S-RAMP Guide

Dedication

TBD

Preface

TBD

Chapter 1. Introduction to S-RAMP

1.1. The S-RAMP Specification

S-RAMP stands for SOA Repository Artifact Model and Protocol. *S-RAMP* [https://www.oasis-open.org/committees/s-ramp/charter.php] is a new specification worked on by the OASIS Technical Committee.

> The SOA Repository Artifact Model and Protocol (S-RAMP) TC defines a common data model for SOA repositories as well as an interaction protocol to facilitate the use of common tooling and sharing of data. The TC will define an ATOM binding which documents the syntax for interaction with a compliant repository for create, read, update, delete and query operations.
>
> — OASIS Charter *https://www.oasis-open.org/committees/s-ramp/charter.php*

The first version of the specification (1.0) should be finalized in the first half of 2013. Two of the developers on the project participated in the Technical Committee.

1.2. Purpose

The OASIS S-RAMP (SOA Repository Artifact Model and Protocol) specification is intended to provide a common data model and protocol for interacting with a repository of (primarily) SOA artifacts. The goal of the specification is to foster interoperability between repository implementations by standardizing on a data model and API.

This guide will discuss both the OASIS standard and the Overlord open source implementation.

1.3. Overview

The S-RAMP specification includes a foundation document and an Atom based protocol binding document. The foundation document describes core concepts and will be the focus of the first part of this guide. The Atom binding document describes an Atom based API that implementations should provide.

An S-RAMP repository primarily stores artifacts. An artifact is comprised of the following meta-data:

1.4. Core Properties

Relationships. All artifacts in S-RAMP contain a set of core properties such as name, description, creation date, etc. Many of these properties are automatically set by the server when the artifact is added and/or updated. Others, such as description, can be set by clients.

However, most importantly every artifact has an Artifact Model and an Artifact Type. These two properties determine what kind of artifact it is (more on artifact types later, in the Data Models section of this Guide).

Additionally, 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.

1.5. Custom Properties

An artifact may have additional properties set on it by clients. These custom properties are simply arbitrary name/value pairs. The only restriction is that a custom property may not have the same name as a Core Property.

1.6. Classifiers

Another type of meta-data found on S-RAMP artifacts are classifiers. Classifiers are a lot like keywords or tags except that they are hierarchical. Every artifact has a collection of classifiers configured by the client, where each classifier must be a node in an ontology previously uploaded to the repository (presumably by an admin).

An ontology is simply a hierarchy of tags (defined as a subset of the OWL Lite format). This approach allows the repository to be configured with a pre-defined set of hierarchical tags (classifiers) that can be associated with an artifact.

An example is helpful in this case. First, a repository administrator would define and upload an ontology:

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

Once this ontology has been added to the repository, then clients can add, for example, #Alaska or #Canada as classifiers on artifacts. This provides a way to "tag" artifacts in interesting and meaningful ways, and provides a useful means of querying (more on that later).

For more information about ontologies and classifiers, have a look at Section 3 of the S-RAMP Foundation document.

1.7. Relationships

The final bit of meta-data that can be found on an artifact are 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 might define a relationship named "documentedBy" between a wsdl artifact and a text or PDF artifact, indicating that the latter provides documentation for the former.

Chapter 2. Getting Started

2.1. Prerequisites

The S-RAMP application is written in Java. To get started make sure your system has the following:

- Java JDK 1.6 or newer
- Apache Ant 1.7 or newer to use the installer
- Maven 3.0.3 or newer to build and run the examples

2.2. Download, Installation and Configuration

The `s-ramp-<version>.zip` (or tar.gz) archive can be downloaded from the *http://www.jboss.org/overlord* website. Grab the latest, extract the archive and run:

```
ant install
```

The first thing this does is to download *jBPM* [http://www.jboss.org/jbpm], *RESTEasy* [http://www.jboss.org/resteasy] and *ModeShape* [http://www.jboss.org/modeshape] from sourceforge.

Note

This download is slow and can take up to 15 minutes.

The downloads will be stored in the jbpm5 directory, so future installs will be faster. In this same directory it will create a `jbpm-installer` directory with jBPM and a sub-directory containing the application server `jboss-as-7.1.1.Final`. The application server has ModeShape and RESTEasy ed as modules. Next run:

```
ant configure
```

This will add some S-RAMP specific services to jBPM. It deploys the `s-ramp-server.war`, the `s-ramp-ui.war` and the `governance.war`. Finally, it points jBPM to obtain its workflow data from the S-RAMP repository, rather than BRMS/Drools-Guvnor. At this point we want to populate the S-RAMP repository with some initial data. For this the repository needs to be up and running. Go ahead and start the JBoss application server by running:

```
ant start
```

If your system support the `tail` command, you can run:

```
ant tail
```

This will monitor the startup process. It should be ready as soon as you see the following in the application server console output:

```
[exec] 12:00:13,905 INFO  [org.jboss.as.server] (DeploymentScanner-threads -
 2) JBAS018559: Deployed "s-ramp-ui.war"
[exec] 12:00:13,910 INFO  [org.jboss.as.server] (DeploymentScanner-threads -
 2) JBAS018559: Deployed "s-ramp-server.war"
[exec] 12:00:13,915 INFO  [org.jboss.as.server] (DeploymentScanner-threads -
 2) JBAS018559: Deployed "s-ramp-governance.war"
[exec] 12:00:13,918 INFO  [org.jboss.as.server] (DeploymentScanner-threads -
 2) JBAS018559: Deployed "jbpm-human-task-war.war"
[exec] 12:00:13,922 INFO  [org.jboss.as.server] (DeploymentScanner-threads -
 2) JBAS018559: Deployed "jbpm-gwt-console.war"
[exec] 12:00:13,925 INFO  [org.jboss.as.server] (DeploymentScanner-threads -
 2) JBAS018559: Deployed "jbpm-gwt-console-server.war"
[exec] 12:00:13,928 INFO  [org.jboss.as.server] (DeploymentScanner-threads -
 2) JBAS018559: Deployed "jbpm-form-builder.war"
[exec] 12:00:13,931 INFO  [org.jboss.as.server] (DeploymentScanner-threads -
 2) JBAS018559: Deployed "drools-guvnor.war"
[exec] 12:00:13,933 INFO  [org.jboss.as.server] (DeploymentScanner-threads -
 2) JBAS018559: Deployed "designer.war"
```

Tip

S-RAMP can be configured to use a remote jBPM server, so to cut down on bootup time you can deploy the S-RAMP applications on a dedicated application server.

Now you can upload the by running:

```
ant upload
```

This completes the installation process.

2.3. Check your Installation

To make sure your installation works you can fire up the *s-ramp-ui* [http://localhost:8080/s-ramp-ui]. You should see the GUI as shown in *Figure 2.1, “Welcome screen of the s-ramp-ui.”*:

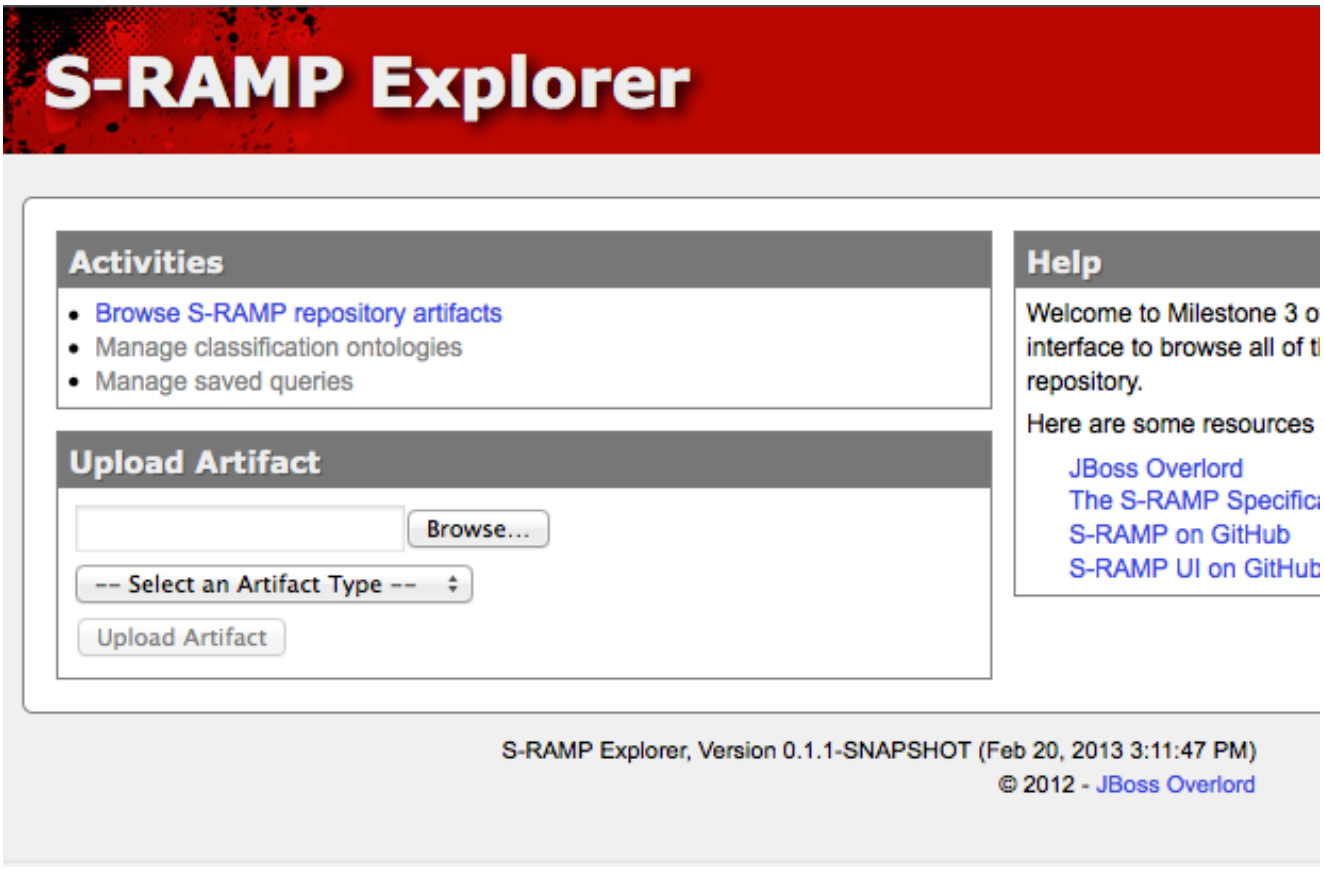

Figure 2.1. Welcome screen of the s-ramp-ui.

You can click on `Browse Artifacts` and see a list of files related to the S-RAMP default workflows. Alternatively you can fire up the `s-ramp shell` in the `bin` directory of the distribution:

```
./s-ramp.sh
************************************************************************
          _____       ______  ___ ___  ________
         /  ___|      | ___ \/ _ \|  \/  | ___ \
         \ `--. ______| |_/ / /_\ \ .  . | |_/ /
          `--. \______|    /|  _  | |\/| |  __/
         /\__/ /      | |\ \| | | | |  | | |
         \____/       \_| \_\_| |_|_|  |_|_|

  JBoss S-RAMP Kurt Stam and Eric Wittmann, Licensed under the
  Apache License, V2.0, Copyright 2012
************************************************************************
s-ramp>
```

To connect the shell to the server type `connect` and hit the tab key. It should auto-complete to say `s-ramp:connect http://localhost:8080/s-ramp-server` and when hitting the return key the cursor should go from red to green. To browse the artifacts in the repository run the following query:

```
s-ramp> s-ramp:query /s-ramp
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
```

In later chapters will go into more detail, but if this all worked you can be sure that your installation is in good working order.

Chapter 3. 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 of the Guide describes these different models.

An artifact may have document (e.g file) content or it may be a purely logical artifact. In either case, clients typically add artifacts to the repository directly (e.g. via the S-RAMP Atom API, described later in this guide).

Additionally, some document style artifact types when added to the repository, will result in the creation of a set of "derived" artifacts. For example, if an XSD document is added to the repository, the server will automatically extract the element declarations from the content of the file resulting in a set of additional artifacts "related" to the original. This will be described in detail further in the XSD Data Model section.

3.1. Core Data Model (core)

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

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

3.2. XML Schema (XSD) Data Model (xsd)

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 will additionally "index" the artifact by automatically creating a number of derived artifacts of the following types from the XSD content.

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

3.3. WSDL Data Model (wsdl)

The WSDL model defines a single document style artifact, WsdlDocument, and a number of derived artifact types. Similarly to the XsdDocument type, when a WSDL dcument is added to the

repository, the server will automatically derive additional artifacts (listed below) from the content of the WSDL file.

For further details about the WSDL Model, please see the S-RAMP specification's foundation document, section 2.4.2.

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
BindingOperationOutput	<derived>	ncName, namespace
BindingOperationFault	<derived>	ncName, namespace

3.4. Policy Data Model (policy)

This data model is present to represent the primary components of a WS-Policy document.

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

3.5. SOA Data Model (soa)

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

Artifact Type	Parent Type	Properties
HumanActor		

Artifact Type	Parent Type	Properties
Choreography		
ChoreographyProcess		
Collaboration		
CollaborationProcess		
Composition		
Effect		
Element		
Event		
InformationType		
Orchestration		
OrchestrationProcess		
Policy		
PolicySubject		
Process		
Service		
ServiceContract		
ServiceComposition		
ServiceInterface		
System		
Task		

3.6. Service Implementation Data Model (serviceImplementation)

The Service Implementation model adds SOA service implementation artifact types underneath the (already mentioned) SOA Data Model.

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

3.7. Custom/Extension Data Models (ext)

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 however that the server will not have a definition of the model - 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.

As an example, a client might define the following Data Model for a J2EE web application domain:

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

Chapter 4. Query Language

Another key aspect of the S-RAMP specification is the query language it defines, which allows clients to find 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.

Tip

for detailed information about the S-RAMP Query Language, see Section 4 of the S-RAMP specification's foundation document.

As you might imagine, the query language allows clients to find artifacts based on any of the already discussed artifact meta-data, including:

*Core Properties *Custom Properties *Classifiers *Relationships

The basic structure of a typical S-RAMP query looks like this:

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

Of course, not all of the components of the above query are required. It is likely best to provide the following table of examples of a range of queries:

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

Query	What It Selects
	named *operation*. (This effectively returns all of the order service port type's operations)
/s-ramp/ext/ServletFilter[relatedDocument[@uuid="12345"]]	All servlet filter artifacts derived from (i.e. 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*.

Chapter 5. S-RAMP REST API

The intent of the S-RAMP specification is to outline a data model and protocol designed to define how a repository should store and manipulate artifacts. The foundation document defines the former, while various protocol binding documents define the latter. Version 1 of the S-RAMP specification includes a single, Atom based protocol binding.

This section of the guide describes the highlights of the Atom API.

Tip

For more information on the S-RAMP Atom API, see the S-RAMP Atom Binding document.

5.1. Overview

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. This Guide will give examples based on this notional format. But it bears repeating that for any given server implementation, a client should first query the Atom service document at the following endpoint:

```
GET /s-ramp/servicedocument
```

The resulting service document will contain a set of workspaces representing the artifact collections supported by the server, along with endpoints indicating how to manipulate them.

However, the Atom Binding document does outline the inputs, outputs, and REST verbs that must be used for each of the supported operations. In general, the Atom API data models are used to wrap custom S-RAMP specific XML structures. Atom Entry documents are used when dealing with individual artifacts, while Atom Feed documents are used when dealing with lists of documents.

5.2. Adding Artifacts

There are two basic types of artifacts from the protocol standpoint: document style artifacts (those artifacts that are based on files/binary content) and logical (direct instantiation) artifacts. In the case of a document-style artifact, the client must POST the binary content to the correct Atom Endpoint. In the case of a direct artifact (no document content) the client must POST an Atom Entry containing an S-RAMP artifact XML entity to the appropriate endpoint. The server will respond with an Atom Entry containing the full meta data of the newly created artifact (if successful).

Notional REST Endpoint

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

5.3. Updating Artifacts

A client can update the meta data (properties, classifiers, relationships) by performing a PUT request to the artifact's endpoint. The artifact's endpoint can be found either by querying for the artifact or as part of the returned Atom Entry returned when the artifact was created.

Notional REST Endpoint

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

5.4. Deleting Artifacts

Not surprisingly, an artifact can be deleted by a client by performing a DELETE request to the artifact's endpoint.

Notional REST Endpoint

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

5.5. Querying

Performing an S-RAMP query is a matter of issueing a GET or POST to the S-RAMP query endpoint. In addition, full feeds are available for all Artifact Models and Artifact Types. In both cases, the response is an Atom Feed where each Entry provides summary information about an artifact in the respository. To retrieve full details about a given entry in the feed (custom properties, classifiers, relationships), the client must issue an additional GET. Only a subset of the core properties, such as name and description, are mapped to the Atom Entry in a feed.

5.5.1. Queries

When querying, the client can either GET or POST to the following notional endpoint:

Notional REST Endpoint

```
GET /s-ramp
```

The following parameters are supported (all parameters except the *query* param have reasonable defaults):

**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. This property can be included multiple times.

5.5.2. Feeds

When retrieving a simple model or type feed, the client must issue a GET request to the appropriate model or type endpoint.

Notional REST Endpoint

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

The following parameters are supported (all parameters have reasonable defaults):

**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.

5.6. Getting Full Artifact

In order to retrieve the full meta data for an artifact, the client must issue a GET request to the appropriate artifact endpoint. 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.

Notional REST Endpoint

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

5.7. Batch changes: S-RAMP Archives (Packages)

A powerful additional feature of the S-RAMP API is the batch processing function. 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. The format of an S-RAMP package archive is described further in the S-RAMP Atom Binding document (Section 2.3.5.2.2.1 - no I'm not kidding, that's really the section).

Chapter 6. Overlord S-RAMP Implementation

6.1. Overview

TBD

6.2. Server

TBD

6.2.1. Overview

TBD

6.2.2. Configuring

TBD

6.2.3. Extending: Custom Deriver

TBD

6.3. Client

TBD

6.3.1. Basic Usage

TBD

6.3.2. Extended Feature: Ontologies

TBD

6.3.3. Extended Feature: Auditing

TBD

Chapter 7. Overlord S-RAMP REST API Endpoints

The S-RAMP Atom API protocol binding does not dictate the format of the API endpoints. Clients must request the /servicedocument and then inspect the workspaces found therein. However, the Overlord implementation's endpoints conform to the notional syntax described in the S-RAMP specification's foundation document. The following table lists the endpoints available in the Overlord implementation:

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*
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*

7.1. API: Get Service Document

```
/s-ramp/servicedocument
```

Retrieves the service document.

HTTP Method	Request	Response
GET	N/A	Atom Service Document

The service document contains a workspace for each of the S-RAMP data models supported by the server.

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>
```

Tip

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

7.2. API: Publish Artifact

Publishes a new artifact into the repository.

There are three ways this endpoint can be invoked, depending on the type of artifact being published:

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

7.2.1. Publish a Document Style Artifact

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

HTTP Method	Request	Response
POST	Binary File	Atom Entry

Publishing a document style artifact is simply a matter of POSTing the binary content of the document to the appropriate endpoint.

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>
```

7.2.2. Publish a Non-Document Style Artifact

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

HTTP Method	Request	Response
POST	Atom Entry	Atom Entry

Publishing a non-document style artifact requires an Atom Entry (which 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.

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=&quot;entry&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>
```

7.2.3. Publish a Document Style Artifact with Meta-Data

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

HTTP Method	Request	Response
POST	Multipart/Related	Atom Entry

Sometimes it is convenient to publish an artifact and update its meta-data in a single request. This can be done by POSTing a multipart/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.

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=&amp;quot;entry&amp;quot;" />
  <atom:link href="http://example.org/s-ramp/core/Document/05778de3-
be85-4696-b5dc-d889a27f1f6e"
    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>2013-05-14T13:43:09.708-04:00</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>
```

7.3. API: Update Artifact

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

Updates an artifact's meta data.

HTTP Method	Request	Response
PUT	Atom Entry	N/A

This endpoint is used to update a single artifact's meta data, including core properties, custom properties, classifiers, and relationships. Typically the client should first retrieve the artifact (e.g. by invoking the Get Artifact endpoint), make changes to the artifact, then issue a PUT request to the Update Artifact endpoint.

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>
```

Chapter 8. Overlord S-RAMP Command Line

Using the S-RAMP cmdline tool `s-ramp.sh`

In the `bin` directory of the distribution you can find the `s-ramp.sh`. Run this command to fire up the shell

```
./s-ramp.sh
************************************************************************
          _____       ______  ___ ___  ________
         /  ___|      | ___ \/ _ \|  \/  | ___ \
         \ `--. ______| |_/ / /_\ \ .  . | |_/ /
          `--. \______|    /|  _  | |\/| |  __/
         /\__/ /      | |\ \| | | | |  | | |
         \____/       \_| \_\_| |_|_|  |_|_|

  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.

8.1. Connecting to S-RAMP server

To connect the shell to the server type `connect` and hit the tab key. It should auto-complete to say `s-ramp:connect http://localhost:8080/s-ramp-server` and when hitting the return key the cursor should go from red to green. Of course you will need to update the server and port information if your S-RAMP repository runs elsewhere.

8.2. Browsing the S-RAMP repository

To browse the artifacts in the repository run the following query:

```
s-ramp> s-ramp:query /s-ramp
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
```

To obtain the metaData of `overlord.demo.SimpleReleaseProcess.bpmn`, which is number 8 in the list, issue

```
s-ramp> 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>
```

8.3. Updating artifact MetaData

8.3.1. Properties

To update a property on the artifact use `s-ramp:property set` and hit the `tab` key

```
s-ramp> s-ramp:property set
description       name           version
```

this shows a list of properties that can be updated. To add a description to this artifact use

```
s-ramp> 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.
```

To verify issue

```
s-ramp> 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: 2013-03-18T16:09:56.879-04:00
  -- Description --
BPMN2 artifact representing the SimpleReleaseProcess
```

and you can see the added description at the bottom of the printout.

8.3.2. Custom Properties

To add a `custom property` called `kurt` with value `stam` you can run

```
s-ramp> s-ramp:property set kurt stam
Successfully set property kurt.
s-ramp> s-ramp:updateMetaData
Successfully updated artifact overlord.demo.SimpleReleaseProcess.bpmn.
```

and to verify that the `custom property` was added issue

```
s-ramp> 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: 2013-03-18T16:21:16.119-04:00
  -- Description --
BPMN2 artifact representing the SimpleReleaseProcess
  -- Custom Properties --
  kurt: stam
s-ramp>
```

When hitting the `tab` key on `s-ramp:property set` results in

```
s-ramp> s-ramp:property set
```

```
description    kurt           name           version
```

which now had the added `custom property kurt`.

8.3.3. Classifications

To add a `classification` of `deployment-status` to your artifact use

```
s-ramp> s-ramp:classification add "http://www.jboss.org/overlord/deployment-
status.owl#Dev"
Successfully added classification 'http://www.jboss.org/overlord/deployment-
status.owl#Dev'.
s-ramp> s-ramp:updateMetaData
Successfully updated artifact overlord.demo.SimpleReleaseProcess.bpmn.
```

and to verify that it was added

```
s-ramp> 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: 2013-03-18T16:30:42.641-04:00
  -- Description --
BPMN2 artifact representing the SimpleReleaseProcess
  -- Classifications --
  Classified By: http://www.jboss.org/overlord/deployment-status.owl#Dev
  -- Custom Properties --
  kurt: stam
s-ramp>
```

8.4. Querying the S-RAMP Repository using XPath2 Syntax

TBD - reference to generic section on XPath2 query syntax

S-RAMP supports an XPath2 Syntax for querying. For example to obtain all WSDL models in the repository use

```
s-ramp> s-ramp:query /s-ramp/wsdl/WsdlDocument
```

```
Querying the S-RAMP repository:
    /s-ramp/wsdl/WsdlDocument
Atom Feed (1 entries)
  Idx                   Type Name
  ---                   ---- ----
    1           WsdlDocument OrderService.wsdl
s-ramp>
```

When this WSDL file was uploaded `derived` information was extracted from it and stored a WSDL model. TO see the various data structures it derived simply hit the tab on `s-ramp:query /s-ramp/wsdl`

```
s-ramp> s-ramp:query /s-ramp/wsdl/
Binding                 BindingOperation         BindingOperationFault
 BindingOperationInput    BindingOperationOutput
Fault                   Message                  Operation
 OperationInput           OperationOutput
Part                    Port                     PortType
 WsdlDocument             WsdlExtension
WsdlService
s-ramp>
```

Note that derived data is `read only`, and cannot be updated by the user.

To obtain all Operations in this WSDL use

```
s-ramp:query /s-ramp/wsdl/Operation
Querying the S-RAMP repository:
    /s-ramp/wsdl/Operation
Atom Feed (1 entries)
  Idx                   Type Name
  ---                   ---- ----
    1              Operation submitOrder
s-ramp>
```

You can narrow this query down even more by adding 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
s-ramp>
```

don't forget to use the surrounding quotes, and a `.` after `submit` as required by XPath2.

To obtain all the artifacts that were derived from an artifact you can use

```
/s-ramp[relatedDocument[@uuid = '<uuid>'
```

In this case we 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
```

To get a list of all artifacts that were extracted from another archive use

```
s-ramp:query "/s-ramp[expandedFromDocument[@uuid = '<uuid>']
```

let's say we 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
```

Chapter 9. Overlord S-RAMP Maven Integration

9.1. Overview

TBD

9.2. Enabling the S-RAMP Wagon

TBD

9.3. Deploying to S-RAMP

TBD

Chapter 10. S-RAMP Samples

Chapter 11. SOA Governance

11.1. Introduction

To support SOA Governance management a governance application monitors a S-RAMP compliant repository for certain events. These events can then be used to kickoff governance workflows.

11.2. Concepts

TBD

11.3. Workflow

Organizations define policies and processes to managing artifacts. These policies and processes will be different for each organization. We therefore enlisted the help of a BPMN workflow engine. This should give users the needed flexibility to implement their specific workflow. We ship a number of best practices workflow that users can copy and or modify to their needs.

11.3.1. Repository Events using S-RAMP Queries

Events in the repository can be used to kickoff workflows. The governance application monitors an S-RAMP compliant repository using S-RAMP queries.

Chapter 12. API Managment

12.1. Introduction

Appendix A. Example Appendix

One or more optional appendixes go here at section level 1.

A.1. Appendix Sub-section

Sub-section body.

Example Bibliography

The bibliography list is a style of AsciiDoc bulleted list.

Books

[walsh-muellner] Norman Walsh & Leonard Muellner. *DocBook - The Definitive Guide*. O'Reilly & Associates. 1999. ISBN 1-56592-580-7.

Articles

[abc2003] Gall Anonim. *An article*, Whatever. 2003.

Example Glossary

Glossaries are optional. Glossaries entries are an example of a style of AsciiDoc labeled lists.

A glossary term
: The corresponding (indented) definition.

A second glossary term
: The corresponding (indented) definition.

Example Colophon

Text at the end of a book describing facts about its production.

Index