform more than one request every 2
seconds</faultstring>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

3.9.2. Asynchronous Enforcement

The Asynchronous Enforcement approach shows how a business policy can be implemented in an asynchronous (or out-of-band) manner, where the decision is taken after the fact, and can therefore only be used to influence future business transactions. The benefit of this approach is that the decision making process does not have to occur immediately and therefore avoids potentially impacting the performance of the business transaction. The disadvantage is that it does not permit any decision that is made to be enforced immediately. This example shows:

- Activity event analysis, using the Event Processor Network mechanism, can be used to implement business policies
- Results from the business policies can be cached for reference by other applications

- Platform specific interceptors can reference the results to impact the behavior of the business transaction (for example, prevent suspended customers purchasing further items)

3.9.2.1. The Policy

There are three components that comprise the policy within this example.

Event Analysis

The runtime governance infrastructure analyses the activity events generated by an executing business transaction using one or more Event Processor Networks (or EPN). A standard EPN is deployed within the infrastructure to isolate the SOA events (e.g. request/responses being sent or received). This quickstart deploys another EPN that subscribes to the events produced by the standard EPN:

```
{
  "name" : "AssessCreditPolicyEPN",
  "version" : "1",
  "subscriptions" : [ {
    "nodeName" : "AssessCredit",
    "subject" : "SOAEvents"
  } ],
  "nodes" : [
    {
      "name" : "AssessCredit",
      "sourceNodes" : [ ],
      "destinationSubjects" : [ ],
      "maxRetries" : 3,
      "retryInterval" : 0,
      "predicate" : {
        "@class" : "org.overlord.rtgov.ep.mvel.MVELPredicate",
        "expression" : "event.serviceProvider && !event.request &&
event.serviceType == \"{urn:switchyard-quickstart-
demo:orders:0.1.0}OrderService\""
      },
      "eventProcessor" : {
        "@class" : "org.overlord.rtgov.ep.mvel.MVELEventProcessor",
        "script" : "AssessCredit.mvel",
        "services" : {
          "CacheManager" : {
            "@class" :
"org.overlord.rtgov.common.infinispan.service.InfinispanCacheManager"
          }
        }
      }
    }
  ]
}
```

This EPN subscribes to the published SOA events and applies the predicate which ensures that only events from a service provider interface, that are responses and are associated with the OrderService service, will be processed. Events that pass this predicate are then submitted to the business policy (defined in the MVEL script AssessCredit.mvel), which is:

```
String customer=event.properties.get("customer");
```

```
if (customer == null) {
        return;
}

cm = epc.getService("CacheManager");

// Attempt to lock the entry
if (!cm.lock("Principals", customer)) {
        epc.handle(new Exception("Unable to lock entry for principal
'"+customer+"'"));

        return;
}

// Access the cache of principals
principals = cm.getCache("Principals");

principal = principals.get(customer);

if (principal == null) {
        principal = new java.util.HashMap();
}

int current=principal.get("exposure");

if (current == null) {
        current = 0;
}

if (event.operation == "submitOrder") {

        double total=event.properties.get("total");

        int newtotal=current+total;

        if (newtotal > 150 && current <= 150) {
                principal.put("suspended", Boolean.TRUE);
        }

        principal.put("exposure", newtotal);

} else if (event.operation == "makePayment") {

        double amount=event.properties.get("amount");

        int newamount=current-amount;

        if (newamount <= 150 && current > 150) {
                principal.put("suspended", Boolean.FALSE);
        }

        principal.put("exposure", newamount);
}

principals.put(customer, principal);
```

```
epc.logDebug("Updated principal '"+customer+"':
"+principals.get(customer));
```

This script uses the CacheManager service, configured within the EPN node, to obtain a cache called "Principals". This cache is used to store information about Principals as a map of properties. The implementation uses Infinispan, to enable the cache to be shared between other applications, as well as in a distributed/cluster environment (based on the infinispan configuration).

Result Management

The results derived from the previous policy are stored in an Infinispan implemented cache called "Principals". To make this information available to runtime governance clients, we use the Active Collection mechanism - more specifically we define an Active Collection, as part of the standard installation, that wraps the Infinispan cache. The configuration of the Active Collection Source is:

```
[
  {
    ......
  },{
    "@class" :
"org.overlord.rtgov.active.collection.ActiveCollectionSource",
    "name" : "Principals",
    "type" : "Map",
    "lazy" : true,
    "visibility" : "Private",
    "factory" : {
        "@class" :
"org.overlord.rtgov.active.collection.infinispan.InfinispanActiveCollectio
nFactory",
        "cache" : "Principals"
    }
  }
]
```

The visibility is marked as private to ensure that exposure information regarding customers is not publicly available via the Active Collection REST API.

The Enforcer

The enforcement is provided by a specific Switchyard Auditor implementation (PolicyEnforcer) that is included with the order management application. The main part of this auditor is:

```
public void beforeCall(Processors processor, org.apache.camel.Exchange
exch) {
        ....

        if (_principals != null) {
            org.switchyard.bus.camel.CamelMessage mesg=
(org.switchyard.bus.camel.CamelMessage)exch.getIn();

            if (mesg == null) {
                LOG.severe("Could not obtain message for phase
("+phase+") and exchange: "+exch);
                return;
            }
```

```
            org.switchyard.Context context=new
org.switchyard.bus.camel.CamelCompositeContext(exch, mesg);

            java.util.Set<Property> contextProps=context.getProperties(
                    org.switchyard.Scope.MESSAGE);

            Property p=null;

            for (Property prop : contextProps) {
                if (prop.getName().equals("org.switchyard.contentType"))
{
                    p = prop;
                    break;
                }
            }

            if (p != null && p.getValue().toString().equals(
                            "{urn:switchyard-quickstart-
demo:orders:1.0}submitOrder")) {

                String customer=getCustomer(mesg);

                if (customer != null) {
                    if (_principals.containsKey(customer)) {

                        @SuppressWarnings("unchecked")
                        java.util.Map<String,java.io.Serializable> props=

(java.util.Map<String,java.io.Serializable>)
                                            _principals.get(customer);

                        // Check if customer is suspended
                        if (props.containsKey("suspended")
                                &&
props.get("suspended").equals(Boolean.TRUE)) {
                            throw new RuntimeException("Customer
'"+customer
                                                +"' has been suspended");
                        }
                    }

                    if (LOG.isLoggable(Level.FINE)) {
                        LOG.fine("*********** Policy Enforcer: customer
'"
                                +customer+"' has not been suspended");
                        LOG.fine("*********** Principal:
"+_principals.get(customer));
                    }
                } else {
                    LOG.warning("Unable to find customer name");
                }
            }
        }
```

The variable _principals refers to an Active Map used to maintain information about the principal (i.e. the customer in this case). This information is updated using the policy rule defined in the previous section..

3.9.2.2. Quickstart Example

The quickstart example in this section demonstrates how a policy enforcement mechanism can be provided using a combination of the Runtime Governance infrastructure and platform specific interceptors. This example uses an asynchronous approach to evaluate the business policies, only enforcing the policy based on a summary result from the decision making process. The benefit of this approach is that it can be more efficient, and reduce the performance impact on the business transaction being policed. The disadvantage is that the decisions are made after the fact, hence leaves a small window of opportunity for invalid transactions to be performed.

3.9.2.3. Installing the Example

1. To install the example, start the Switchyard server using the following command from the bin folder:

```
./standalone.sh -c standalone-full.xml
```

2. Install the example Switchyard application, achieved by running the following command from the ${rtgov}/samples/ordermgmt folder:

```
mvn jboss-as:deploy
```

3. Change to the **${rtgov}/samples/policy/async** folder and run the following command again:

```
mvn jboss-as:deploy
```

3.9.2.4. Running the Example

To demonstrate the asynchronous policy enforcement,send the following message to the example Switchyard application at http://localhost:8080/demo-orders/OrderService:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <orders:submitOrder xmlns:orders="urn:switchyard-quickstart-
demo:orders:1.0">
            <order>
                <orderId>1</orderId>
                <itemId>BUTTER</itemId>
                <quantity>100</quantity>
                <customer>Fred</customer>
            </order>
        </orders:submitOrder>
    </soap:Body>
</soap:Envelope>
```

The message can be sent using an appropriate SOAP client (e.g. SOAP-UI) or by running the test client available with the Switchyard application, by running the following command from the ${rtgov}/samples/ordermgmt/app folder:

```
mvn exec:java -Dreq=order1
```

Result

This results in the following response, indicating that the purchase was successful, as well as identifying the total cost of the purchase (that is 125).

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <orders:submitOrderResponse xmlns:orders="urn:switchyard-quickstart-
demo:orders:1.0">
         <orderAck>
            <orderId>1</orderId>
            <accepted>true</accepted>
            <status>Order Accepted</status>
            <customer>Fred</customer>
            <total>125.0</total>
         </orderAck>
      </orders:submitOrderResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If the customer's debt exceeds the threshold of 150 then the customer would be suspended. Therefore if the same request is issued again, resulting in another total of 125, then the overall exposure regarding this customer is now 250. If we then attempt to issue the same request a third time, this time we will receive a SOAP fault from the server. This is due to the fact that the PolicyEnforcer auditor has intercepted the request, and detected that the customer is now suspended.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
         <faultcode>SOAP-ENV:Server</faultcode>
         <faultstring>Customer 'Fred' has been suspended</faultstring>
      </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If we now send a "makePayment" request as follows to the same URL:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:switchyard-quickstart-demo:orders:1.0">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:makePayment>
         <payment>
            <customer>Fred</customer>
            <amount>200</amount>
```

```
         </payment>
      </urn:makePayment>
   </soapenv:Body>
</soapenv:Envelope>
```

This can be sent using a suitable SOAP client (for example, SOAP-UI) or the test client in the order management application:

```
mvn exec:java -Dreq=fredpay
```

This results in the customer being unsuspended, as it removes 200 from their current exposure (leaving 50). To confirm this, try sending the submitOrder request again.

3.10. SLA

3.10.1. Monitor

The example located in the **`samples/sla/monitor`** folder demonstrates an approach to provide "Service Level Agreement" monitoring. This example makes uses of the example Switchyard application located in the **`samples/ordermgmt`** folder. This example will shows:

- activity event analysis, using the Event Processor Network mechanism, can be used to implement Service Level Agreements
  - uses the Complex Event Processing (CEP) based event processor (using Drools Fusion)
- impending or actual SLA violations can be reported for the attention of end users, via
  - JMX notifications
  - REST service
- to build a custom application to access the analysis results

This example shows a simple Service Level Agreement that checks whether a service response time exceeds expected levels. The CEP rule detects whether a situation of interest has occurred, and if so, creates a**`org.overlord.rtgov.analytics.situation.Situation`** object and initializes it with the appropriate description or severity information, before forwarding it back into the EPN. This results in the "Situation" object being published as a notification on the "Situations" subject.

The CEP rule is:

```
import org.overlord.rtgov.analytics.service.ResponseTime
import org.overlord.rtgov.analytics.situation.Situation

global org.overlord.rtgov.ep.EPContext epc

declare ResponseTime
    @role( event )
end

rule "check for SLA violations"
when
    $rt : ResponseTime() from entry-point "ServiceResponseTimes"
```

```
then

        if ($rt.getAverage() > 200) {
                epc.logError("\r\n\r\n**** RESPONSE TIME
"+$rt.getAverage()+"ms EXCEEDED SLA FOR "+$rt.getServiceType()+"
****\r\n");

                Situation situation=new Situation();

                situation.setType("SLA Violation");

situation.setSubject(Situation.createSubject($rt.getServiceType(),
$rt.getOperation(),
                                                $rt.getFault()));
                situation.setTimestamp(System.currentTimeMillis());

                situation.getProperties().putAll($rt.getProperties());

                if ($rt.getRequestId() != null) {

situation.getActivityTypeIds().add($rt.getRequestId());
                }
                if ($rt.getResponseId() != null) {

situation.getActivityTypeIds().add($rt.getResponseId());
                }

                situation.getContext().addAll($rt.getContext());

                String serviceName=$rt.getServiceType();

                if (serviceName.startsWith("{")) {
                        serviceName =
javax.xml.namespace.QName.valueOf(serviceName).getLocalPart();
                }

                if ($rt.getAverage() > 400) {
                        situation.setDescription(serviceName+" exceeded
maximum response time of 400 ms");

situation.setSeverity(Situation.Severity.Critical);
                } else if ($rt.getAverage() > 320) {
                        situation.setDescription(serviceName+" exceeded
response time of 320 ms");
                        situation.setSeverity(Situation.Severity.High);
                } else if ($rt.getAverage() > 260) {
                        situation.setDescription(serviceName+" exceeded
response time of 260 ms");
                        situation.setSeverity(Situation.Severity.Medium);
                } else {
                        situation.setDescription(serviceName+" exceeded
response time of 200 ms");
                        situation.setSeverity(Situation.Severity.Low);
                }

                epc.handle(situation);
```

```
        }

end
```

The "out of the box" active collection configuration is pre-initialized with a collection for the **`org.overlord.rtgov.analytics.situation.Situation`** objects, subscribing to the "Situations" subject from the Event Processor Network. Therefore any detected SLA violations will automatically be stored in this collection (accessible via a RESTful service), and reported to the associated JMX notifier.

3.10.1.1. Quickstart Example

The quickstart example in this esection demonstrates how Service Level Agreements can be policed using rules defined in an Event Processor Network, and reporting to end users using the pre-configured "Situations" active collection. The rule used in this example is detecting whether the response time associated with an operation on a service exceeds a particular level. However more complex temporal rules could be defined to identify the latency between any two points in a business transaction flow.

3.10.1.2. Installing the Example

1. Start the Switchyard server using the following command from the bin folder:

   ```
   ./standalone.sh -c standalone-full.xml
   ```

2. Install the example Switchyard application by running the following command from the **`${rtgov}/samples/ordermgmt`** folder:

   ```
   mvn jboss-as:deploy
   ```

3. Change to the **`${rtgov}/samples/sla/epn and ${rtgov}/samples/sla/monitor`** folder and run the following command again:

   ```
   mvn jboss-as:deploy
   ```

3.10.1.3. Running the Example

- To demonstrate a Service Level Agreement violation, send the following message to the example Switchyard application at http://localhost:8080/demo-orders/OrderService:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <orders:submitOrder xmlns:orders="urn:switchyard-quickstart-
demo:orders:1.0">
         <order>
            <orderId>3</orderId>
            <itemId>JAM</itemId>
            <quantity>400</quantity>
            <customer>Fred</customer>
         </order>
      </orders:submitOrder>
   </soap:Body>
</soap:Envelope>
```

The message can be sent using an appropriate SOAP client (for example, SOAP-UI) or by running the test client available with the Switchyard application, by running the following command from the **`${rtgov}/samples/ordermgmt/app`** folder:

```
mvn exec:java -Dreq=order3
```

The itemId of "JAM" causes a delay to be introduced in the service, resulting in a SLA violation being detected. This violation can be viewed using the following approaches:

- REST Service
- JMX Console

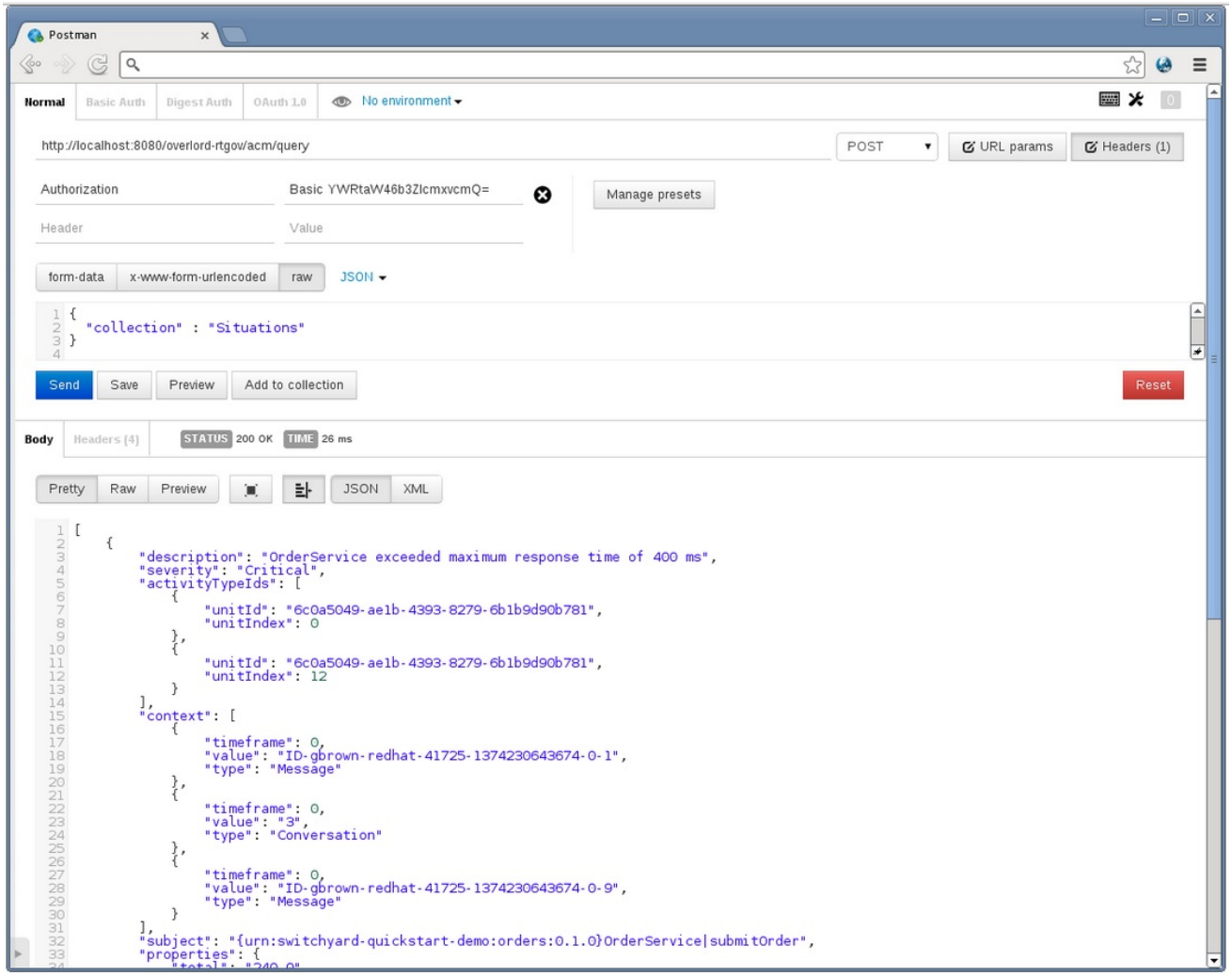
3.10.1.3.1. REST Service

Using a suitable REST client, send the following POST to http://localhost:8080/overlord-rtgov/acm/query (using content-type of "application/json", username as admin, and password as overlord):

```
{
    "collection" : "Situations"
}
```

This will result in the following response:

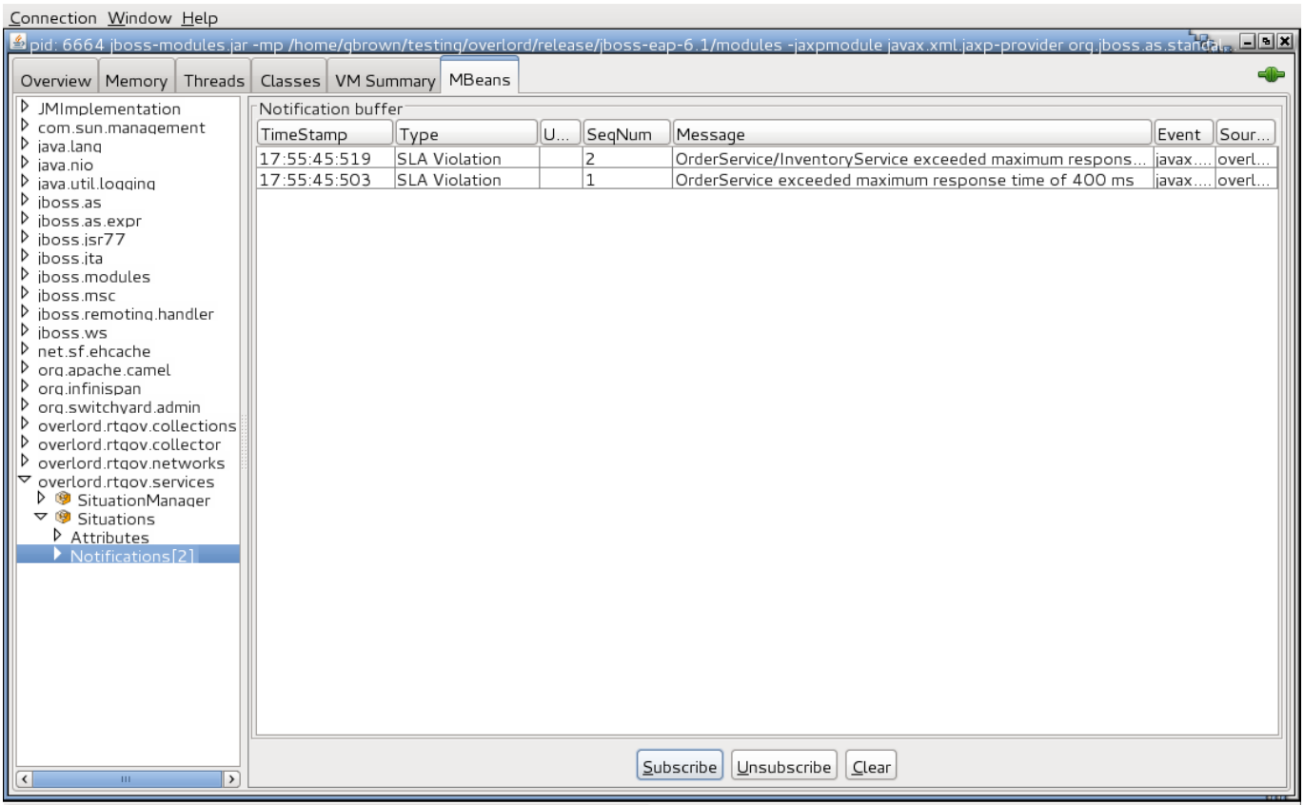
Figure 3.6. Response



3.10.1.3.2. JMX Console

The Situations active collection source also generates JMX notifications that can be subscribed to using a suitable JMX management application. For example, using JConsole we can view the SLA violation:

Figure 3.7. JMX Console



3.10.1.3.3. Accessing Results Within a Custom Application

Apart from accessing the information via REST or JMX, you may also have more direct access to the active collection results. This section describes the custom application defined in the **`${rtgov}/samples/sla/monitor`** folder.

The following code shows how the custom application initializes access to the relevant active collections:

```
@Path("/monitor")
@ApplicationScoped
public class SLAMonitor {

    private static final String SERVICE_RESPONSE_TIMES =
"ServiceResponseTimes";
    private static final String SITUATIONS = "Situations";

    private static final Logger
LOG=Logger.getLogger(SLAMonitor.class.getName());

    private ActiveCollectionManager _acmManager=null;

    private ActiveList _serviceResponseTime=null;
    private ActiveList _situations=null;

    /**
     * This is the default constructor.
     */
    public SLAMonitor() {

        try {
```

```
            _acmManager =
ActiveCollectionManagerAccessor.getActiveCollectionManager();

            _serviceResponseTime = (ActiveList)

_acmManager.getActiveCollection(SERVICE_RESPONSE_TIMES);

            _situations = (ActiveList)
                    _acmManager.getActiveCollection(SITUATIONS);

        } catch (Exception e) {
            LOG.log(Level.SEVERE, "Failed to initialize active collection
manager", e);
        }

    }
```

When the REST request is received (for example, for SLA violations defined as Situations):

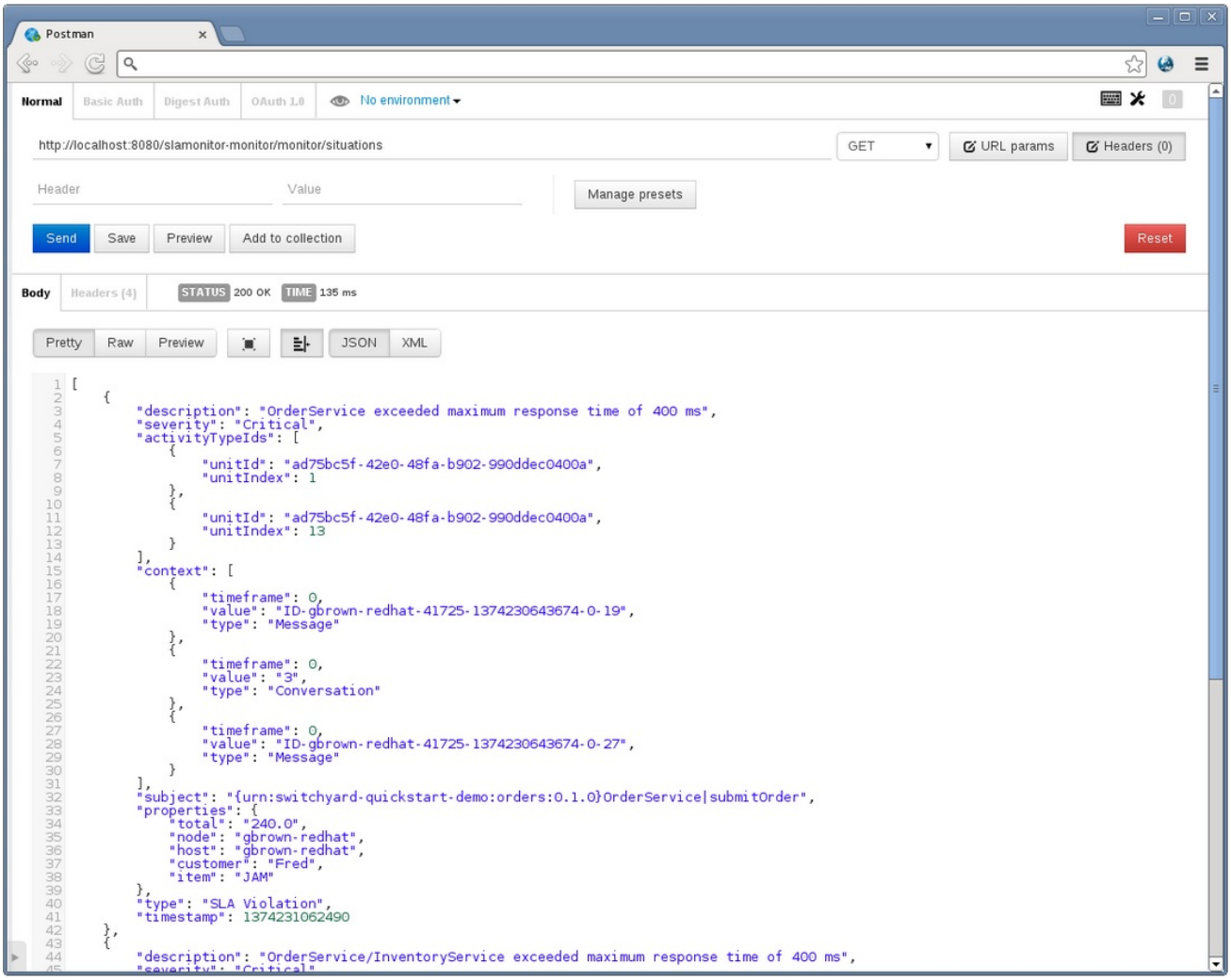
```
@GET
    @Path("/situations")
    @Produces("application/json")
    public java.util.List<Situation> getSituations() {
        java.util.List<Situation> ret=new java.util.ArrayList<Situation>
();

        for (Object obj : _situations) {
            if (obj instanceof Situation) {
                ret.add((Situation)obj);
            }
        }

        return (ret);
    }
```

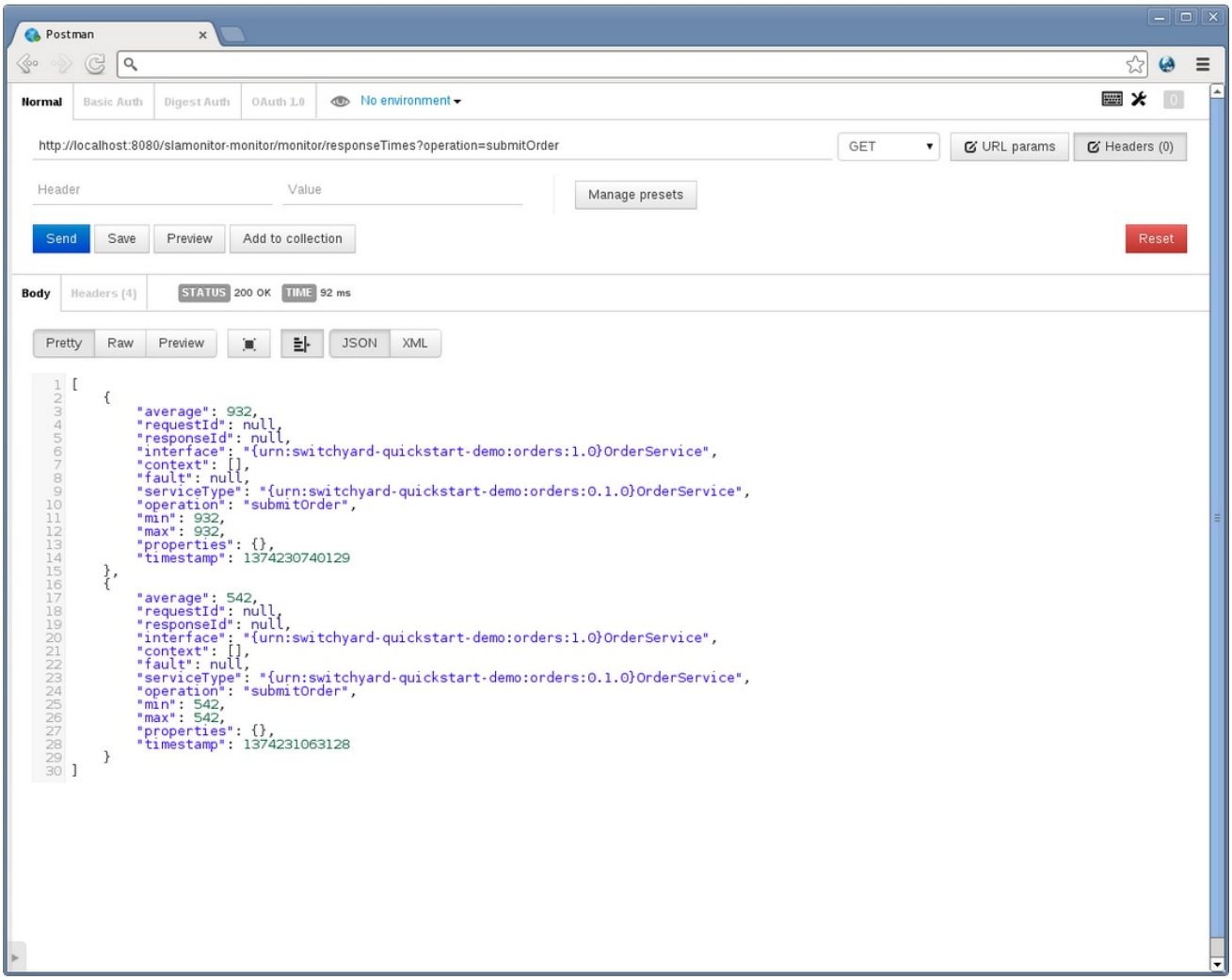
To see the SLA violations, send a REST GET request to http://localhost:8080/slamonitor-monitor/monitor/situations . This returns the following information:

Figure 3.8. Response



You can also request the list of response time information from the same custom service, using the URL http://localhost:8080/slamonitor-monitor/monitor/responseTimes?operation=submitOrder:

Figure 3.9. Response



CAUTION

If no query parameter is provided, then response times for all operations will be returned.

3.10.2. Report

The example located in the `samples/sla/report` folder demonstrates how to provide pluggable reports that can access information in the activity store. This example uses the activity information to compile a Service Level Agreement report, highlighting violations above a specified response time.

This example shows:

- how to configure a pluggable report
- how to generate the report via a REST API

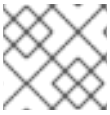
This example provides a simple Service Level Agreement report, based on identifying service invocations that exceed a specified maximum response time over an optionally specified averaged duration. If the averaged duration is not specified, then each service invocation is checked to determine if it exceeded the maximum response time. If so, it gets added to the report. If the averaged duration is specified, then when an invocation is detected (that exceeds the max response time), then all other suitable invocations within the specified duration are averaged to determine if the response time overall still exceeds the specified maximum. This is to ensure that periodic spikes are not unnecessarily reported.

It is also possible to optionally specify a business calendar, which can be used to determine the business period in which activities are of interest. If SLA violations occur outside the specified business calendar, then they are not relevant.

The report definition is:

```
[
  {
    "name" : "SLAReport",
    "generator" : {
      "@class" : "org.overlord.rtgov.reports.MVELReportGenerator",
       "scriptLocation" : "SLAReport.mvel"
    }
  }
]
```

You can find the MVEL based report generator script in the **${rtgov}/samples/sla/report/src/main/resources** folder.



NOTE

Currently the report parameters serviceType, operation and principal are not used.

3.10.2.1. Quickstart Example

The quickstart example in this section demonstrates how report definitions can be deployed to the Runtime Governance infrastructure and invoked to generate a report instance via a REST service.

3.10.2.2. Installing the Example

1. Start the Switchyard server using the following command from the bin folder:

```
./standalone.sh -c standalone-full.xml
```

2. Run the following mvn command from the **${rtgov}/samples/sla/report** folder:

```
mvn jboss-as:deploy
```

3.10.2.3. Running the Example

To demonstrate a Service Level Agreement report, you can create relevant activities that can be reported upon. For this, send multiple instances of the following messages to the example Switchyard application at :

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <orders:submitOrder xmlns:orders="urn:switchyard-quickstart-
demo:orders:1.0">
         <order>
            <orderId>1</orderId>
            <itemId>BUTTER</itemId>
            <quantity>100</quantity>
            <customer>Fred</customer>
         </order>
```

```
      </orders:submitOrder>
   </soap:Body>
</soap:Envelope>

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <orders:submitOrder xmlns:orders="urn:switchyard-quickstart-
demo:orders:1.0">
         <order>
            <orderId>3</orderId>
            <itemId>JAM</itemId>
            <quantity>100</quantity>
            <customer>Fred</customer>
         </order>
      </orders:submitOrder>
   </soap:Body>
</soap:Envelope>
```

Send the second message few time, as this is the one that results in SLA violations.

The message can be sent using an appropriate SOAP client (for example, SOAP-UI) or by running the test client available with the Switchyard application, by running the following commands from the **${rtgov}/samples/ordermgmt/app** folder:

```
mvn exec:java -Dreq=order1
mvn exec:java -Dreq=order3
```

To generate a report, send a GET request to the following URL using Basic Authentication with a valid Application Realm username and password:

```
http://localhost:8080overlordrtgovreportgeneratereport=SLAReport&startDay=
1&startMonth=1&startYear=2013&endDay=31&endMonth=12&endYear=2013&maxRespon
seTime=400&averagedDuration=450
```

This returns a report with name SLAReport, containing violations that occur within the year 2013. To experiment with the default business calendar, append the following to the end of the URL:

```
&calendar=Default
```

This also identifies what percentage of the business working period has been impacted by the SLA violations.

CHAPTER 4. S-RAMP

SOA Repository Artifact Model and Protocol (S-RAMP) is supported by a Technical Committee at OASIS. It is a specification of SOA repository that provides a common data model and protocol for interacting with a repository of SOA artifacts. For more information, see https://www.oasis-open.org/committees/s-ramp/charter.php.

S-RAMP supports interoperability between repository implementations by standardizing on a data model and API. The S-RAMP specification includes the following:

- A foundation document that describes the core concepts.
- An Atom based protocol binding document that describes an Atom based API.

An S-RAMP repository primarily stores artifacts. An artifact comprises of the following metadata:

- Core Properties
- Custom Properties
- Classifiers
- Relationships

4.1. S-RAMP WORKING SAMPLES

This chapter discusses the S-RAMP concepts, implementation, and Integration. To help you understand them better, we ship the following working examples, which are installed at **`jboss-eap-6.1/quickstarts/sramp`**:

- s-ramp-demos-archive-package
- s-ramp-demos-classifications
- s-ramp-demos-custom-deriver
- s-ramp-demos-derived-artifacts
- s-ramp-demos-mvn-integration
- s-ramp-demos-ontologies
- s-ramp-demos-project
- s-ramp-demos-properties
- s-ramp-demos-query
- s-ramp-demos-relationships
- s-ramp-demos-shell-command
- s-ramp-demos-simple-client
- s-ramp-demos-switchyard
- s-ramp-demos-switchyard-multiapp

4.2. S-RAMP USER MANAGEMENT

By default S-RAMP uses the standard EAP Application Realm configuration as its authentication source. This means that adding users is a simple matter of using the existing EAP add-user script. If you are running on Windows you can use the **`add-user.bat`** script. Otherwise run the **`add-user.sh`** script. Both of these scripts can be found in EAP's bin directory.

Here is an example of how to add an S-RAMP user using the add-user.sh script:

```
[user@host jboss-eap-6.1]$ pwd
/home/user/FSW6/jboss-eap-6.1
[user@host jboss-eap-6.1]$ ./bin/add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): b

Enter the details of the new user to add.
Realm (ApplicationRealm) : ApplicationRealm
Username : fitzuser
Password : P4SSW0RD!
Re-enter Password : P4SSW0RD!
What roles do you want this user to belong to? (Please enter a comma
separated list, or leave blank for none)[  ]: overlorduser,admin.sramp
About to add user 'fitzuser' for realm 'ApplicationRealm'
Is this correct yes/no? yes
Added user 'fitzuser' to file '/home/user/FSW6/jboss-eap-
6.1/standalone/configuration/application-users.properties'
Added user 'fitzuser' to file '/home/user/FSW6/jboss-eap-
6.1/domain/configuration/application-users.properties'
Added user 'fitzuser' with roles overlorduser,admin.sramp to file
'/home/user/FSW6/jboss-eap-6.1/standalone/configuration/application-
roles.properties'
Added user 'fitzuser' with roles overlorduser,admin.sramp to file
'/home/user/FSW6/jboss-eap-6.1/domain/configuration/application-
roles.properties'
Is this new user going to be used for one AS process to connect to another
AS process?
e.g. for a slave host controller connecting to the master or for a
Remoting connection for server to server EJB calls.
yes/no? no
```

Required Roles

TThere are several roles that the user must have in order to interact with the S-RAMP repository. These roles are as follows:

- overlorduser : users must have this role in order to access the S-RAMP user interface (browser)
- admin.sramp : users must have this role in order to access the S-RAMP repository (both read and write)

NOTE

If you change the S-RAMP repository name in the **`standalone.xml`** file and set the new repository name in **`standalone/configuration/sramp.properties`** (under sramp.config.jcr.repository.jndi-path), make sure you modify the user's roles. If the role that grants users access to the ModeShape repository is *admin.sramp*, where the ModeShape role is *admin* on repository named *sramp*, ensure you change this value to *admin.<new repository name>* in the **`application-roles.properties`** file.

4.3. ARTIFACT METADATA

4.3.1. Core Properties

All artifacts in S-RAMP contain a set of core properties such as name, description, and creation date. The server automatically sets many of these properties when the artifact is added or updated. Other properties such as description are set by clients. Every artifact has an Artifact Model and an Artifact Type. These two properties determine what kind of artifact it is.

Some artifact types contain additional core properties. For example, the Document artifact type includes additional core properties of contentType and contentSize, while the XsdDocument artifact type includes the targetNamespace property.

NOTE

We support only the S-RAMP shell and user interface that are shipped with JBoss Enterprise SOA Platform 6, however you can use your own custom clients.

4.3.2. Custom Properties

In addition to the core properties, an artifact may have additional properties called the custom properties set on it by clients. These custom properties are arbitrary name-value pairs. A custom property may not have the same name as a core property.

4.3.3. Classifiers

An ontology is a hierarchy of tags defined as a subset of the OWL Lite language. A classifier is a node in an ontology that is previously uploaded to the repository. Classifiers are like keywords or tags except that they are hierarchical. Every artifact has a collection of classifiers configured by the client.

Here is an example of how you can configure the repository with a pre-defined set of hierarchical tags (classifiers) that can be associated with an artifact:

Consider a case where a repository administrator defines and uploads the following ontology:

```
World
  |-> North America
     |-> United States
         |-> Alabama
         |-> Alaska
     |-> Mexico
     |-> Canada
  |-> South America
  |-> Australia
```

Once this ontology is added to the repository, the clients can add classifiers such as #Alaska or #Canada on artifacts. This provides a way to tag artifacts in interesting and meaningful ways, and a useful means of querying.

4.3.4. Relationships

An S-RAMP relationship is a uni-directional link between a source artifact and a target artifact. Artifacts can have arbitrary, client-defined relationships. Every relationship has a name and a target artifact. For example, a client may define a relationship with name "documentedBy" between a wsdl artifact and a text or PDF artifact, indicating that the latter provides documentation for the former.

4.4. S-RAMP DATA MODELS

The S-RAMP specification defines a number of built-in artifact types, while also allowing clients to define their own implicit types. This section describes these different models.

An artifact may have document (such as a file) content or it may be a purely logical artifact. In either case, clients typically add artifacts to the repository directly (for example, via the S-RAMP Atom API). Additionally, some document style artifact types when added to the repository, result in the creation of a set of derived artifacts. For example, if an XSD document is added to the repository, the server automatically extracts the element declarations from the content of the file resulting in a set of additional artifacts related to the original.

4.4.1. Core Data Model

The S-RAMP core data model defines some basic artifact types that include Document and XmlDocument. These basic types allow clients to add simple files to the repository as artifacts.

Table 4.1. S-RAMP Core Data Model

Artifact Type	Parent Type	Properties
Document		contentType, contentSize, contentHash
XmlDocument	Document	contentEncoding

4.4.2. XML Schema Data Model

The XSD model defines a single document style artifact, XsdDocument, and a number of derived artifact types. When an XSD document is added to the repository, the server additionally indexes the artifact by automatically creating a number of derived artifacts of the following types from the XSD content:

Table 4.2. S-RAMP XML Schema Data Model

Artifact Type	Parent Type	Properties
XsdDocument	XmlDocument	targetNamespace
AttributeDeclaration	<derived>	ncName, namespace

Artifact Type	Parent Type	Properties
ElementDeclaration	<derived>	ncName, namespace
SimpleTypeDeclaration	<derived>	ncName, namespace
ComplexTypeDeclaration	<derived>	ncName, namespace

4.4.3. WSDL Data Model

The WSDL model defines a single document style artifact called WsdlDocument, and a number of derived artifact types. Similar to the XsdDocument type, when you add a WSDL document to the repository, the server automatically derives additional artifacts from the content of the WSDL file. Here is the list of these artifacts:

Table 4.3. WSDL Artifacts

Artifact Type	Parent Type	Properties
WsdlDocument	XmlDocument	targetNamespace, xsdTargetNamespaces
WsdlService	<derived>	ncName, namespace
Port	<derived>	ncName, namespace
WsdlExtension	<derived>	ncName, namespace
Part	<derived>	ncName, namespace
Message	<derived>	ncName, namespace
Fault	<derived>	ncName, namespace
PortType	<derived>	ncName, namespace
Operation	<derived>	ncName, namespace
OperationInput	<derived>	ncName, namespace
OperationOutput	<derived>	ncName, namespace
Binding	<derived>	ncName, namespace
BindingOperation	<derived>	ncName, namespace
BindingOperationInput	<derived>	ncName, namespace

Artifact Type	Parent Type	Properties
BindingOperationOutput	<derived>	ncName, namespace
BindingOperationFault	<derived>	ncName, namespace

4.4.4. Policy Data Model

The Policy Data Model represents the primary components of a WS-Policy document. Here is the list of artifacts for Policy Data Model:

Table 4.4. Policy Data Model Artifacts

Artifact Type	Parent Type
PolicyDocument	XmlDocument
PolicyExpression	<derived>
PolicyAttachment	<derived>

4.4.5. SOA Data Model

The SOA model provides a link to the work done by the Open Group SOA Ontology group. All of the artifacts in this model are non-document artifacts that are directly instantiated by clients.

Table 4.5. SOA Data Model Artifacts

Artifact Type
HumanActor
Choreography
ChoreographyProcess
Collaboration
CollaborationProcess
Composition
Effect
Element
Event

Artifact Type
InformationType
Orchestration
OrchestrationProcess
Policy
PolicySubject
Process
Service
ServiceContract
ServiceComposition
ServiceInterface
System
Task

4.4.6. Service Implementation Data Model

The Service Implementation model adds SOA service implementation artifact types under the SOA Data Model.

Table 4.6. Service Implementation Data Model Artifacts

Artifact Type	Properties
Organization	end
ServiceEndpoint	end, url
ServiceInstance	end, url
ServiceOperation	end, url

4.4.7. Custom or Extension Data Models

Clients can define their own implicit data models by using the *ext* model space defined by the S-RAMP specification. This allows clients to add documents with custom artifact types. For example, a client can

add an artifact to **`/s-ramp/ext/PdfDocument`**. This provides a way for clients to define their own data models with their own properties and relationships. Note that the server does not have a definition of the model and it is up to the client to properly conform to their own implicit model. Custom properties and user-defined relationships allow clients to richly define their own models.

Here is an example where a client defines the following Data Model for a J2EE web application domain:

Table 4.7. Custom/Extension Data Models Artifacts

Artifact Type	Parent Type	Properties
WebXmlDocument	ExtendedDocument	displayName
ServletFilter	ExtendedArtifactType	displayName, filterClass
Servlet	ExtendedArtifactType	servletClass, loadOnStartup

4.5. QUERY LANGUAGE

4.5.1. S-RAMP Query Language

S-RAMP defines a query language that allows clients to search artifacts by various criteria. The S-RAMP query language is a subset of the XPath 2.0 language, designed specifically to find and select S-RAMP artifacts.

The query language allows clients to search artifacts based on any of the following artifact meta-data:

- Core Properties
- Custom Properties
- Classifiers
- Relationships

Here is a basic structure of a typical S-RAMP query:

```
/s-ramp/<artifactModel>/<artifactType>/[ <artifact-predicate>
]/relationship[ <target-artifact-predicate> ]
```

4.5.2. S-RAMP Query Examples

The following table lists a range of S-RAMP queries:

Table 4.8. S-RAMP Queries

Query	What It Selects
`/s-ramp`	All artifacts.

Query	What It Selects
/s-ramp/core	All Core Model artifacts.
/s-ramp/xsd/XsdDocument	All XsdDocument artifacts.
/s-ramp/xsd/XsdDocument[@name='core.xsd']	XsdDocument artifacts named core.xsd.
/s-ramp/xsd/XsdDocument[@name='core.xsd' and @version='1.0']	XsdDocument artifacts named core.xsd and versioned as 1.0.
/s-ramp/soa[@myCustomProperty='foo']	SOA artifacts with a custom property named myCustomProperty that has value foo.
/s-ramp/core[classifiedByAnyOf(., Maine, Alaska)]	Core artifacts classified by either Maine or Alaska (presumably from the Regions ontology).
/s-ramp/wsdl/PortType[@name='OrderServicePT']/operation	Artifacts related to any PortType artifact named OrderServicePT via a relationship named operation. This effectively returns all of the order service port type operations.
/s-ramp/ext/ServletFilter[relatedDocument[@uuid='12345']]	All servlet filter artifacts derived from (that is, contain a relatedDocument relationship to) an artifact with UUID 12345.
/s-ramp/wsdl/Message[xp2:matches(.,'get.*')]/part[element]	Element style WSDL parts from WSDL messages with names starting with get.

NOTE

Use single quotes for query arguments and double quotes to surround the query on execution.

For example,

```
s-ramp> s-ramp:query "/s-ramp/xsd/XsdDocument[@name='core.xsd'
and @version='1.0']"
```

4.6. S-RAMP REST API

This section describes the API for REST clients that are written by the users. These are different than the S-RAMP shell and user interface shipped with JBoss Enterprise SOA Platform 6.

The S-RAMP specification outlines a data model and protocol that define how a repository should store and manipulate artifacts. The S-RAMP specification does not dictate the format of the Atom REST

endpoints. Instead, the client is expected to query a service document endpoint and inspect it to find the various relevant endpoints. The specification does present a notional format, but implementations are not required to follow it. In general, the Atom API data models are used to wrap custom S-RAMP specific XML structures.

4.6.1. Atom Entry Document

Atom Entry documents are used when dealing with individual artifacts.

The Atom entry document is the Atom representation of an S-RAMP artifact object. There are two Atom representations of an S-RAMP artifact object:

- A summary Atom entry that appears in an Atom feed document. Summary entries do not include the S-RAMP structured extension element (S-RAMP foreign markup).
- A complete Atom entry which includes the S-RAMP structured extension element.

4.6.2. Atom Feed Document

A feed in S-RAMP is XML data which contains Atom entries. S-RAMP defines several Atom feeds which are used to access fine-grained support for relationships.

An Atom Feed Document is a representation of an Atom feed. It includes metadata about the feed, and the entries associated with it. Use Atom Feed documents when dealing with lists of documents.

4.6.3. Adding Artifacts

From the protocol standpoint, S-RAMP provides two types of artifacts:

- Document Style Artifacts

  These artifacts are based on files or binary content. For the document style artifacts, the client must POST the binary content to the correct Atom Endpoint.

- Logical Artifacts

  These artifacts are direct instantiation artifacts. For the logical artifacts, there is no document content and the client must POST an Atom Entry containing an S-RAMP artifact XML entity to the appropriate endpoint. If successful, the server responds with an Atom Entry containing the full meta data of the newly created artifact.

You can add one of these artifacts using the POST request as shown below:

```
POST /s-ramp/{model}/{type}
```

4.6.4. Updating Artifacts

A client can update the artifact metadata such as properties, classifiers, and relationships. You can update artifacts using the PUT request as shown below to the endpoint of an artifact:

```
PUT /s-ramp/{model}/{type}/{uuid}
```

You can find the endpoint of an artifact either by querying for the artifact or as part of the Atom Entry returned when the artifact was created.

4.6.5. Deleting Artifacts

A cient can delete an artifact by performing a DELETE request to the endpoint of an artifact as shown below:

```
DELETE /s-ramp/{model}/{type}/{uuid}
```

4.6.6. S-RAMP Queries

4.6.6.1. Running an S-RAMP Query

Running an S-RAMP query means issuing a GET or POST to the S-RAMP query endpoint. S-RAMP provides full feed for all Artifact Models and Artifact Types. In both the cases, the response is an Atom Feed where each Entry provides summary information about an artifact in the respository.

Only a subset of the core properties, such as name and description, are mapped to the Atom Entry in a feed. To retrieve full details such as custom properties, classifiers, and relationships about a given entry in a feed, the client must issue an additional GET request.

4.6.6.2. S-RAMP GET and POST Query Parameters

A client can query S-RAMP by performing either GET or POST to the following notional endpoint:

```
GET /s-ramp
```

S-RAMP supports the following parameters for GET and POST:

Table 4.9. GET/POST Query Parameters

Parameter	Description
query	The S-RAMP query.
startPage	The page to start from.
startIndex	The index number to start from.
count	The number of artifacts to return.
orderBy	The sort order to use when creating the feed.
ascending	The sort direction to use when creating the feed.
propertyName	Additional custom property to return for each artifact in the feed. You can include this property multiple times.

4.6.6.3. S-RAMP Feeds Parameters

When retrieving a simple model or type feed, the client must issue a GET request to the appropriate model or type endpoint as shown below:

```
GET /s-ramp/{model}
GET /s-ramp/{model}/{type}
```

S-RAMP supports the following parameters for retrieving feeds:

Table 4.10. S-RAMP Feeds Parameters

Parameter	Description
startPage	The page to start from.
startIndex	The index number to start from.
count	The number of artifacts to return.
orderBy	The sort order to use when creating the feed.
ascending	The sort direction to use when creating the feed.
propertyName	Additional custom property to return for each artifact in the feed. This property can be included multiple times.

4.6.6.4. Retrieving Full Metadata for an Artifact

In order to retrieve the full metadata for an artifact, the client must issue a GET request to the appropriate artifact endpoint as shown below:

```
GET /s-ramp/{model}/{type}/{uuid}
```

This is necessary after a query or feed, when only the summary information is available. The summary information found in a feed or query response contains the UUID of the artifact, as well as a URL to the endpoint needed to retrieve the full artifact details.

4.6.6.5. Batch Changes to S-RAMP Archives

The batch processing function is a powerful additional feature of the S-RAMP API. The batch processing endpoint allows the client to POST an S-RAMP package, which can contain multiple Atom Entries and binary files. The package allows a client to add, update, and delete multiple artifacts in a single batch.

4.7. S-RAMP IMPLEMENTATION

The S-RAMP implementation strives to be a fully compliant reference implementation of the S-RAMP specification. This chapter describes the overall architecture of the implementation and also provides some information about how to configure it.

S-RAMP also provides a Java based client library that consumers can use to integrate their own applications with an S-RAMP compliant server.

4.7.1. S-RAMP Server

The server implementation is a conventional Java web application (WAR). The following technologies are used to provide the various components that make up the server implementation:

- JCR (ModeShape): Used as the persistence engine, where all S-RAMP data is stored. Artifacts and ontologies are both stored as nodes in a JCR tree. All S-RAMP queries are mapped to JCRSQL2 queries for processing by the JCR API. The ModeShape JCR implementation is used by default. However, the persistence layer is pluggable allowing alternative providers to be implemented in the future.
- AX-RS (RESTEasy): Used to provide the S-RAMP Atom based REST API. The S-RAMP specification documents an Atom based REST API that implementations must make available. The S-RAMP implementation uses JAX-RS (specifically RESTEasy) to expose all of the REST endpoints defined by the specification.
- JAXB: Used to expose a Java data model based on the S-RAMP data structures defined by the specification (S-RAMP XSD schemas).

4.7.1.1. Configuring Server

You can configure the server by providing a configuration file to the server on startup. You can provide the configuration file in a number of ways:

- **`sramp.properties`**: You can provide this external file in the JBoss application server's configuration directory. An alternative location is the home directory of the user running the application server.
- custom external file: You can specify a custom location for the **`sramp.properties`** file by starting the application server with the sramp.config.file.name system property set. This is typically done using **`-Dsramp.config.file.name=<pathToFile>`** on the application server's command line startup script (often in JAVA_OPTS).
- On the classpath: If no external file is found, you can use the classpath to lookup a default configuration.

The configuration file is a simple Java properties file, with the following properties available to be set:

```
# The base URL of the S-RAMP server - can be useful in some advanced
configurations where
# the incoming Request URL is not the canonical server address.
sramp.config.baseurl = http://host:port/context
# Turn on/off auditing of changes to S-RAMP artifacts
sramp.config.auditing.enabled = true
# Turn on/off auditing of changes to derived S-RAMP artifacts
sramp.config.auditing.enabled-derived = true
```

4.7.1.2. Extending Custom Deriver

Part of the S-RAMP specification is the concept of Derived content. This happens when an artifact of a certain type is added to the S-RAMP repository. The server is responsible for creating relevant derived artifacts from it. For example, when an XML Schema (XSD) document is added to the repository, the server is responsible for automatically creating an artifact for every top level Element, Complex Type, Simple Type, and Attribute declaration found in the XSD.

The S-RAMP implementation includes Artifact Derivers for all of the logical models defined by the S-RAMP specification (such as WSDL, XSD, Policy). However, it also provides a mechanism that allows

users to provide Artifact Derivers for their own artifact types. This is done by performing the following steps:

Procedure 4.1. Task

1. Write a custom Deriver Java class. It must implement **`ArtifactDeriver`**.
2. Create a **`DeriverProvider`** (a class that implements DeriverProvider) used to map artifact types to implementations of **`ArtifactDeriver`**.
3. Provide a text file named **`org.overlord.sramp.common.derived.DeriverProvider`** in the location **`META-INF/services`**. The content of this file must be a single line containing the fully qualified classname of the class defined in the previous step.
4. Package everything into a JAR and make it available either on the classpath or in an external directory. Configure the external directory by setting property sramp.derivers.customDir.

4.7.2. S-RAMP Client

The S-RAMP Clients include the following:

- S-RAMP Client Library (Jar)
- S-RAMP Interactive Shell (CLI)
- S-RAMP Browser (UI)
- S-RAP+Maven Integration (maven wagon)

The S-RAMP Client Library is a Java client library implementing the S-RAMP Atom API. Other items in the list use the S-RAMP Client Library when connecting to the S-RAMP repository. This section describes how to use the S-RAMP Client Library.

4.7.2.1. S-RAMP Client Usage Examples

The S-RAMP client is a simple Java based client library and can be included in a Maven project by including the following **`pom.xml`** dependency:

```
<dependency>
      <groupId>org.overlord.sramp</groupId>
      <artifactId>s-ramp-client</artifactId>
      <version>${sramp.client.version}</version>
</dependency>
```

Once the library is included in your project, you can use the client by instantiating the **`SrampAtomApiClient`** class. Note that the client class supports pluggable authentication mechanisms, although BASIC auth is just a matter of including the username and password upon construction of the client. For details, refer to the javadoc of the required class. Here are some usage examples to help you get started:

- Upload an XSD document to S-RAMP

```
SrampAtomApiClient client = new SrampAtomApiClient(urlToSramp);
String artifactFileName = getXSDArtifactName();
InputStream is = getXSDArtifactContentStream();
```

```
ArtifactType type = ArtifactType.XsdDocument();
BaseArtifactType artifact =
client.uploadArtifact(ArtifactType.XsdDocument(), is,
artifactFileName);
```

- Create a custom artifact in S-RAMP (meta-data only, no file content)

```
SrampAtomApiClient client = new SrampAtomApiClient(urlToSramp);
ExtendedArtifactType artifact = new ExtendedArtifactType();
artifact.setArtifactType(BaseArtifactEnum.EXTENDED_ARTIFACT_TYPE);
artifact.setExtendedType("MyArtifactType");
artifact.setName("My Test Artifact #1");
artifact.setDescription("Description of my test artifact.");
BaseArtifactType createdArtifact = client.createArtifact(artifact);
```

- Retrieve full meta-data for an XSD artifact by its UUID

```
SrampAtomApiClient client = new SrampAtomApiClient(urlToSramp);
String uuid = getArtifactUUID();
BaseArtifactType metaData =
client.getArtifactMetaData(ArtifactType.XsdDocument(), uuid);
```

- Retrieve artifact content

```
SrampAtomApiClient client = new SrampAtomApiClient(urlToSramp);
String uuid = getArtifactUUID();
InputStream content =
client.getArtifactContent(ArtifactType.XsdDocument(), uuid);
```

- Query the S-RAMP repository (by artifact name)

```
SrampAtomApiClient client = new SrampAtomApiClient(urlToSramp);
String artifactName = getArtifactName();
QueryResultSet rset = client.buildQuery("/s-
ramp/xsd/XsdDocument[@name = ?]")
        .parameter(artifactName)
        .count(10)
        .query();
```

4.7.2.2. Ontologies

The S-RAMP implementation provides an extension to the Atom based REST API to support management of ontologies. You can use any of the client's ontology related methods when communicating with the implementation of S-RAMP, however it is likely to fail when communicating with any other S-RAMP server. The S-RAMP client supports adding, updating, and getting (both individual and a full list) ontologies from the S-RAMP repository.

4.7.2.3. Auditing

The S-RAMP implementation offers an extension to the Atom based REST API to get and set auditing information for artifacts in the repository.

4.7.2.4. Custom Expander

A special feature of the S-RAMP client is the ability to automatically expand archive style artifacts (artifacts that are JARs, WARs, ZIPs, etc). This feature is similar to how the server creates Derived content. The result is that certain files from the archive being uploaded as an S-RAMP artifact are extracted from the archive and also uploaded to the server. When this happens, these expanded artifacts are added with an S-RAMP relationship (expandedFromDocument) that points to the archive artifact they were expanded from.

The S-RAMP implementation comes with a few built-in expanders (such as, java archive and SwitchYard archive). Additionally, custom expanders can be created and provided by implementing **`ZipToSrampArchiveProvider`**. In order to inform the S-RAMP client about the custom provider, you need to put it in a JAR along with a file named **`META-INF/services/org.overlord.sramp.atom.archive.expand.registry.ZipToSrampArchiveProvider`**. The contents of this file must be a single line with the fully qualified Java classname of the provider implementation.

4.8. S-RAMP REST API ENDPOINTS

The S-RAMP Atom API protocol binding does not dictate the format of the API endpoints. Clients request the /servicedocument and to inspect the workspaces inside it. However, the implementations endpoints conform to the notional syntax described in the S-RAMP specifications foundation document. The following table lists the endpoints available in the Red Hat JBoss Governance implementation:

Table 4.11. Implementation Endpoints

Endpoint	Name
GET /s-ramp/servicedocument	Get Service Document
POST /s-ramp/{model}/{type}	Publish Artifact
PUT /s-ramp/{model}/{type}/{uuid}	Update Artifact
PUT /s-ramp/{model}/{type}/{uuid}/media	Update Artifact Content
GET /s-ramp/{model}/{type}/{uuid}	Get Artifact
GET /s-ramp/{model}/{type}/{uuid}/media	Get Artifact Content
DELETE /s-ramp/{model}/{type}/{uuid}	Delete Artifact
GET /s-ramp/{model}	Get Artifact Feed (by model)
GET /s-ramp/{model}/{type}	Get Artifact Feed (by type)
GET /s-ramp	Query
POST /s-ramp	Query

Endpoint	Name
POST /s-ramp	Batch Processing
POST /s-ramp/ontology	Add Ontology
GET /s-ramp/ontology	List Ontologies
PUT /s-ramp/ontology/{uuid}	Update Ontology
GET /s-ramp/ontology/{uuid}	Get Ontology
DELETE /s-ramp/ontology/{uuid}	Delete Ontology
GET /s-ramp/audit/artifact/{artifactUuid}	Get Artifact Audit History
GET /s-ramp/audit/user/{username}	Get User Audit History
POST /s-ramp/audit/artifact/{artifactUuid}	Add Artifact Audit Entry
GET /s-ramp/audit/artifact/{artifactUuid}/{auditEntryUuid}	Get Artifact Audit Entry

4.8.1. Get Service Document API

The Get Service Document API (`/s-ramp/servicedocument`) retrieves the service document. The service document contains a workspace for each of the S-RAMP data models supported by the server.

Table 4.12. GET Method Response

HTTP Method	Request	Response
GET	N/A	Atom Service Document

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<app:service xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:app="http://www.w3.org/2007/app">
    <app:workspace>
        <atom:title>Core Model</atom:title>
        <app:collection href="http://example.org/s-ramp/core">
            <atom:title>Core Model Objects</atom:title>
            <app:accept>application/zip</app:accept>
            <app:categories fixed="yes">
                <atom:category label="Document" scheme="urn:x-s-
ramp:v1:type" term="Document"/>
```

```
                <atom:category label="XML Document" scheme="urn:x-s-
ramp:v1:type" term="XmlDocument"/>
            </app:categories>
        </app:collection>
        <app:collection href="http://example.org/s-ramp/core/Document">
            <atom:title>Documents</atom:title>
            <app:accept>application/octet-stream</app:accept>
            <app:categories fixed="yes">
                <atom:category label="Document" scheme="urn:x-s-
ramp:v1:type" term="Document"/>
            </app:categories>
        </app:collection>
        <app:collection href="http://example.org/s-ramp/core/XmlDocument">
            <atom:title>XML Documents</atom:title>
            <app:accept>application/xml</app:accept>
            <app:categories fixed="yes">
                <atom:category label="XML Document" scheme="urn:x-s-
ramp:v1:type" term="XmlDocument"/>
            </app:categories>
        </app:collection>
    </app:workspace>
</app:service>
```

NOTE

This example only includes the Core data model and thus the service document has a single workspace. The full service document has multiple workspaces, one for each data model supported by the server.

4.8.2. Publishing a New Artifact Into a Repository

4.8.2.1. Publish Artifact API

The Publish Artifact API publishes a new artifact into the repository. You can invoke this endpoint in the following ways, depending on the type of artifact being published:

- Document Style Artifact
- Non-Document Style Artifact
- Document Style Artifact with Meta Data

4.8.2.2. Publish a Document Style Artifact

You can publish a document style artifact by POSTing the binary content of the document to the appropriate endpoint.

```
/s-ramp/{model}/{type}
```

Table 4.13. Post Method Response for Document Style Artifact

HTTP Method	Request	Response
POST	Binary File	Atom Entry

Example Request:

```
POST /s-ramp/core/Document HTTP/1.1

This is a simple text document, uploaded as an artifact
into S-RAMP.
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:entry xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
  xmlns:xlink="http://www.w3.org/1999/xlink" s-ramp:derived="false">
  <atom:title>test.txt</atom:title>
  <atom:link
    href="http://example.org/s-ramp/core/Document/05778de3-be85-4696-b5dc-
d889a27f1f6e/media"
    rel="alternate" type="text/plain" />
  <atom:link href="http://example.org/s-ramp/core/Document/05778de3-be85-
4696-b5dc-d889a27f1f6e"
    rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
  <atom:link
    href="http://example.org/s-ramp/core/Document/05778de3-be85-4696-b5dc-
d889a27f1f6e/media"
    rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;" />
  <atom:link href="http://example.org/s-ramp/core/Document/05778de3-be85-
4696-b5dc-d889a27f1f6e"
    rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
  <atom:category label="Document" scheme="x-s-ramp:2010:type"
term="Document" />
  <atom:category label="Document" scheme="x-s-ramp:2010:model" term="core"
/>
  <atom:updated>2013-05-14T13:43:09.708-04:00</atom:updated>
  <atom:id>05778de3-be85-4696-b5dc-d889a27f1f6e</atom:id>
  <atom:published>2013-05-14T13:43:09.708-04:00</atom:published>
  <atom:author>
    <atom:name>ewittman</atom:name>
  </atom:author>
  <atom:content
    src="http://example.org/s-ramp/core/Document/05778de3-be85-4696-b5dc-
d889a27f1f6e/media"
    type="text" />
  <s-ramp:artifact>
    <s-ramp:Document artifactType="Document" contentSize="69"
contentType="text/plain"
      createdBy="&lt;anonymous&gt;" createdTimestamp="2013-05-
14T13:43:09.708-04:00" lastModifiedBy="&lt;anonymous&gt;"
      lastModifiedTimestamp="2013-05-14T13:43:09.708-04:00"
```

```
name="test.txt" uuid="05778de3-be85-4696-b5dc-d889a27f1f6e" />
  </s-ramp:artifact>
</atom:entry>
```

4.8.2.3. Publish a Non-Document Style Artifact

To publish a non-document style artifact, you require an Atom Entry that contains an s-ramp:artifact child element, to be POSTed to the appropriate endpoint. The appropriate endpoint is based on the desired artifact model and type.

```
 /s-ramp/{model}/{type}
```

Table 4.14. Post Method Response for Non-Document Style Artifact

HTTP Method	Request	Response
POST	Atom Entry	Atom Entry

Example Request:

```
POST /s-ramp/ext/MyArtifact HTTP/1.1

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:entry xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
  xmlns:xlink="http://www.w3.org/1999/xlink" s-ramp:derived="false">
  <atom:title>Example Artifact</atom:title>
  <s-ramp:artifact>
    <s-ramp:ExtendedArtifactType extendedType="MyArtifact"
      artifactType="ExtendedArtifactType" name="My Artifact One" />
  </s-ramp:artifact>
</atom:entry>
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<atom:entry xmlns:s-ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-
v1.0" xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:atom="http://www.w3.org/2005/Atom" s-ramp:derived="false" s-
ramp:extendedType="MavenPom">
  <atom:title>pom.xml</atom:title>
  <atom:link href="http://example.org/s-ramp/ext/MavenPom/5f4cbf1e-cafb-
4479-8867-fc5df5f21867/media"
    rel="alternate" type="application/xml" />
  <atom:link href="http://example.org/s-ramp/ext/MavenPom/5f4cbf1e-cafb-
4479-8867-fc5df5f21867" rel="self"
    type="application/atom+xml;type=&amp;quot;entry&amp;quot;" />
  <atom:link href="http://example.org/s-ramp/ext/MavenPom/5f4cbf1e-cafb-
```

```
4479-8867-fc5df5f21867/media"
    rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;" />
  <atom:link href="http://example.org/s-ramp/ext/MavenPom/5f4cbf1e-cafb-
4479-8867-fc5df5f21867" rel="edit"
    type="application/atom+xml;type=&quot;entry&quot;" />
  <atom:category label="Extended Document" scheme="x-s-ramp:2010:type"
term="MavenPom" />
  <atom:category label="Extended Document" scheme="x-s-ramp:2010:model"
term="ext" />
  <atom:updated>2013-05-14T13:49:20.645-04:00</atom:updated>
  <atom:id>5f4cbf1e-cafb-4479-8867-fc5df5f21867</atom:id>
  <atom:published>2013-05-14T13:49:20.645-04:00</atom:published>
  <atom:author>
    <atom:name>ewittman</atom:name>
  </atom:author>
  <atom:content type="application/xml"
    src="http://example.org/s-ramp/ext/MavenPom/5f4cbf1e-cafb-4479-8867-
fc5df5f21867/media" />
  <s-ramp:artifact>
    <s-ramp:ExtendedDocument extendedType="MavenPom"
contentType="application/xml"
      contentSize="4748" artifactType="ExtendedDocument" name="pom.xml"
createdBy="&lt;anonymous&gt;"
      uuid="5f4cbf1e-cafb-4479-8867-fc5df5f21867" createdTimestamp="2013-
05-14T13:49:20.645-04:00"
      lastModifiedTimestamp="2013-05-14T13:49:20.645-04:00"
lastModifiedBy="&lt;anonymous&gt;"
      s-ramp:contentType="application/xml" s-ramp:contentSize="4748" />
  </s-ramp:artifact>
</atom:entry>
```

4.8.2.4. Publish a Document Style Artifact with Meta-Data

To publish an artifact and update its meta-data in a single request, you can POST a multipart or related request to the server at the appropriate endpoint. The first part in the request must be an Atom Entry containing the meta-data being set, while the second part must be the binary content. The appropriate endpoint is based on the desired artifact model and type.

```
/s-ramp/{model}/{type}
```

Table 4.15. Post Method Response for Document Style Artifact with Meta-Data

HTTP Method	Request	Response
POST	Multipart/Related	Atom Entry

Example Request:

```
POST /s-ramp/core/Document HTTP/1.1
Content-Type: multipart/related;boundary="===============1605871705==";
type="application/atom+xml"
MIME-Version: 1.0
```

```
--===============1605871705==
Content-Type: application/atom+xml; charset="utf-8"
MIME-Version: 1.0

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom"
       xmlns:s-ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0">
  <title type="text">myfile.txt</title>
  <summary type="text">The description of my text file.</summary>
  <category term="Document" label="Document"
            scheme="urn:x-s-ramp:2013urn:x-s-ramp:2013:type" />
  <s-ramp:artifact xmlns:s-ramp="http://docs.oasis-open.org/s-ramp/ns/s-
ramp-v1.0"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <s-ramp:Document name="myfile.txt" version="1.0"
                     description="The description of my text file." >
      <s-ramp:classifiedBy>
        http://example.org/ontologies/regions.owl/Maine
      </s-ramp:classifiedBy>
      <s-ramp:property>
        <propertyName>foo</propertyName>
        <propertyValue>pity him</propertyValue>
      </s-ramp:property>
    </s-ramp:Document>
  </s-ramp:artifact>
</entry>
--===============1605871705==
Content-Type: application/xml
MIME-Version: 1.0

This is a simple text document, uploaded as an artifact
into S-RAMP.
--===============1605871705==--
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:entry xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
  xmlns:xlink="http://www.w3.org/1999/xlink" s-ramp:derived="false">
  <atom:title>test.txt</atom:title>
  <atom:link
    href="http://example.org/s-ramp/core/Document/05778de3-be85-4696-b5dc-
d889a27f1f6e/media"
    rel="alternate" type="text/plain" />
  <atom:link href="http://example.org/s-ramp/core/Document/05778de3-be85-
4696-b5dc-d889a27f1f6e"
    rel="self" type="application/atom+xml;type=&amp;quot;entry&amp;quot;" />
  <atom:link
    href="http://example.org/s-ramp/core/Document/05778de3-be85-4696-b5dc-
d889a27f1f6e/media"
```

```
    rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;" />
  <atom:link href="http://example.org/s-ramp/core/Document/05778de3-be85-
4696-b5dc-d889a27f1f6e"
    rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
  <atom:category label="Document" scheme="x-s-ramp:2010:type"
term="Document" />
  <atom:category label="Document" scheme="x-s-ramp:2010:model" term="core"
/>
  <atom:updated>2013-05-14T13:43:09.708-04:00</atom:updated>
  <atom:id>05778de3-be85-4696-b5dc-d889a27f1f6e</atom:id>
  <atom:published>2013-05-14T13:43:09.708-04:00</atom:published>
  <atom:author>
    <atom:name>ewittman</atom:name>
  </atom:author>
  <atom:content
    src="http://example.org/s-ramp/core/Document/05778de3-be85-4696-b5dc-
d889a27f1f6e/media"
    type="text" />
  <s-ramp:artifact>
    <s-ramp:Document artifactType="Document" contentSize="69"
contentType="text/plain"
      name="myfile.txt" uuid="05778de3-be85-4696-b5dc-d889a27f1f6e">
      description="The description of my text file." version="1.0"
      createdBy="&lt;anonymous&gt;" createdTimestamp="2013-05-
14T13:43:09.708-04:00"
      lastModifiedBy="&lt;anonymous&gt;" lastModifiedTimestamp="2013-05-
14T13:43:09.708-04:00"
      <s-ramp:classifiedBy>
        http://example.org/ontologies/regions.owl/Maine
      </s-ramp:classifiedBy>
      <s-ramp:property>
        <propertyName>foo</propertyName>
        <propertyValue>pity him</propertyValue>
      </s-ramp:property>
    </s-ramp:Document>
  </s-ramp:artifact>
</atom:entry>
```

4.8.2.5. Update Artifact API

The Update Artifact API (**`/s-ramp/{model}/{type}/{uuid}`**) updates an artifact's meta data.This endpoint is used to update a single artifact's meta data, including core properties, custom properties, classifiers, and relationships. Typically the client first retrieves the artifact (for example, by invoking the Get Artifact endpoint), makes changes to the artifact, and then issues a PUT request to the Update Artifact endpoint.

Table 4.16. PUT Method Request for Update Artifact

HTTP Method	Request	Response
PUT	Atom Entry	N/A

Example Request:

```
PUT /s-ramp/core/Document/098da465-2eae-49b7-8857-eb447f03ac02 HTTP/1.1

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<atom:entry xmlns:s-ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-
v1.0" xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:atom="http://www.w3.org/2005/Atom">
  <atom:title>pom.xml</atom:title>
  <atom:updated>2013-05-15T08:12:01.985-04:00</atom:updated>
  <atom:id>098da465-2eae-49b7-8857-eb447f03ac02</atom:id>
  <atom:published>2013-05-15T08:12:01.985-04:00</atom:published>
  <atom:author>
    <atom:name>ewittman</atom:name>
  </atom:author>
  <atom:summary>Sample description of my document.</atom:summary>
  <s-ramp:artifact>
    <s-ramp:Document contentType="text/plain" contentSize="4748"
artifactType="Document"
      name="myfile.txt" description="Sample description of my document."
createdBy="ewittman"
      uuid="098da465-2eae-49b7-8857-eb447f03ac02" createdTimestamp="2013-
05-15T08:12:01.985-04:00"
      lastModifiedTimestamp="2013-05-15T08:12:01.985-04:00"
lastModifiedBy="ewittman">
      <s-ramp:property>
        <s-ramp:propertyName>foo</s-ramp:propertyName>
        <s-ramp:propertyValue>bar</s-ramp:propertyValue>
      </s-ramp:property>
    </s-ramp:Document>
  </s-ramp:artifact>
</atom:entry>
```

4.8.2.6. Update Artifact Content API

The Update Artifact Content API (**`/s-ramp/{model}/{type}/{uuid}/media`**) updates an artifact's content.

This endpoint is used to update a single artifact's content, regardless if the artifact is a text document or some sort of binary. The body of the request must be the new binary content of the artifact.

Table 4.17. PUT Method Request for Update Artifact Content

HTTP Method	Request	Response
PUT	Binary Content	N/A

Example Response:

```
PUT /s-ramp/core/Document/0f6f9b6b-9952-4059-ab70-7ee3442ddcf0/media
HTTP/1.1
```

4.8.2.7. Get Artifact API

The Get Artifact API (`/s-ramp/{model}/{type}/{uuid}`) retrieves an artifact's meta data.

This endpoint is used to retrieve the full meta-data for a single artifact in the repository. The data is returned wrapped up in an Atom Entry document. The Atom Entry contains an extended XML element containing the S-RAMP artifact data.

Table 4.18. GET Method Response for Get Artifact

HTTP Method	Request	Response
Get	N/A	Atom Entry (full)

Example Request:

```
PUT /s-ramp/xsd/ComplexTypeDeclaration/0104e848-fe91-4d93-a307-
fb69ec9fd638 HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:entry xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
xmlns:xlink="http://www.w3.org/1999/xlink" s-ramp:derived="true">
<atom:title>submitOrderResponseType</atom:title>
<atom:link href="http://localhost:8080/s-ramp-server/s-
ramp/xsd/ComplexTypeDeclaration/0104e848-fe91-4d93-a307-fb69ec9fd638"
rel="self" type="application/atom+xml;type=&amp;quot;entry&amp;quot;"/>
<atom:link href="http://localhost:8080/s-ramp-server/s-
ramp/xsd/ComplexTypeDeclaration/0104e848-fe91-4d93-a307-
fb69ec9fd638/media" rel="edit-media"
type="application/atom+xml;type=&amp;quot;entry&amp;quot;"/>
<atom:link href="http://localhost:8080/s-ramp-server/s-
ramp/xsd/ComplexTypeDeclaration/0104e848-fe91-4d93-a307-fb69ec9fd638"
rel="edit" type="application/atom+xml;type=&amp;quot;entry&amp;quot;"/>
<atom:category label="XML Schema Complex Type Declaration" scheme="x-s-
ramp:2010:type" term="ComplexTypeDeclaration"/>
<atom:category label="XML Schema Complex Type Declaration" scheme="x-s-
ramp:2010:model" term="xsd"/>
<atom:updated>2013-07-22T12:19:23.554-04:00</atom:updated>
<atom:id>0104e848-fe91-4d93-a307-fb69ec9fd638</atom:id>
<atom:published>2013-07-22T12:19:22.630-04:00</atom:published>
<atom:author>
<atom:name>eric</atom:name>
</atom:author>
<s-ramp:artifact>
<s-ramp:ComplexTypeDeclaration artifactType="ComplexTypeDeclaration"
createdBy="eric" createdTimestamp="2013-07-22T12:19:22.630-04:00"
lastModifiedBy="eric" lastModifiedTimestamp="2013-07-22T12:19:23.554-
04:00" name="submitOrderResponseType" namespace="urn:switchyard-
quickstart-demo:multiapp:1.0" uuid="0104e848-fe91-4d93-a307-fb69ec9fd638">
```

```
<s-ramp:relatedDocument artifactType="XsdDocument">fe7b72ec-5ad9-436c-
b7aa-0391da5cc972</s-ramp:relatedDocument>
</s-ramp:ComplexTypeDeclaration>
</s-ramp:artifact>
</atom:entry>
```

4.8.2.8. Get Artifact Content API

The Get Artifact Content API (**`/s-ramp/{model}/{type}/{uuid}/media`**) retrieves an artifact's content.

This endpoint is used to retrieve the full content of a single artifact in the repository. If the artifact is not a Document style artifact, this call fails. Otherwise, it returns the full artifact content. For example, if the artifact is a PdfDocument, then this call returns the PDF file.

Table 4.19. Get Method Response for Get Artifact Content

HTTP Method	Request	Response
Get	N/A	Binary artifact content

Example Request:

```
GET /s-ramp/core/Document/0f6f9b6b-9952-4059-ab70-7ee3442ddcf0/media
HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK
```

4.8.2.9. Delete Artifact API

The Delete Artifact API (**`/s-ramp/{model}/{type}/{uuid}`**) deletes an artifact.

This endpoint is used to delete a single artifact from the repository. If the artifact does not exist or is a derived artifact, then this fails. This may also fail if other artifacts have relationships with it. Otherwise, this artifact and all of it's derived artifacts are deleted.

Table 4.20. DELETE Method Request and Response

HTTP Method	Request	Response
DELETE	N/A	N/A

Example Request:

```
DELETE /s-ramp/core/Document/0f6f9b6b-9952-4059-ab70-7ee3442ddcf0 HTTP/1.1
```

4.8.2.10. Get Artifact Feed (by Model) API

The Get Artifact Feed (by model) API (**`/s-ramp/{model}`**)retrieves an Atom feed of all artifacts in a

given model.

This endpoint is used to retrieve an Atom feed of all artifacts in a single S-RAMP model. The feed contains Atom summary Entries, one for each artifact in the feed. Standard paging options apply.

Table 4.21. GET Method Response for Get Artifact Feed by Model

HTTP Method	Request	Response
GET	N/A	Atom Feed

Example Request:

```
GET /s-ramp/core HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
  s-ramp:itemsPerPage="100" s-ramp:provider="JBoss Overlord" s-
ramp:startIndex="0" s-ramp:totalResults="5">
  <atom:title>S-RAMP Feed</atom:title>
  <atom:subtitle>Ad Hoc query feed</atom:subtitle>
  <atom:updated>2013-07-22T12:50:16.605-04:00</atom:updated>
  <atom:id>1647967f-a6f4-4e9c-82d3-ac422fb152f3</atom:id>
  <atom:author>
    <atom:name>anonymous</atom:name>
  </atom:author>
  <atom:entry s-ramp:derived="false">
    <atom:title>sramp.sh</atom:title>
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0/media"
      rel="alternate" type="application/x-sh" />
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0"
      rel="self" type="application/atom+xml;type=&amp;quot;entry&amp;quot;" />
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0/media"
      rel="edit-media" type="application/atom+xml;type=&amp;quot;entry&amp;quot;"
/>
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0"
      rel="edit" type="application/atom+xml;type=&amp;quot;entry&amp;quot;" />
    <atom:category label="Document" scheme="x-s-ramp:2010:type"
term="Document" />
    <atom:category label="Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:22:01.953-04:00</atom:updated>
    <atom:id>0f6f9b6b-9952-4059-ab70-7ee3442ddcf0</atom:id>
    <atom:published>2013-07-22T12:21:49.499-04:00</atom:published>
    <atom:author>
```

```
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/Document/0f6f9b6b-9952-4059-ab70-7ee3442ddcf0/media"
      type="application/x-sh" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>beans.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:27.660-04:00</atom:updated>
    <atom:id>20474032-9536-4cef-812c-4fea432fdebd</atom:id>
    <atom:published>2013-07-22T12:19:27.644-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>forge.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:25.576-04:00</atom:updated>
    <atom:id>2c21a9d3-0d09-41d8-8783-f3e795d8690d</atom:id>
```

```
    <atom:published>2013-07-22T12:19:25.555-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>route.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:25.602-04:00</atom:updated>
    <atom:id>5b653bfe-4f58-451e-b738-394e61c0c5f9</atom:id>
    <atom:published>2013-07-22T12:19:25.577-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>beans.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
```

```
    <atom:updated>2013-07-22T12:19:21.498-04:00</atom:updated>
    <atom:id>a3f9d4d7-0f95-4219-85f6-84df445ef270</atom:id>
    <atom:published>2013-07-22T12:19:21.376-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      type="application/xml" />
  </atom:entry>
</atom:feed>
```

4.8.2.11. Get Artifact Feed (by Type) API

The Get Artifact Feed (by type) API (**`/s-ramp/{model}/{type}`**)retrieves an Atom feed of all artifacts of a specific type.

This endpoint is used to retrieve an Atom feed of all artifacts of a specific S-RAMP type. The feed contains Atom summary Entries, one for each artifact in the feed. Standard paging options (as query params) apply.

Table 4.22. GET Method Response for Get Artifact Feed by Model

HTTP Method	Request	Response
GET	N/A	Atom Feed

Example Request:

```
GET /s-ramp/core/Document HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
  s-ramp:itemsPerPage="100" s-ramp:provider="JBoss Overlord" s-
ramp:startIndex="0" s-ramp:totalResults="5">
  <atom:title>S-RAMP Feed</atom:title>
  <atom:subtitle>Ad Hoc query feed</atom:subtitle>
  <atom:updated>2013-07-22T12:50:16.605-04:00</atom:updated>
  <atom:id>1647967f-a6f4-4e9c-82d3-ac422fb152f3</atom:id>
  <atom:author>
    <atom:name>anonymous</atom:name>
  </atom:author>
  <atom:entry s-ramp:derived="false">
    <atom:title>sramp.sh</atom:title>
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0/media"
      rel="alternate" type="application/x-sh" />
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0"
```

```
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="Document" scheme="x-s-ramp:2010:type"
term="Document" />
    <atom:category label="Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:22:01.953-04:00</atom:updated>
    <atom:id>0f6f9b6b-9952-4059-ab70-7ee3442ddcf0</atom:id>
    <atom:published>2013-07-22T12:21:49.499-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/Document/0f6f9b6b-9952-4059-ab70-7ee3442ddcf0/media"
      type="application/x-sh" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>beans.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:27.660-04:00</atom:updated>
    <atom:id>20474032-9536-4cef-812c-4fea432fdebd</atom:id>
    <atom:published>2013-07-22T12:19:27.644-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>forge.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      rel="alternate" type="application/xml" />
```

```
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:25.576-04:00</atom:updated>
    <atom:id>2c21a9d3-0d09-41d8-8783-f3e795d8690d</atom:id>
    <atom:published>2013-07-22T12:19:25.555-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>route.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:25.602-04:00</atom:updated>
    <atom:id>5b653bfe-4f58-451e-b738-394e61c0c5f9</atom:id>
    <atom:published>2013-07-22T12:19:25.577-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>beans.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
```

```
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:21.498-04:00</atom:updated>
    <atom:id>a3f9d4d7-0f95-4219-85f6-84df445ef270</atom:id>
    <atom:published>2013-07-22T12:19:21.376-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      type="application/xml" />
  </atom:entry>
</atom:feed>
```

4.8.2.12. Query API

The Query API (**/s-ramp**) performs an S-RAMP query and returns an Atom feed containing the matching artifacts.

This endpoint is used to perform an S-RAMP query and return an Atom Feed of the results. Ordering and paging is supported. The query and other parameters are passed as query params in the request. The feed contains Atom summary Entries, one for each artifact in the feed.

Table 4.23. GET and POST Methods Request and Response for Query

HTTP Method	Request	Response
GET	N/A	Atom Feed
POST	FormData	Atom Feed

Example Requests:

```
GET /s-ramp?query=/s-ramp/core/Document HTTP/1.1
```

```
POST /s-ramp HTTP/1.1

--ac709f11-bfc5-48df-8918-e58b254d0490
Content-Disposition: form-data; name="query"
```

```
Content-Type: text/plain

core/Document
--ac709f11-bfc5-48df-8918-e58b254d0490
Content-Disposition: form-data; name="startIndex"
Content-Type: text/plain

0
--ac709f11-bfc5-48df-8918-e58b254d0490
Content-Disposition: form-data; name="count"
Content-Type: text/plain

100
--ac709f11-bfc5-48df-8918-e58b254d0490
Content-Disposition: form-data; name="orderBy"
Content-Type: text/plain

uuid
--ac709f11-bfc5-48df-8918-e58b254d0490
Content-Disposition: form-data; name="ascending"
Content-Type: text/plain

true
--ac709f11-bfc5-48df-8918-e58b254d0490--
```

Example Responses:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
  s-ramp:itemsPerPage="100" s-ramp:provider="JBoss Overlord" s-
ramp:startIndex="0" s-ramp:totalResults="5">
  <atom:title>S-RAMP Feed</atom:title>
  <atom:subtitle>Ad Hoc query feed</atom:subtitle>
  <atom:updated>2013-07-22T12:50:16.605-04:00</atom:updated>
  <atom:id>1647967f-a6f4-4e9c-82d3-ac422fb152f3</atom:id>
  <atom:author>
    <atom:name>anonymous</atom:name>
  </atom:author>
  <atom:entry s-ramp:derived="false">
    <atom:title>sramp.sh</atom:title>
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0/media"
      rel="alternate" type="application/x-sh" />
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
```

```
    <atom:category label="Document" scheme="x-s-ramp:2010:type"
term="Document" />
    <atom:category label="Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:22:01.953-04:00</atom:updated>
    <atom:id>0f6f9b6b-9952-4059-ab70-7ee3442ddcf0</atom:id>
    <atom:published>2013-07-22T12:21:49.499-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/Document/0f6f9b6b-9952-4059-ab70-7ee3442ddcf0/media"
      type="application/x-sh" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>beans.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:27.660-04:00</atom:updated>
    <atom:id>20474032-9536-4cef-812c-4fea432fdebd</atom:id>
    <atom:published>2013-07-22T12:19:27.644-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>forge.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
```

```
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:25.576-04:00</atom:updated>
    <atom:id>2c21a9d3-0d09-41d8-8783-f3e795d8690d</atom:id>
    <atom:published>2013-07-22T12:19:25.555-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>route.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:25.602-04:00</atom:updated>
    <atom:id>5b653bfe-4f58-451e-b738-394e61c0c5f9</atom:id>
    <atom:published>2013-07-22T12:19:25.577-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>beans.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
```

```
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270"
      rel="edit" type="application/atom+xml;type=&amp;quot;entry&amp;quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:21.498-04:00</atom:updated>
    <atom:id>a3f9d4d7-0f95-4219-85f6-84df445ef270</atom:id>
    <atom:published>2013-07-22T12:19:21.376-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      type="application/xml" />
  </atom:entry>
</atom:feed>
```

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
  s-ramp:itemsPerPage="100" s-ramp:provider="JBoss Overlord" s-
ramp:startIndex="0" s-ramp:totalResults="5">
  <atom:title>S-RAMP Feed</atom:title>
  <atom:subtitle>Ad Hoc query feed</atom:subtitle>
  <atom:updated>2013-07-22T12:50:16.605-04:00</atom:updated>
  <atom:id>1647967f-a6f4-4e9c-82d3-ac422fb152f3</atom:id>
  <atom:author>
    <atom:name>anonymous</atom:name>
  </atom:author>
  <atom:entry s-ramp:derived="false">
    <atom:title>sramp.sh</atom:title>
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0/media"
      rel="alternate" type="application/x-sh" />
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0"
      rel="self" type="application/atom+xml;type=&amp;quot;entry&amp;quot;" />
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0/media"
      rel="edit-media" type="application/atom+xml;type=&amp;quot;entry&amp;quot;"
/>
    <atom:link href="http://localhost:8080/s-ramp/core/Document/0f6f9b6b-
9952-4059-ab70-7ee3442ddcf0"
      rel="edit" type="application/atom+xml;type=&amp;quot;entry&amp;quot;" />
    <atom:category label="Document" scheme="x-s-ramp:2010:type"
term="Document" />
    <atom:category label="Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:22:01.953-04:00</atom:updated>
    <atom:id>0f6f9b6b-9952-4059-ab70-7ee3442ddcf0</atom:id>
    <atom:published>2013-07-22T12:21:49.499-04:00</atom:published>
```

```
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/Document/0f6f9b6b-9952-4059-ab70-7ee3442ddcf0/media"
      type="application/x-sh" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>beans.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:27.660-04:00</atom:updated>
    <atom:id>20474032-9536-4cef-812c-4fea432fdebd</atom:id>
    <atom:published>2013-07-22T12:19:27.644-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/20474032-9536-4cef-812c-4fea432fdebd/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>forge.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:25.576-04:00</atom:updated>
```

```
    <atom:id>2c21a9d3-0d09-41d8-8783-f3e795d8690d</atom:id>
    <atom:published>2013-07-22T12:19:25.555-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/2c21a9d3-0d09-41d8-8783-f3e795d8690d/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>route.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
term="core" />
    <atom:updated>2013-07-22T12:19:25.602-04:00</atom:updated>
    <atom:id>5b653bfe-4f58-451e-b738-394e61c0c5f9</atom:id>
    <atom:published>2013-07-22T12:19:25.577-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/5b653bfe-4f58-451e-b738-394e61c0c5f9/media"
      type="application/xml" />
  </atom:entry>
  <atom:entry s-ramp:derived="false">
    <atom:title>beans.xml</atom:title>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      rel="alternate" type="application/xml" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270"
      rel="self" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      rel="edit-media" type="application/atom+xml;type=&quot;entry&quot;"
/>
    <atom:link href="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270"
      rel="edit" type="application/atom+xml;type=&quot;entry&quot;" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:type"
term="XmlDocument" />
    <atom:category label="XML Document" scheme="x-s-ramp:2010:model"
```

```
term="core" />
    <atom:updated>2013-07-22T12:19:21.498-04:00</atom:updated>
    <atom:id>a3f9d4d7-0f95-4219-85f6-84df445ef270</atom:id>
    <atom:published>2013-07-22T12:19:21.376-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:content src="http://localhost:8080/s-
ramp/core/XmlDocument/a3f9d4d7-0f95-4219-85f6-84df445ef270/media"
      type="application/xml" />
  </atom:entry>
</atom:feed>
```

4.8.2.13. Batch Processing API

The Batch Processing API (**`/s-ramp`**) performs an S-RAMP query and returns an Atom feed containing the matching artifacts.

This endpoint is used to perform an S-RAMP query and return an Atom Feed of the results. Ordering and paging is supported. The query and other parameters are passed as form data params in the request body. The feed contains Atom summary Entries, one for each artifact in the feed.

Table 4.24. POST Method Request and Response for Batch Processing

HTTP Method	Request	Response
POST	multipart/form-data	Atom Feed

Example Request:

```
POST XX_TBD_XX HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK
```

4.8.2.14. Add Ontology API

The Add Ontology API (**`/s-ramp/ontology`**) adds a new ontology (*.owl file) to the repository. This allows artifacts to be classified using the classes defined in the ontology.

This endpoint is used to add an ontology to the repository. The body of the request must be the OWL Lite formatted ontology (see the S-RAMP specification for more details). The response is an Atom Entry containing meta-data about the ontology, most importantly the UUID of the ontology (which can be later used to update or delete it).

Table 4.25. POST Method Request Response for Add Ontology

HTTP Method	Request	Response
POST	application/rdf+xml	Atom Feed

Example Request:

```
POST /s-ramp/ontology HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xml:base="http://www.example.org/sample-ontology-1.owl">

    <owl:Ontology rdf:ID="SampleOntology1">
        <rdfs:label>Sample Ontology 1</rdfs:label>
        <rdfs:comment>A sample ontology.</rdfs:comment>
    </owl:Ontology>

    <owl:Class rdf:ID="All">
        <rdfs:label>All</rdfs:label>
    </owl:Class>

    <owl:Class rdf:ID="King">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#All" />
        <rdfs:label>King</rdfs:label>
        <rdfs:comment>Feudal ruler.</rdfs:comment>
    </owl:Class>
    <owl:Class rdf:ID="Imperator">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#All" />
        <rdfs:label>Imperator</rdfs:label>
        <rdfs:comment>Roman ruler.</rdfs:comment>
    </owl:Class>

    <owl:Class rdf:ID="Baron">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#King" />
        <rdfs:label>Baron</rdfs:label>
    </owl:Class>
    <owl:Class rdf:ID="Rex">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#Imperator" />
        <rdfs:label>Imperator</rdfs:label>
    </owl:Class>

    <owl:Class rdf:ID="Knight">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#Baron" />
        <rdfs:label>Knight</rdfs:label>
    </owl:Class>
    <owl:Class rdf:ID="Dux">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#Rex" />
        <rdfs:label>Dux</rdfs:label>
    </owl:Class>

</rdf:RDF>
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:entry xmlns="http://www.w3.org/2000/01/rdf-schema#"
xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:ns2="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ns3="http://www.w3.org/2002/07/owl#">
  <atom:title>Sample Ontology 1</atom:title>
  <atom:id>e8fe74f3-c9c3-4678-ba76-d71158141ddd</atom:id>
  <atom:author />
  <atom:summary>A sample ontology.</atom:summary>
  <ns2:RDF xml:base="http://www.example.org/sample-ontology-1.owl">
    <ns3:Ontology ns2:ID="SampleOntology1">
      <label>Sample Ontology 1</label>
      <comment>A sample ontology.</comment>
    </ns3:Ontology>
    <ns3:Class ns2:ID="All">
      <label>All</label>
    </ns3:Class>
    <ns3:Class ns2:ID="King">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
1.owl#All" />
      <label>King</label>
      <comment>Feudal ruler.</comment>
    </ns3:Class>
    <ns3:Class ns2:ID="Imperator">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
1.owl#All" />
      <label>Imperator</label>
      <comment>Roman ruler.</comment>
    </ns3:Class>
    <ns3:Class ns2:ID="Baron">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
1.owl#King" />
      <label>Baron</label>
    </ns3:Class>
    <ns3:Class ns2:ID="Knight">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
1.owl#Baron" />
      <label>Knight</label>
    </ns3:Class>
    <ns3:Class ns2:ID="Rex">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
1.owl#Imperator" />
      <label>Imperator</label>
    </ns3:Class>
    <ns3:Class ns2:ID="Dux">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
1.owl#Rex" />
      <label>Dux</label>
    </ns3:Class>
  </ns2:RDF>
</atom:entry>
```

4.8.2.15. List Ontologies API

The List Ontologies API (**`/s-ramp/ontology`**) retrieves all ontologies currently known to the repository as an Atom feed.

This endpoint is used to retrieve all ontologies known to the repository as an Atom Feed of Entries, with one Entry for each ontology. You can subsequently retrieve full information about the ontology by calling the Get Ontology endpoint.

Table 4.26. GET Method Response for List Ontologies

HTTP Method	Request	Response
GET	N/A	Atom Feed

Example Request:

```
GET /s-ramp/ontology HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom">
  <atom:title>S-RAMP ontology feed</atom:title>
  <atom:updated>2013-07-23T10:58:40.356-04:00</atom:updated>
  <atom:entry>
    <atom:title>Sample Ontology 1</atom:title>
    <atom:updated>2013-07-23T10:56:50.410-04:00</atom:updated>
    <atom:id>e8fe74f3-c9c3-4678-ba76-d71158141ddd</atom:id>
    <atom:published>2013-07-23T10:56:50.410-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:source xml:base="http://www.example.org/sample-ontology-1.owl">
      <atom:id>SampleOntology1</atom:id>
    </atom:source>
    <atom:summary>A sample ontology.</atom:summary>
  </atom:entry>
  <atom:entry>
    <atom:title>Animal Kingdom</atom:title>
    <atom:updated>2013-07-23T10:58:37.737-04:00</atom:updated>
    <atom:id>fd0e5210-2567-409f-8df0-f851e1ce630d</atom:id>
    <atom:published>2013-07-23T10:58:37.737-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:source xml:base="http://www.example.org/sample-ontology-2.owl">
      <atom:id>AnimalKingdom</atom:id>
    </atom:source>
    <atom:summary>Animal Kingdom Ontology</atom:summary>
  </atom:entry>
</atom:feed>
```

4.8.2.16. Update Ontology API

The Update Ontologies API (**`/s-ramp/ontology/{uuid}`**) updates an existing ontology by its UUID.

This endpoint is used to update a single ontology in the repository. The request body must be a new version of the ontology in OWL Lite RDF format. Note that, this might fail if the ontology changes in an incompatible way (for example, a class is removed that is currently in use).

Table 4.27. PUT Method Request for Update Ontology

HTTP Method	Request	Response
PUT	application/rdf+xml	N/A

Example Request:

```
PUT /s-ramp/ontology/fd0e5210-2567-409f-8df0-f851e1ce630d HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xml:base="http://www.example.org/sample-ontology-1.owl">

    <owl:Ontology rdf:ID="SampleOntology1">
        <rdfs:label>Sample Ontology 1</rdfs:label>
        <rdfs:comment>A sample ontology.</rdfs:comment>
    </owl:Ontology>

    <owl:Class rdf:ID="All">
        <rdfs:label>All</rdfs:label>
    </owl:Class>

    <owl:Class rdf:ID="King">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#All" />
        <rdfs:label>King</rdfs:label>
        <rdfs:comment>Feudal ruler.</rdfs:comment>
    </owl:Class>
    <owl:Class rdf:ID="Imperator">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#All" />
        <rdfs:label>Imperator</rdfs:label>
        <rdfs:comment>Roman ruler.</rdfs:comment>
    </owl:Class>

    <owl:Class rdf:ID="Baron">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#King" />
        <rdfs:label>Baron</rdfs:label>
    </owl:Class>
    <owl:Class rdf:ID="Rex">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#Imperator" />
```

```
        <rdfs:label>Imperator</rdfs:label>
    </owl:Class>

    <owl:Class rdf:ID="Knight">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#Baron" />
        <rdfs:label>Knight</rdfs:label>
    </owl:Class>
    <owl:Class rdf:ID="Dux">
        <rdfs:subClassOf rdf:resource="http://www.example.org/sample-
ontology-1.owl#Rex" />
        <rdfs:label>Dux</rdfs:label>
    </owl:Class>

</rdf:RDF>
```

Example Response:

```
HTTP/1.1 200 OK
```

4.8.2.17. Get Ontology API

Get Ontology API (**`/s-ramp/ontology/{uuid}`**) returns the OWL representation of an ontology (wrapped in an Atom Entry).

This endpoint is used to get the full ontology (by its UUID) in OWL Lite (RDF) format, wrapped in an Atom Entry. The response body is an Atom Entry with a single extension element that is the ontology RDF. This fails if no ontology exists with the given UUID.

Table 4.28. GET Method Response for Update Ontology

HTTP Method	Request	Response
GET	N/A	Atom Entry

Example Request:

```
GET /s-ramp/ontology/fd0e5210-2567-409f-8df0-f851e1ce630d HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:entry xmlns="http://www.w3.org/2000/01/rdf-schema#"
xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:ns2="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ns3="http://www.w3.org/2002/07/owl#">
  <atom:title>Animal Kingdom</atom:title>
  <atom:updated>2013-07-23T10:58:37.737-04:00</atom:updated>
  <atom:id>fd0e5210-2567-409f-8df0-f851e1ce630d</atom:id>
  <atom:published>2013-07-23T10:58:37.737-04:00</atom:published>
  <atom:author>
```

```
    <atom:name>eric</atom:name>
  </atom:author>
  <atom:summary>Animal Kingdom Ontology</atom:summary>
  <ns2:RDF xml:base="http://www.example.org/sample-ontology-2.owl">
    <ns3:Ontology ns2:ID="AnimalKingdom">
      <label>Animal Kingdom</label>
      <comment>Animal Kingdom Ontology</comment>
    </ns3:Ontology>
    <ns3:Class ns2:ID="Animal">
      <label>Animal</label>
      <comment>All animals.</comment>
    </ns3:Class>
    <ns3:Class ns2:ID="UnicellularAnimal">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
2.owl#Animal" />
      <label>Unicellular Animal</label>
      <comment>Single-celled animal.</comment>
    </ns3:Class>
    <ns3:Class ns2:ID="MulticellularAnimal">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
2.owl#Animal" />
      <label>Multicellular Animal</label>
      <comment>Multi-celled animal.</comment>
    </ns3:Class>
    <ns3:Class ns2:ID="Protozoa">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
2.owl#UnicellularAnimal" />
      <label>Protozoa</label>
    </ns3:Class>
    <ns3:Class ns2:ID="Metazoa">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
2.owl#MulticellularAnimal" />
      <label>Metazoa</label>
    </ns3:Class>
    <ns3:Class ns2:ID="Invertebrate">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
2.owl#Metazoa" />
      <label>Invertebrate</label>
    </ns3:Class>
    <ns3:Class ns2:ID="Vertebrate">
      <subClassOf ns2:resource="http://www.example.org/sample-ontology-
2.owl#Metazoa" />
      <label>Vertebrate</label>
    </ns3:Class>
  </ns2:RDF>
</atom:entry>
```

4.8.2.18. Delete Ontology API

The Delete Ontology API (**`/s-ramp/ontology/{uuid}`**) deletes an ontology from the repository.

This endpoint is used to delete a single ontology from the repository. This may fail if the ontology is currently in-use (at least one artifact is classified by at least one class defined by the ontology).

Table 4.29. DELETE Method Request and Response for Delete Ontology

HTTP Method	Request	Response
DELETE	N/A	N/A

Example Request:

```
DELETE /s-ramp/ontology/fd0e5210-2567-409f-8df0-f851e1ce630d HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK
```

4.8.2.19. Get Artifact Audit History API

The Get Artifact Audit History API (**`/s-ramp/audit/artifact/{artifactUuid}`**) retrieves an Atom feed containing all of the audit entries for a single artifact.

This endpoint is used to get a feed of the audit history of a single artifact. The request URL can include standard paging parameters. The response is an Atom Feed where each Entry in the feed represents a single audit event in the history of the artifact. A follow up call must be made to the Get Artifact Audit Entry endpoint in order to retrieve full detail information about the audit event. This call may fail if no artifact exits with the given UUID.

Table 4.30. GET Method Response for Get Artifact Audit History

HTTP Method	Request	Response
GET	N/A	Atom Feed

Example Request:

```
GET /s-ramp/audit/artifact/b086c558-58d6-4837-bb38-6c3da760ae80 HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
  s-ramp:itemsPerPage="100" s-ramp:provider="JBoss Overlord" s-
ramp:startIndex="0" s-ramp:totalResults="2">
  <atom:title>S-RAMP Audit Feed</atom:title>
  <atom:subtitle>All Audit Entries for Artifact</atom:subtitle>
  <atom:updated>2013-07-23T11:14:07.189-04:00</atom:updated>
  <atom:id>bff03dd5-e55c-4528-b1aa-ee1eb471b899</atom:id>
  <atom:entry>
    <atom:title>artifact:update</atom:title>
    <atom:updated>2013-07-23T11:14:03.225-04:00</atom:updated>
    <atom:id>2947f90e-0f5a-4099-b3dc-29124c96c7d0</atom:id>
    <atom:published>2013-07-23T11:14:03.225-04:00</atom:published>
```

```
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:summary />
  </atom:entry>
  <atom:entry>
    <atom:title>artifact:add</atom:title>
    <atom:updated>2013-07-23T11:13:28.513-04:00</atom:updated>
    <atom:id>e41404b3-9ec6-43f5-a6d8-aa6089bc6704</atom:id>
    <atom:published>2013-07-23T11:13:28.513-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:summary />
  </atom:entry>
</atom:feed>
```

4.8.2.20. Get User Audit History API

The Get User Audit History API (**`/s-ramp/audit/user/{username}`**) retrieves an Atom feed containing all of the audit entries for a specific user.

This endpoint is used to get a feed of the audit history for a single user. The request URL can include standard paging parameters. The response is an Atom Feed where each Entry in the feed represents a single audit event in the history of the artifact. A follow up call must be made to the Get Artifact Audit Entry endpoint in order to retrieve full detail information about the audit event. This call may fail if no user exists with the given username.

Table 4.31. PUT Method Request for Update Ontology

HTTP Method	Request	Response
GET	N/A	Atom Feed

Example Request:

```
GET /s-ramp/audit/user/eric HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom" xmlns:s-
ramp="http://docs.oasis-open.org/s-ramp/ns/s-ramp-v1.0"
  s-ramp:itemsPerPage="100" s-ramp:provider="JBoss Overlord" s-
ramp:startIndex="0" s-ramp:totalResults="2">
  <atom:title>S-RAMP Audit Feed</atom:title>
  <atom:subtitle>All Audit Entries for Artifact</atom:subtitle>
  <atom:updated>2013-07-23T11:16:00.545-04:00</atom:updated>
  <atom:id>d49057a2-2f84-48aa-9c79-078b1e86680a</atom:id>
  <atom:entry>
    <atom:title>artifact:update</atom:title>
    <atom:updated>2013-07-23T11:14:03.225-04:00</atom:updated>
```

```
    <atom:id>2947f90e-0f5a-4099-b3dc-29124c96c7d0</atom:id>
    <atom:published>2013-07-23T11:14:03.225-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:summary />
  </atom:entry>
  <atom:entry>
    <atom:title>artifact:add</atom:title>
    <atom:updated>2013-07-23T11:13:28.513-04:00</atom:updated>
    <atom:id>e41404b3-9ec6-43f5-a6d8-aa6089bc6704</atom:id>
    <atom:published>2013-07-23T11:13:28.513-04:00</atom:published>
    <atom:author>
      <atom:name>eric</atom:name>
    </atom:author>
    <atom:summary />
  </atom:entry>
</atom:feed>
```

4.8.2.21. Add Artifact Audit Entry API

The Add Artifact Audit Entry API (**`/s-ramp/audit/artifact/{artifactUuid}`**) adds a user-defined (custom) audit entry to an artifact.

This endpoint is used to add a custom audit entry to a particular artifact. The request must be a POST of an XML document conforming to the audit schema type auditEntry. This call may fail if no artifact exists with the given UUID.

Table 4.32. POST Method Request and Response for Add Artifact Audit Entry

HTTP Method	Request	Response
POST	application/auditEntry+xml	Atom Entry

Example Request:

```
POST /s-ramp/audit/artifact/b086c558-58d6-4837-bb38-6c3da760ae80 HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK

<auditEntry type="custom:foo" uuid="" when="" who="">
  <auditItem type="custom:item-type-1">
    <property name="my-property-1" value="some-value" />
    <property name="my-property-2" value="other-value" />
  </auditItem>
  <auditItem type="custom:item-type-2" />
</auditEntry>
```

4.8.2.22. Get Artifact Audit Entry API

The Get Artifact Audit Entry API (`/s-ramp/audit/artifact/{artifactUuid}/{auditEntryUuid}`) retrieves full detailed information about a single audit entry.

This endpoint is used to get the full details for a single audit event for a particular artifact. The particulars of the detailed information are specific to the type of audit entry, so artifact create detail information might be different from artifact update detail information. Additionally, there is the possibility that the detail information might be from a custom audit entry added by an end user. This call may fail if the artifact does not exist or the audit entry does not exist.

Table 4.33. GET Method Response for Get Artifact Audit Entry

HTTP Method	Request	Response
GET	N/A	Atom Entry

Example Request:

```
GET /s-ramp/audit/artifact/b086c558-58d6-4837-bb38-6c3da760ae80/2947f90e-
0f5a-4099-b3dc-29124c96c7d0 HTTP/1.1
```

Example Response:

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<atom:entry
xmlns="http://downloads.jboss.org/overlord/sramp/2013/auditing.xsd"
xmlns:atom="http://www.w3.org/2005/Atom">
  <atom:title>artifact:update</atom:title>
  <atom:updated>2013-07-23T11:14:03.225-04:00</atom:updated>
  <atom:id>2947f90e-0f5a-4099-b3dc-29124c96c7d0</atom:id>
  <atom:published>2013-07-23T11:14:03.225-04:00</atom:published>
  <atom:author>
    <atom:name>eric</atom:name>
  </atom:author>
  <atom:summary />
  <auditEntry type="artifact:update" uuid="2947f90e-0f5a-4099-b3dc-
29124c96c7d0" when="2013-07-23T11:14:03.225-04:00"
    who="eric">
    <auditItem type="property:added">
      <property name="sramp-properties:foo" value="bar" />
      <property name="sramp-properties:hello" value="world" />
    </auditItem>
    <auditItem type="property:changed" />
    <auditItem type="property:removed" />
  </auditEntry>
</atom:entry>
```

4.9. S-RAMP COMMAND LINE

4.9.1. Start the S-RAMP Shell

1. Navigate to the **../bin** directory.
2. Run this command: **./s-ramp.sh**

Result

The following appears on the screen:

```
************************************************************************
          _____        _______  ___ ___  _________
         /  ___|      | ___ \/ _ \|  \/  | ___ \
         \ `--. _______| |_/ / /_\ \ .  . | |_/ /
          `--. \_______|    /|  _  | |\/| |  __/
         /\__/ /       | |\ \| | | | |  | | |
         \____/        \_| \_\_| |_|_|  |_|_|

  JBoss S-RAMP Kurt Stam and Eric Wittmann, Licensed under the
  Apache License, V2.0, Copyright 2012
************************************************************************
s-ramp>
```

The shell supports auto-completion and keeps a command history for duration of the session.

4.9.2. Connect to the S-RAMP Server

Connecting to the S-RAMP server gives you access to the S-RAMP dashboard and repository. This allows you to access artifacts and their metadata in one place. Once you have signed in, you can browse the repository and inspect your data.

1. Open a command terminal and navigate to ***EAP_HOME*/bin** directory.
2. Start the JBoss EAP server by entering following command:

   $ ./standalone.sh

3. Open another terminal and type **connect** and press Tab key. The command will auto-complete to say **s-ramp:connect http://localhost:8080/s-ramp-server.**

   Enter username and password to connect via the S-RAMP shell. The username and password are defined in sramp.properties.

Result

When you press Tab key, the command will auto-complete to say **s-ramp:connect http://localhost:8080/s-ramp-server** and when you press the return key the cursor will go from red to green indicating successful connection to S-ramp server.

4.9.3. Browse the S-RAMP Repository

Running the commands in this task will allow you to access the S-RAMP repository and peruse its contents. You can view server output and metadata for artifacts. This lets you view additional details of indexed items.

1. Run this command: **s-ramp:query /s-ramp**

   Here is an example of the output of this command:

```
Querying the S-RAMP repository:
    /s-ramp
Atom Feed (9 entries)
  Idx                    Type Name
  ---                    ---- ----
    1           ImageDocument user-properties.png
    2                Document overlord.demo.CheckDeployment-
taskform.flt
    3         BrmsPkgDocument SRAMPPackage.pkg
    4           ImageDocument overlord.demo.SimpleReleaseProcess-
image.png
    5           ImageDocument run-build-install.png
    6                Document overlord.demo.SimpleReleaseProcess-
taskform.flt
    7           ImageDocument audio-input-microphone-3.png
    8            BpmnDocument
overlord.demo.SimpleReleaseProcess.bpmn
    9            TextDocument HttpClientWorkDefinitions.wid
```

2. To obtain the metaData of overlord.demo.SimpleReleaseProcess.bpmn (which is number 8 in the list in the output example), issue this command: **`s-ramp:getMetaData feed:8`**

```
Meta Data for: 31b3acbc-cda8-4856-9e34-d3e645283035
--------------
  -- Core S-RAMP Info --
  Type: BpmnDocument
  Model: ext
  UUID: 31b3acbc-cda8-4856-9e34-d3e645283035
  Name: overlord.demo.SimpleReleaseProcess.bpmn
  Derived: false
  Created By: anonymous
  Created On: 2013-03-08T14:00:37.036-05:00
  Modified By: anonymous
  Modified On: 2013-03-18T14:58:46.328-04:00
s-ramp>
```

4.9.4. Update Artifact Metadata

1. To update a property on the artifact, enter **`s-ramp:property set`** and hit the tab key.

   This lists the properties that you can update.

```
description       name           version
```

2. Add a description: **`s-ramp:property set description "BPMN2 artifact representing the SimpleReleaseProcess"`**.

```
Successfully set property description.
s-ramp> s-ramp:updateMetaData
Successfully updated artifact
overlord.demo.SimpleReleaseProcess.bpmn.
```

3. Verify the change, by entering **`s-ramp:getMetaData feed:8`**.

The change will appear at the bottom of the output:

```
Meta Data for: 31b3acbc-cda8-4856-9e34-d3e645283035
--------------
  -- Core S-RAMP Info --
  Type: BpmnDocument
  Model: ext
  UUID: 31b3acbc-cda8-4856-9e34-d3e645283035
  Name: overlord.demo.SimpleReleaseProcess.bpmn
  Derived: false
  Created By: anonymous
  Created On: 2013-03-08T14:00:37.036-05:00
  Modified By: anonymous
  Modified On: 2013-03-18T16:09:56.879-04:00
  -- Description --
BPMN2 artifact representing the SimpleReleaseProcess
```

4.9.5. Add Custom Properties

1. To add a custom property, run the **`s-ramp> s-ramp:property set month December`** command, where month is the name of the property and December is the value.

```
Successfully set property month.
```

2. To update artifact, run the **`s-ramp> s-ramp:updateMetaData`** command.

```
Successfully updated artifact
overlord.demo.SimpleReleaseProcess.bpmn.
```

3. Verify the change, by entering **`s-ramp:getMetaData feed:8`**.

   The change will appear at the bottom of the output:

```
Meta Data for: 31b3acbc-cda8-4856-9e34-d3e645283035
--------------
  -- Core S-RAMP Info --
  Type: BpmnDocument
  Model: ext
  UUID: 31b3acbc-cda8-4856-9e34-d3e645283035
  Name: overlord.demo.SimpleReleaseProcess.bpmn
  Derived: false
  Created By: anonymous
  Created On: 2013-03-08T14:00:37.036-05:00
  Modified By: anonymous
  Modified On: 2013-03-18T16:21:16.119-04:00
  -- Description --
BPMN2 artifact representing the SimpleReleaseProcess
  -- Custom Properties --
  month: December
```

4. Now when you enter **`s-ramp:property set`** and hit the tab key, you can see your newly added custom property:

```
description    month          name           version
```

4.9.6. Add Classifications

1. To add a classification of deployment-status to your artifact, use the **`s-ramp> s-ramp:classification add "http://www.jboss.org/overlord/deployment-status.owl#Dev"`** command.

```
Successfully added classification
'http://www.jboss.org/overlord/deployment-status.owl#Dev'.
```

2. To update artifact, run the **`s-ramp> s-ramp:updateMetaData`** command.

```
Successfully updated artifact
overlord.demo.SimpleReleaseProcess.bpmn.
```

3. Verify the change, by entering **`s-ramp:getMetaData feed:8`**.

   The change will appear at the bottom of the output:

```
Meta Data for: 31b3acbc-cda8-4856-9e34-d3e645283035
--------------
  -- Core S-RAMP Info --
  Type: BpmnDocument
  Model: ext
  UUID: 31b3acbc-cda8-4856-9e34-d3e645283035
  Name: overlord.demo.SimpleReleaseProcess.bpmn
  Derived: false
  Created By: anonymous
  Created On: 2013-03-08T14:00:37.036-05:00
  Modified By: anonymous
  Modified On: 2013-03-18T16:30:42.641-04:00
  -- Description --
BPMN2 artifact representing the SimpleReleaseProcess
  -- Classifications --
  Classified By: http://www.jboss.org/overlord/deployment-
status.owl#Dev
  -- Custom Properties --
  month: December
```

4.9.7. Query the S-RAMP Repository using XPath2 Syntax

1. S-RAMP supports an XPath2 Syntax for querying. For example, to obtain all WSDL models in the repository, use the **`s-ramp:query /s-ramp/wsdl/WsdlDocument`** command.

```
Querying the S-RAMP repository:
    /s-ramp/wsdl/WsdlDocumenta
Atom Feed (1 entries)
  Idx                    Type Name
  ---                    ---- ----
    1            WsdlDocument OrderService.wsdl
```

When this WSDL file is uploaded, the derived information is extracted from it and stored a WSDL model.

2. To see the various data structures the WSDL file derives, enter **`s-ramp:query /s-ramp/wsdl`** command and press the tab key.

```
Binding                  BindingOperation
BindingOperationFault    BindingOperationInput
BindingOperationOutput
Fault                    Message                  Operation
OperationInput           OperationOutput
Part                     Port                     PortType
WsdlDocument             WsdlExtension
WsdlService
```

The derived data is read-only.

3. To obtain all Operations in this WSDL, use the **`s-ramp:query /s-ramp/wsdl/Operation`** command.

```
Querying the S-RAMP repository:
    /s-ramp/wsdl/Operation
Atom Feed (1 entries)
  Idx                    Type Name
  ---                    ---- ----
    1               Operation submitOrder
```

To narrow down this query, add a condition that the name needs to start with submit: **`s-ramp:query "/s-ramp/wsdl/Operation[xp2:matches(@name, 'submit.*')]"`**

```
Querying the S-RAMP repository:
    /s-ramp/wsdl/Operation[xp2:matches(@name, 'submit.*')]
Atom Feed (1 entries)
  Idx                    Type Name
  ---                    ---- ----
    1               Operation submitOrder
```

Ensure that you use the surrounding quotes, and a **`.`** (dot) after **`submit`** as required by XPath2.

4. To obtain all the artifacts that were derived from an artifact, use the following command:

```
/s-ramp[relatedDocument[@uuid = '<uuid>'
```

In this case, you can use the uuid of a wsdl and get all the artifacts derived from the wsdl:

```
s-ramp:query "/s-ramp[relatedDocument[@uuid = '15a94308-a088-4a03-
ad83-e60239af74e4']]"
Querying the S-RAMP repository:
 /s-ramp[relatedDocument[@uuid = '15a94308-a088-4a03-ad83-
e60239af74e4']]
Atom Feed (16 entries)
  Idx                    Type Name
  ---                    ---- ----
```

```
 1          OperationInput submitOrder
 2             WsdlService OrderService
 3             SoapAddress soap:address
 4   BindingOperationInput wsdl:input
 5             SoapBinding soap:binding
 6                    Part parameters
 7                 Binding OrderServiceBinding
 8  BindingOperationOutput wsdl:output
 9                 Message submitOrderResponse
10         OperationOutput submitOrderResponse
11        BindingOperation submitOrder
12                 Message submitOrder
13               Operation submitOrder
14                    Port OrderServicePort
15                    Part parameters
16                PortType OrderService
```

5. To get a list of all artifacts that were extracted from another archive, use the following command:

```
s-ramp:query "/s-ramp[expandedFromDocument[@uuid = '<uuid>']]"
```

For example, if you uploaded a jar file containing switchyard artifacts, with uddi 67c6f2d3-0f10-4f0d-ada6-d85f92f02a33:

```
s-ramp:query "/s-ramp[expandedFromDocument[@uuid = '67c6f2d3-0f10-
4f0d-ada6-d85f92f02a33']]"
Querying the S-RAMP repository:
 /s-ramp[expandedFromDocument[@uuid = '67c6f2d3-0f10-4f0d-ada6-
d85f92f02a33']]
Atom Feed (3 entries)
  Idx                    Type Name
  ---                    ---- ----
    1             XmlDocument switchyard.xml
    2             XmlDocument beans.xml
    3             XmlDocument faces-config.xml
```

4.9.8. Extending the S-RAMP CLI

The S-RAMP CLI has a number of built-in commands that are ready to be used. However, it is also possible to extend the CLI with new custom commands. This section describes how to do it.

New CLI commands are contributed by creating a class that implements the ShellCommandProvider interface. The provider indicates a namespace for its commands along with a Map of commands (command name > command). The provider and command implementations should be packaged up into a JAR along with a file called **`META-INF/services/org.overlord.sramp.shell.api.ShellCommandProvider`**.

Ensure to make the JAR available to the S-RAMP CLI, either by putting it on the classpath, or else by putting it in the **`~/.s-ramp/commands`** directory.

4.10. S-RAMP MAVEN INTEGRATION

A key feature of the S-RAMP project is the integration between Maven and S-RAMP. This integration is

currently primarily provided by a Maven Wagon that supports the S-RAMP Atom based REST API protocol. You can use this wagon to upload deployable artifacts directly from Maven into a compliant S-RAMP repository.

Additionally, you can use artifacts from the S-RAMP repository as dependencies in a Maven project, also by using the S-RAMP Maven Wagon.

4.10.1. Enabling the S-RAMP Wagon

- In order to use the S-RAMP Wagon in a maven project, enable it in the **pom.xml** build section:

```
<build>
  <extensions>
    <extension>
      <groupId>org.overlord.sramp</groupId>
      <artifactId>s-ramp-wagon</artifactId>
      <version>${s-ramp-wagon.version}</version>
    </extension>
  </extensions>
</build>
```

4.10.2. Deploying to S-RAMP

Once the wagon is enabled, then you can use URLs with a schema of sramp in the **pom.xml**'s distributionManagement section. For example:

```
<distributionManagement>
  <repository>
    <id>local-sramp-repo</id>
    <name>S-RAMP Releases Repository</name>
    <url>sramp://localhost:8080/s-ramp-server/</url>
  </repository>
  <snapshotRepository>
    <id>local-sramp-repo-snapshots</id>
    <name>S-RAMP Snapshots Repository</name>
    <url>sramp://localhost:8080/s-ramp-server/</url>
  </snapshotRepository>
</distributionManagement>
```

With these settings, maven deployments are sent directly to the S-RAMP repository using the S-RAMP API. Note that artifacts are added to the S-RAMP repository with an artifact type based on the maven type of the project. You can override this behavior by adding a query parameter to the repository URL in the **pom.xml**. For example:

```
<distributionManagement>
  <repository>
    <id>local-sramp-repo</id>
    <name>S-RAMP Releases Repository</name>
    <url>sramp://localhost:8080/s-ramp-server/?
artifactType=SwitchYardApplication</url>
  </repository>
</distributionManagement>
```

The above example causes the maven artifact to be uploaded with an S-RAMP artifact type of SwitchYardApplication.

Perform the following steps to deploy the maven artifact directly into s-ramp

1. Cd into the sample workflow directory.
2. Create a file called **sramp-settings.xml** with the following content:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                       http://maven.apache.org/xsd/settings-
1.0.0.xsd">
   <servers>
     <server>
       <id>local-sramp-repo</id>
       <username>admin</username>
       <password>ADMIN_PASSWORD</password>
     </server>
     <server>
       <id>local-sramp-repo-snapshots</id>
       <username>admin</username>
       <password>ADMIN_PASSWORD</password>
     </server>
   </servers>
</settings>
```

   Here, ADMIN_PASSWORD is the password that you chose during installation.

3. Run the following maven command:

```
mvn -s sramp-settings.xml deploy
```

4.10.3. Adding S-RAMP Artifacts as Dependencies

Once you enable the wagon, you can use artifacts from the S-RAMP repository as dependencies in your maven project.

Firstly, you must configure the S-RAMP repository in the maven project as a maven repository. You can do this with the following markup in the **pom.xml**:

```
<repositories>
  <repository>
    <id>local-sramp-repo</id>
    <name>Local S-RAMP Repository</name>
    <url>sramp://localhost:8080/s-ramp-server</url>
    <layout>default</layout>
  </repository>
</repositories>
```

Once you configure the repository, you can reference an S-RAMP artifact as a dependency in the following ways:

- If the artifact was added to S-RAMP using the maven integration to deploy it, then the artifact

in S-RAMP will contain maven specific properties, allowing it to be referenced as a dependency using those maven specific properties. In this case, add the dependency as shown in this example:

```
<dependency>
  <groupId>org.overlord.sramp.wiki</groupId>
  <artifactId>s-ramp-wiki-example</artifactId>
  <version>1.0</version>
</dependency>
```

- If an artifact was added to the S-RAMP repository in some other way (and therefore does not have any maven specific properties), you can still use it as a dependency. In this case, you can reference the dependency by using its S-RAMP artifact model, type, and UUID. The model and type are used to make up a maven groupId, while the UUID becomes the maven artifactId. The version information is not used, but is still required in the **pom.xml**). For example, if a JAR is added to the S-RAMP repository and you wish to use it as a dependency, your **pom.xml** might contain the following dependency:

```
<dependency>
  <groupId>ext.JavaArchive</groupId>
  <artifactId>8744-437487-4734525-382345-923424</artifactId>
  <version>1.0</version>
</dependency>
```

4.10.4. Authentication

Whenever the S-RAMP Maven integration features are used, it is likely that you will need to provide valid authentication credentials. You may provide the S-RAMP repository username and password in the Maven **settings.xml** file. If no credentials are found there, then you will be prompted to enter them when they are needed during the build. Here is an example of providing credentials in the **settings.xml** file:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>local-sramp-repo</id>
      <username>admin</username>
      <password>ADMIN_PASSWORD</password>
    </server>
    <server>
      <id>local-sramp-repo-snapshots</id>
      <username>admin</username>
      <password>ADMIN_PASSWORD</password>
    </server>
  </servers>
</settings>
```