HP Systinet

Software Version: 10.01
Windows and Linux Operating Systems

Developer Guide

Document Release Date: June 2015
Software Release Date: June 2015

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to: **http://h20230.www2.hp.com/selfsolve/manuals**

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to: **http://h20229.www2.hp.com/passport-registration.html**

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at: **http://www.hp.com/go/hpsoftwaresupport**

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

HP Software Solutions Now accesses the HPSW Solution and Integration Portal Web site. This site enables you to explore HP Product Solutions to meet your business needs, includes a full list of Integrations between HP Products, as well as a listing of ITIL Processes. The URL for this Web site is **http://h20230.www2.hp.com/sc/solutions/index.jsp**

About this PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

Contents

Chapter 1: In this Guide

This HP Systinet Developer Guide describes additional features and methods to enable developers to better interact with Systinet and contains the following topics:

- "IDE Integration" on page 40

  How to integrate Systinet with IDEs.

- "Atom-Based REST Interface" on page 10

  A guide to the Atom REST Interface.

- "Executable Objects" on page 46

  Execute tasks in Systinet directly using URLs.

- "Using DQL" on page 47

  A guide to using DQL to write queries.

- "WebDAV Compliant Publishing" on page 70

  Using WebDav clients with the publishing location space.

- "Technical Security" on page 73

  A technical overview of Systinet from the security point of view.

- "Custom Source Parsers" on page 81

  How to write your own source parser.

- "Custom Validation Handlers" on page 84

  How to write your own validation handler.

- "Lifecycle Remote Client" on page 42

  A remote client for lifecycle manipulation.

- "Validation Client" on page 85

  A command-line tool for policy compliance validation.

- "Publishing Extensibility" on page 91

  How to extend publishing for custom document types.

Chapter 2: Atom-Based REST Interface

Systinet uses an ATOM-based REST interface.

Note: Systinet also includes a deprecated proprietary REST Interface. For information about this interface, see the *Systinet3.20 Developer Guide*.

Access the Systinet platform service document using the following URL:

`http://`*`hostname`*`:`*`port`*`/`*`context`*`/platform/rest`

Hostname, port, and context are set during installation. For example, if you used the default settings and installed to your local machine, use the following URL:

`http://localhost:8080/systinet/platform/rest`

If set up during installation, an HTTPS secure endpoint is available which requires credentials to access.

A default secure endpoint uses the following URL:

`https://localhost:8443/systinet/platform/rest`

Note: Use `restSecure` instead of `rest` if you are using HTTP basic authentication.

The service document consists of workspaces, which in turn contains feeds made up of entries, as shown in the following example:

Platform Service Document

```
<?xml version="1.0" encoding="UTF-8"?>
<app:service xml:base="http://localhost:8080/systinet/platform/rest/"
    xmlns:app="http://www.w3.org/2007/app">
 <app:workspace>
  <atom:title type="text"
      xmlns:atom="http://www.w3.org/2005/Atom">SDM collections</atom:title>
  <app:collection href="./artifact/reportArtifact">
   <app:accept/>
   <atom:title type="text"
       xmlns:atom="http://www.w3.org/2005/Atom">Collection of Reports</atom:title>
   <app:categories href="./category-document/
       uddi:systinet.com:systinet:model:taxonomies:artifactTypes:_artifactType"/>
   <app:categories href="./category-document/
       uddi:systinet.com:systinet:model:taxonomies:reportTypes:reportType"/>
   <app:categories href="./category-document/

uddi:systinet.com:systinet:model:taxonomies:reportCategories:reportCategory"/>
   <app:categories href="./category-document/
```

```
       uddi:systinet.com:systinet:model:taxonomies:reportStatus:reportStatus"/>
   <app:categories href="./category-document/

uddi:systinet.com:systinet:model:taxonomies:reportResultCodes:reportResultCode"/>
 </app:collection>
  ...
 </app:workspace>
 <app:workspace>
  <atom:title type="text"
      xmlns:atom="http://www.w3.org/2005/Atom">Publishing Locations</atom:title>
  <app:collection href="./location">
   <app:accept/>
  </app:collection>
 </app:workspace>
 <app:workspace>
  <atom:title type="text"
      xmlns:atom="http://www.w3.org/2005/Atom">System Information</atom:title>
  <app:collection href="./system">
   <app:accept/>
  </app:collection>
 </app:workspace>
</app:service>
```

The interface is described in the following sections:

- "Workspaces" below
- "Feeds" on page 13
- "Entries" on page 21
- "Category Documents" on page 31
- "Atom REST Operations" on page 31
- "Atom REST ETags" on page 33
- "Atom REST Client" on page 35

Workspaces

The platform service document consists of the following workspaces:

- "SDM Collections Workspace" on the next page

  The SDM workspace reflects the structure of the Service Definition Model (SDM) and defines feeds for the collections in the Systinet repository (read-only).

- "Publishing Locations Workspace" below

  The locations workspace reflects the structure of attached data content in Systinet created by the publisher.

- "System Collections Workspace" on the next page

  The system workspace contains system information used by Systinet (read-only).

SDM Collections Workspace

The SDM collections workspace contains a collection for each artifact type in the Service Definition Model (SDM) for which an instance can be created within its artifact hierarchy.

Note: Customization Editor can be used to modify the SDM, so your configuration may vary from specific examples in this documentation. For details, see the *Customization Editor Guide*.

Each collection in the workspace consists of the following:

- <app:collection href="./artifact/*artifactType*">

  The reference defines the URL used for the feed for that particular artifact type collection. For details, see "Artifact Collection Feeds" on the next page.

- <app:categories href="./category-documents/*taxonomy*">

  Categories can occur in feed entries and some feed readers can perform filtering according to these categories.

Publishing Locations Workspace

The publishing locations workspace consists of a single collection. This collection is an atom feed made up of entries where the entry can be one of the following types:

- Subcollection
- Resource

The subcollections and resources reflect content uploaded to Systinet using its publication feature.

For more details, see "How to Publish Content" in the *User Guide*.

This location is available as a feed and is accessible with a WebDAV client.

For details, see "Publishing Location Feeds" on page 19 and "WebDAV Compliant Publishing" on page 70.

System Collections Workspace

The system collections workspace contains a single collection. This collection contains information about the running system.

Feeds

You can access the content of the repository using feeds.

- "Artifact Collection Feeds" below
- "Publishing Location Feeds" on page 19
- "Artifact Relationships Feed" on page 20
- "Artifact History Feed" on page 21
- "Artifact Comments Feed" on page 21

Artifact Collection Feeds

Every artifact type collection in the SDM is accessible as a feed.

Use the reference defined in the SDM collections workspace to access a collection feed.

For example, the WSDL collection feed is accessed with URL:

`http://localhost:port/context/platform/rest/artifact/wsdlArtifact`

WSDL Collection Feed

```
<feed xml:base="http://localhost:8180/platform/rest/artifact/wsdlArtifact"
    xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/"
    xmlns="http://www.w3.org/2005/Atom">
  <id>urn:hp.com:2009:02:systinet:platform:artifacts:sdm:wsdlArtifact</id>
  <updated>2009-06-19T14:54:11.614+02:00</updated>
  <title type="text" xml:lang="en">Collection of WSDLs</title>
  <opensearch:itemsPerPage>50</opensearch:itemsPerPage>
  <opensearch:startIndex>1</opensearch:startIndex>
  <link href="artifactBase" type="application/atom+xml;type=feed"
      rel="urn:hp.com:2009:02:systinet:platform:artifacts:parent"
      title="parent sdm feed"/>
  <link href="wsdlArtifact?start-index=1&amp;page-size=50"
      type="application/atom+xml;type=feed"
      rel="self" title="feed self"/>
  <author>
    <name>system:restadmin</name>
  </author>
  <generator>HP Systinet</generator>
  <entry>
    <id>urn:hp.com:2009:02:systinet:platform:artifact:4465c1e1-f214-47c5-a958-
d3202ab20dfa</id>
    <updated>2009-06-09T10:06:35.443+02:00</updated>
    <title type="text" xml:lang="en">paymentMethod.wsdl</title>
    ...
  </entry>
  ...
</feed>
```

Each artifact type collection feed consists of the following descriptors:

Descriptors	Description
id	The feed identification.
updated	The last update time.
title	The name of the feed.
link	A set of links with the following link types indicated by the `rel` attribute: • `urn:hp.com:2009:02:systinet:platform:artifacts:parent` Links to collection feeds for super artifacts in the inheritance category. • `urn:hp.com:2009:02:systinet:platform:artifacts:child` Links to collection feeds for descendant artifact types.
entry	The set of entries in the feed. For more details, see "Artifact Atom Entries" on page 22.
opensearch:startIndex	Starting point for the feed relative to index entries. The first indexed item is `1`.
opensearch:itemsPerPage	Number of items per page.

You can modify the output of the feed as described in the following sections:

- "Filtering Feeds" on the next page
- "Viewing Entry Content in Feeds" on the next page
- "Domains in Feeds" on page 16
- "Property Based Searching" on page 16

- "Feed Ordering" on page 17
- "Bulk GETs" on page 18

You can also combine these output methods.

Separate each term with **&**.

For example, to get artifacts 10-79 which contain `policy` in the description, ordered primarily by their name in descending order and then by description in ascending order, and displaying properties defined in `artifactBase`, use the following URL:

```
http://host:port/context/platform/rest/artifact/artifactBase?p.description=*policy*
&start-index=10&page-size=70&order-by=name-,description&inline-content
```

Filtering Feeds

Feeds are presented in the REST interface as a set of equivalent collections.

Examples of feeds include:

- `http://localhost:port/context/platform/rest/artifact/implementationArtifact`
- `http://localhost:port/context/platform/rest/artifact/xmlServiceArtifact`
- `http://localhost:port/context/platform/rest/artifact/webServiceArtifact`
- `http://localhost:port/context/platform/rest/artifact/businessServiceArtifact`
- `http://localhost:port/context/platform/rest/artifact/wsdlArtifact`

Viewed in this way, the feeds form a flat structure. However, there are established relationships between feeds in terms of an inheritance hierarchy.

The root of the hierarchy is
`http://localhost:port/context/platform/rest/artifact/artifactBase`.

You can use abstract artifact type feeds to obtain all artifact types lower in the hierarchy. For example, the implementationArtifact feed contains all SOAP service, XML service, and web application artifacts.

The relationships between feeds are realized via
`urn:hp.com:2009:02:systinet:platform:artifacts:parent` and
`urn:hp.com:2009:02:systinet:platform:artifacts:child` links.

Viewing Entry Content in Feeds

You can use feeds to obtain multiple artifact entry content as well.

Add `?inline-content` to the collection feed URL to obtain the full content for each entry in the feed.

Note: The properties displayed in the content for an entry are determined by the artifact type used in the feed URL. Properties specific to an artifact type lower in the hierarchy are not displayed.

Domains in Feeds

The domain can be specified using a domain parameter in the `/artifact/` segment or the feed URL.

For example, `http://localhost:port/context/platform/rest/artifact;domain=defaultDomain/wsdlArtifact` shows all WSDLs in the Default Domain.

> **Note:** Artifacts may be moved across domains using a PUT operation that specifies the system property `_domainId`.

Property Based Searching

You can search for specific artifacts in a feed with property based filtering. You can filter by any property type regardless of its type and cardinality, but the elementary conditions are always primitive values. The filtering property must be present in the artifact type defining the feed.

The property must be one of the following elementary types:

- text
- integer
- bigInteger
- date
- double
- boolean
- uuid

To view the permitted property names for a particular artifact feed, you can examine the SDM with URL:

`http://host:port/context/platform/rest/system/model`.

If you want to filter by a compound property (for example, category property which has 3 compounds: taxonomyUri, name, value) you must use dot notation. For example to search by compound `val` (value) of property `criticality` on `businessServiceArtifact` use the following URL:

```
http://host:port/systinet/platform/rest/artifact/businessServiceArtifact?p.criticality.val=uddi:systinet.com:soa:model:taxonomies:impactLevel:high
```

Only business services artifacts with high criticality are listed.

For text property filtering, operator case-insensitive-equals is used, but can explicitly use wildcards. To find all service artifact with `svc` in their name submit the following URL:

```
http://host:port/systinet/platform/rest/artifact/businessServiceArtifact?p.name=*sv
c*
```

The following wildcards are supported:

- * for zero or more arbitrary characters.
- _ for exactly one arbitrary character.

> **Note:** Systinet does not support explicit boolean operators but there is an implicit AND for conditions on different properties and an implicit OR on conditions on the same property.

The following examples show various ways to use property searching:

- Artifacts with a name starting with `service` and a description containing `assertion`:

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?p.name=
  service*&p.description=*assertion*
  ```

- Artifacts with a name containing either starting with `service` or containing `assertion`:

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?p.name=
  service*&p.name=*assertion*
  ```

- Deleted artifacts only.

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?p._deleted=true
  ```

> **Tip:** To view the category values, open the category document, for details, see "Category Documents" on page 31.

Feed Ordering

By default, entries in feeds are ordered by their `atom:updated` element.

Add `?order-by=` to the collection feed URL to change the order.

- Entries ordered by name (ascending):

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?order-by=name
  ```

- Entries ordered by name (descending):

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?order-by=name-
  ```

- Entries ordered by name (descending), then description (ascending):

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?order-
  by=name-,description
  ```

You can also use properties for ordering with the same conditions as for searching.

For details, see "Property Based Searching" on page 16.

Feed Paging

You can also control the feed paging.

- The first ten entries:

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?page-size=10
  ```

- Entries 10-19 (inclusive):

  ```
  http://host:port/context/platform/rest/artifact/artifactBase?page-size=10&start-
  index=10
  ```

Note: The default number of entries is 50 and the maximum number of entries is 500.

Bulk GETs

A specific REST use case is a Bulk GET - getting multiple artifacts in a single request/response interaction. This can be handled via a property based search on specific collections, presuming that the UUIDs of the artifacts to retrieve are known.

For example, assume the following business service artifacts with UUIDs, bs1 and bs2. There are 3 web service artifacts with UUIDs ws1, ws2, and ws3. The ATOM GET request to return all 5 artifacts at once is as follows:

```
http://host:8080/systinet/platform/rest/artifact/artifactBase?p._uuid=bs1&p._
uuid=bs2&p._uuid=ws1&p._uuid=ws2&p._uuid=ws3&inline-content
```

Notice the inline-content flag, it specifies the inclusion of proprietary XML representation into atom entries.

Submitting this URL returns a feed with 5 artifacts, assuming they exist. But inside the atom content there are only properties specific to the artifactBase artifact type. For example: businessServiceArtifact defines the property `criticality`. This property is not present in the atom content because it is not declared at artifactBase level. The properties listed in the atom content are strictly driven by artifact type, specified as one part of the URL (in our case artifactBase).

However, there is one exception, relationship properties are always listed in the atom content regardless of the given artifact type. The business service artifact defines a relationship property `service`. This property is not declared at artifactBase level, however, it is present in the XML representation regardless of the artifact type given in the URI.

If you want to get the full set of properties (even those specific to the given artifact type), you must perform multiple GETs per artifact type. In our example, this requires the following 2 GETs:

```
http://host:8080/systinet/platform/rest/artifact/businessServiceArtifact?p._
uuid=bs1&p._uuid=bs2&inline-content
```

```
http://host:8080/systinet/platform/rest/artifact/webServiceArtifact?p._uuid=ws1&p._
uuid=ws2&p._uuid=ws3&inline-content
```

By submitting these two HTTP GETs, you obtain full representation of the 5 artifacts: bs1, bs2, ws1, ws2, and ws3.

Publishing Location Feeds

The location feed enables you to browse the attached data content in the repository.

Systinet adds this content whenever you publish an artifact associated with attached data content. For details, see "How to Publish Content" in the *User Guide*.

The publishing location is accessible using a WebDAV client. For details, see "WebDAV Compliant Publishing" on page 70.

The content feed consists of resources (the data content) organized into collections (folders). Access the feed using the following URL:

```
http://localhost:8080/systinet/platform/rest/location
```

If you use a browser, this opens a view which enables you to browse the data content and interact with it.

Note: The view of a collection location only displays resources that you have permissions for.

Systinet publisher creates a collection within the publishing location when you upload data content. For more details, see "How to Publish Content" in the *User Guide*.

Open a collection by clicking its name, or download a zip file of its content by clicking **Download as Archive**. At the lowest level, the browser shows the actual data content. For the actual content, click the content name.

Click **Advanced View** to open the detail view of the related artifact in Systinet. For details, see "Artifact Detail Page" in the User Guide.

You can change the output of the location space on your browser using alternative media types:

- `http://hostname:port/context/platform/rest/location`

  The default output as described above.

- `http://hostname:port/context/platform/rest/location?alt=text/html`

  The HTML representation which is the default output for locations. For artifacts with non-HTML content there is no HTML representation.

- `http://hostname:port/context/platform/rest/location/foo?alt=application/zip`

  Output all files from a particular collection (foo) to a zip archive.

  Add the following optional switches to output additional related documentation:

- `&inline-desc`

  Includes document descriptor files in the archive (files with the .desc suffix in .meta subdirectories).

- `&inline-acl`

  Includes ACL files in the archive (files with the .acl suffix in .meta directories).

- `&zip-compat`

  Enable zip compatibility mode (no directory entries are created in the archive).

- `http://hostname:port/context/platform/rest/location/test?alt=application/atom%2bxml`

  View the Atom feed for a collection location.

- `http://hostname:port/context/platform/rest/location/foo?alt=application/json`

  Output a particular collection location as a JSon representation.

By default, the last revision of a resource or collection is shown, but you can request revisions from a particular date using the following pattern:

`http://hostname:port/context/platform/rest/location;datetime=[datetimeValue]`

For example, `http://hostname:port/context/platform/rest/location/foo/a.wsdl`, corresponds to the last revision of a the `a.wsdl` resource in the `foo` location.

`http://hostname:port/context/platform/rest/location;datetime=2008-01-01T12:00:00.000Z/foo/a.wsdl`, corresponds to the revision of the `a.wsdl` resource at 12:00 on 1/1/2008.

Specifying a collection location that does not exist returns an exception.

Artifact Relationships Feed

You can view the relationships of an artifact as a feed.

For example, to view the comments feed of a WSDL artifact, use the URL:

`http://host:port/context/platform/rest/artifact/wsdlArtifact/`*`UUID`*`/relation`

The feed returns both incoming and outgoing relationships to/from the artifact. The content shows a proprietary representation of the relationship, with the related artifact available by following the 'alternate' link.

If the related artifact is readable by the current client identity, its name is displayed, otherwise only its UUID is shown.

Artifact History Feed

You can view the revision history of an artifact as a feed.

For example, to view the revision history of `my.wsdl`, use the URL:

`http://host:port/context/platform/rest/artifact/wsdlArtifact/my.wsdl/history`

Artifact Comments Feed

You can view the comments made about an artifact as a feed.

For example, to view the comments feed of a WSDL artifact, use the URL:

`http://host:port/context/platform/rest/artifact/wsdlArtifact/`*`UUID`*`/comments`

Full Text Search

Full text search can be run in an SDM collection feed.

Add `?fulltext=SEACHEDTEXT` to the collection feed URL to perform full text search.

For example, to search for the text "lifecycle" in all artifacts:

`http://host:port/context/platform/rest/artifact/artifactBase?fulltext=lifecycle`

`Feed Ordering` and `Feed Paging` can be also applied to the result.

Full text search result can be only ordered by `relevance`, `name` or `timestamp`.

Default ordering is `relevance-,name`.

A new parameter "`lightsearch`" is used to speed up performance of search function in some cases. It affects Full Text Search only.

The `lightsearch=true` is valid only for FTS whereas default FTS values are combined with property based search results and the performance improve is significant.

Note: Full text search must be enabled in the database, for more details see the *Installation Guide*.

Entries

The detailed information about an artifact in the repository is available as an entry.

Entries are described in the following sections:

- "Artifact Atom Entries" below
- "Artifact History Entries" on page 25
- "Atom Entry Property Descriptors" on page 25
- "Artifact Data" on page 29
- "Resource Identification" on page 30

Artifact Atom Entries

The information about each entry in the collection feed is only a summary. Each entry can be accessed directly using its `self link` as referenced in the artifact feed, which is formed from either its `restName` or `id`.

For example, you can access a particular user profile entry with URL:

`http://localhost:port/context/platform/rest/artifact/personArtifact/admin`

Admin User Profile Entry

```
<entry xml:base=
    "http://localhost:8180/systinet/platform/restSecure/artifact/personArtifact"
    xmlns="http://www.w3.org/2005/Atom">
  <id>urn:hp.com:2009:02:systinet:platform:artifact:d82a5dcc-d85c-4766-9967-
93eb5dc0bd0a</id>
  <updated>2009-06-01T09:30:23.154+02:00</updated>
  <title type="text" xml:lang="en">HP SOA Administrator</title>
  <summary type="text" xml:lang="en">HP SOA Administrator.</summary>
  <link href="personArtifact/d82a5dcc-d85c-4766-9967-
93eb5dc0bd0a?alt=application%2Fatom%2Bxml"
      type="application/atom+xml" rel="self" title="artifact detail"/>
  <link href="personArtifact/d82a5dcc-d85c-4766-9967-
93eb5dc0bd0a?alt=application%2Fxml"
      type="application/xml" rel="alternate" title="XML representation"/>
  <link href="personArtifact/d82a5dcc-d85c-4766-9967-
93eb5dc0bd0a?alt=application%2Fatom%2Bxml"
      type="application/atom+xml"
      rel="urn:hp.com:2009:02:systinet:platform:artifact:last-revision"
      title="last revision"/>
  <link href="personArtifact" type="application/atom+xml;type=feed"
      rel="urn:hp.com:2009:02:systinet:platform:artifacts:collection"
      title="sdm feed"/>
  <link href="personArtifact/d82a5dcc-d85c-4766-9967-93eb5dc0bd0a/history"
      type="application/atom+xml;type=feed"
      rel="urn:hp.com:2009:02:systinet:platform:artifact:history"
      title="history feed"/>
  <link href="personArtifact/d82a5dcc-d85c-4766-9967-93eb5dc0bd0a/acl"
      type="application/xml"
      rel="urn:hp.com:2009:02:systinet:platform:artifact:acl"
      title="access control list"/>
  <link href="personArtifact/d82a5dcc-d85c-4766-9967-93eb5dc0bd0a?alt=text%2Fhtml"
      type="text/html" rel="alternate" title="UI view page"/>
  <author>
    <name>systinet:admin</name>
  </author>
  <category label="Active"
      scheme="uddi:systinet.com:soa:model:taxonomies:accountStates:accountState"
      term="S1"/>
  <category label="Artifact"
      scheme="uddi:systinet.com:soa:model:taxonomies:artifactTypes:_artifactType"
      term="urn:com:systinet:soa:model:artifacts"/>
  <category label="Content"
      scheme="uddi:systinet.com:soa:model:taxonomies:artifactTypes:_artifactType"
      term="urn:com:systinet:soa:model:artifacts:content"
      ext:parent="urn:com:systinet:soa:model:artifacts"
      xmlns:ext="http://schemas.hp.com/2008/symphony/atom/extensions"/>
  <category label="Contact"
      scheme="uddi:systinet.com:soa:model:taxonomies:artifactTypes:_artifactType"
      term="urn:com:systinet:soa:model:artifacts:content:contact"
      ext:parent="urn:com:systinet:soa:model:artifacts:content"
      xmlns:ext="http://schemas.hp.com/2008/symphony/atom/extensions"/>
  <category label="User Profile"
      scheme="uddi:systinet.com:soa:model:taxonomies:artifactTypes:_artifactType"
      term="urn:com:systinet:soa:model:artifacts:content:contact:person"
      ext:parent="urn:com:systinet:soa:model:artifacts:content:contact"
      xmlns:ext="http://schemas.hp.com/2008/symphony/atom/extensions"/>
  <content type="application/xml">
    ...
  </content>
</entry>
```

Each artifact entry consists of the following descriptors:

Descriptor	Description
`id`	A unique id for the artifact (uuid).
`updated`	The last update time.
`title`	The name of the entry.

Descriptor	Description
`link`	A set of links with the following link types indicated by the `rel` attribute: • `self` The atom entry details. • `urn:hp.com:2009:02:systinet:platform:artifacts:collection` The associated artifact collection feed. For details, see "Artifact Collection Feeds" on page 13. • `urn:hp.com:2009:02:systinet:platform:artifact:last-revision` The last revision of this artifact. • `edit-media` The associated data content for an artifact. • `urn:hp.com:2009:02:systinet:platform:artifact:history` The collection feed for revisions of this artifact. • `alternate` A set of alternate views of the artifact, including: ■ `application/xml` The bare XML representation of the `content` descriptor. ■ `text/html` Points to the Systinet UI view of the artifact. • `related` Links to related artifacts. **Note:** Related artifacts may also be linked where the link has the `rel` attribute with a specific relationship name. For details, see "Relationship Properties Atom Representation" on page 28.
`category`	A set of taxonomic values from: • Taxonomy property values • categoryBag and identifierBag • sdmTypes taxonomy values
`author`	The creator of this revision of the artifact.

Descriptor	Description
`content`	The bare XML representation of the content descriptor. For details, see "Atom Entry Property Descriptors" below.
`summary`	An artifact description.

Artifact History Entries

By default, entries display the latest revision. You can view older revisions by adding `;rev=X` to the entry URL.

For example, the first revision of a WSDL can be obtained with the URL:

```
https://host:port/context/platform/rest/artifact/wsdlArtifacts/mywsdl;rev=1
```

Atom Entry Property Descriptors

Atom entries contains an XML representation of an artifact in the `content` descriptor.

Admin User Entry Content

```
<content type="application/xml">
  <art:artifact name="personArtifact" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:p="http://hp.com/2008/02/systinet/platform/model/property"
      xmlns:sdm="http://hp.com/2007/10/systinet/platform/model/propertyType"
      xmlns:art="http://hp.com/2008/02/systinet/platform/model/artifact">
    <p:primaryGroup xsi:nil="true" sdm:type="text"/>
    <p:accountState name="Active"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:accountStates"
value="S1"
        sdm:type="category"/>
    <p:designTimePolicy xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:documentation xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:_uuid sdm:type="uuid">d82a5dcc-d85c-4766-9967-93eb5dc0bd0a</p:_uuid>
    <p:_revision sdm:type="integer">1</p:_revision>
    <p:_checksum sdm:type="bigInteger">0</p:_checksum>
    <p:_contentType xsi:nil="true" sdm:type="text"/>
    <p:_revisionTimestamp sdm:type="date">2009-06-01T07:30:23.154Z</p:_
revisionTimestamp>
    <p:keyword xsi:nil="true" sdm:type="category" p:multi="true"/>
    <p:categoryBag xsi:nil="true" sdm:type="categoryBag"/>
    <p:_revisionCreator sdm:type="text">systinet:admin</p:_revisionCreator>
    <p:_artifactType name="Artifact"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:artifactTypes"
        value="urn:com:systinet:soa:model:artifacts" sdm:type="category"
p:multi="true"/>
    <p:_artifactType name="Content"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:artifactTypes"
        value="urn:com:systinet:soa:model:artifacts:content" sdm:type="category"
p:multi="true"/>
    <p:_artifactType name="Contact"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:artifactTypes"
        value="urn:com:systinet:soa:model:artifacts:content:contact"
sdm:type="category"
        p:multi="true"/>
    <p:_artifactType name="User Profile"
        taxonomyUri="uddi:systinet.com:soa:model:taxonomies:artifactTypes"
        value="urn:com:systinet:soa:model:artifacts:content:contact:person"
sdm:type="category"
        p:multi="true"/>
    <p:identifierBag xsi:nil="true" sdm:type="identifierBag"/>
    <p:description sdm:type="text">HP SYSTINET Administrator.</p:description>
    <p:_owner sdm:type="text">admin</p:_owner>
    <p:_deleted sdm:type="boolean">false</p:_deleted>
    <p:name sdm:type="text">HP SOA Administrator</p:name>
    <p:consumptionContract xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:consumptionRequest xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:r_consumerOwner2contract xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:provides xsi:nil="true" sdm:type="documentRelationship" p:multi="true"/>
    <p:contactRole xsi:nil="true" sdm:type="category" p:multi="true"/>
    <p:r_contactClassification xsi:nil="true" sdm:type="category"/>
    <p:geographicalLocation xsi:nil="true" sdm:type="category" p:multi="true"/>
    <p:languageCode xsi:nil="true" sdm:type="category"/>
    <p:hpsoaApplicationContact xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
    <p:r_memberOf xsi:nil="true" sdm:type="documentRelationship" p:multi="true"/>
    <p:loginName sdm:type="text">admin</p:loginName>
    <p:address xsi:nil="true" sdm:type="address"/>
    <p:email sdm:type="text" p:multi="true">admin@comp.com</p:email>
    <p:phone xsi:nil="true" sdm:type="text" p:multi="true"/>
    <p:instantMessenger xsi:nil="true" sdm:type="text" p:multi="true"/>
    <p:externalDefinition xsi:nil="true" sdm:type="documentRelationship"
p:multi="true"/>
  </art:artifact>
</content>
```

The content is effectively a list of the properties of an artifact.

The property types are described in the following sections:

- "Primitive Properties Atom Representation" below
- "Category Properties Atom Representation" below
- "Relationship Properties Atom Representation" on the next page
- "Special Properties Atom Representation" on page 29

Primitive Properties Atom Representation

Primitive properties are represented as follows:

`<p:NAMEsdm:type="TYPE">VALUE<p:NAME>`

The following primitive property types use this form:

Property Type	xsi:type Correspondance
date	xs:dateTime
boolean	xs:boolean
double	xs:double
integer	xs:int
bigInteger	xs:integer
text	xs:string
uuid	xs:string

For example:

`<p:phone sdm:type="text">774 789 784</p:phone>`

Category Properties Atom Representation

Category properties are propagated in two places in the Atom entries.

The `category` descriptor, which also appears in collection feeds, describes the taxonomy and category as follows:

`<category label="..." scheme="..." term="..."/>`

- `label` corresponds to the category name.
- `scheme` corresponds to the taxonomy URI combined with the property name.
- `term` corresponds to the category URI.

This is reproduced in the entry `content` as a property:

```
<p:NAME name="..." taxonomyUri="..." value="..." sdm:type="category"/>
```

For example, a web service with Failure Impact set to High is represented as a property in the entry for the web service:

```
<p:criticality name="High"
taxonomyUri="uddi:systinet.com:soa:model:taxonomies:impactLevel".
value="uddi:systinet.com:soa:model:taxonomies:impactLevel:high"
sdm:type="category"/>
```

Note that the property representing this taxonomic category is `criticality`.

The property is propagated to Atom metadata as an `atom:category` element:

```
<atom:category label="High"
scheme="uddi:systinet.com:soa:model:taxonomies:impactLevel:criticality"
term="uddi:systinet.com:soa:model:taxonomies:impactLevel:high"/ >
```

Relationship Properties Atom Representation

Relationship properties are propagated in two places in the Atom entry.

In feeds the `link` exists as metadata.

The `link` descriptor describes the following link types:

- A generic `related` link.
- A specific relationship bound link where the `rel` attribute uses a `'urn:hp.com:2009:02:systinet:platform:artifact:relation:`prefix with the relationship name.

In entries, relationships are described as a set of `property` atom content descriptors:

Relationship Properties

```
Incoming relationship example:
<p:inBusinessService xlink:href="businessServiceArtifact/1210"
    sdm:type="documentRelationship" p:multi="true">
  <t:source>c519d961-03b3-4303-b61b-8809b945b7ae</t:source>
  <t:exact>false</t:exact>
</p:inBusinessService>

Exact incoming:
<p:inBusinessService xlin:href="businessServiceArtifact/1210"
    sdm:type="documentRelationship" p:multi="true">
  <t:source>c519d961-03b3-4303-b61b-8809b945b7ae</t:source>
  <t:exact>true</t:exact>
</p:inBusinessService>

Outgoing relationship example:
<p:service xlin:href="webServiceArtifact/5"
```

```
    sdm:type="documentRelationship" p:multi="true">
  <t:target deleted="false">5a4aeca7-a8f9-4761-b504-82723ab2f417</t:target>
</p:service>

Exact outgoing:
<p:service xlin:href="xmlServiceArtifact/101.xml;rev=1"
    sdm:type="documentRelationship" p:multi="true">
  <t:target revision="1" deleted="false">72ab6f1f-e943-4fd2-a7bc-
5d227e6e134a</t:target>
</p:service>
```

Special Properties Atom Representation

Special properties are defined by an XML schema which determines their structure.

Systinet contains an XML schema which defines the following property types:

- address
- categoryBag
- identifierBag
- dailyInterval
- nameURLPair
- nameValuePair
- parameterList (XQuery parameter)
- scheduled
- selector

Artifact Data

If an artifact has associated data content, then you can directly access the data content.

For example, a WSDL artifact is usually associated with the actual WSDL file.

Access the WSDL entry with the URL:

`https://localhost:8443/systinet/platform/rest/artifact/wsdlArtifact/mywsdl?alt=atom`

WSDL Entry

```
<entry
xml:base="http://localhost:8180/systinet/platform/restSecure/artifact/wsdlArtifact"
    xmlns="http://www.w3.org/2005/Atom">
```

```
  <id>urn:hp.com:2009:02:systinet:platform:artifact:f5aff3eb-95fd-4791-856b-
3ac551666da2</id>
  <updated>2009-06-08T16:24:55.609+02:00</updated>
  <title type="text" xml:lang="en">mywsdl</title>
  ...
  <link href="../location/wsdls/mywsdl.wsdl" type="application/xml" rel="edit-
media" title="attached data" />
  ...
</entry>
```

The entry contains a link pointing to the locations workspace. The data is also available using a `/data` suffix.

For example,
`https://localhost:8443/systinet/platform/rest/artifact/wsdlArtifact/mywsdl/data`

You can also access older revisions of the data with the URL:

`https://localhost:8443/systinet/platform/rest/artifact/wsdlArtifact/mywsdl;rev=1/data`

Caution: Using any relative references in the XML data will probably cause an error because they are resolved relatively to the GET context. Use the `location` context to navigate references instead.

Resource Identification

A web service artifact with uuid `65a2b119-9a6b-491e-8353-3692f4b9e3e5` and name `MyService` is available in the artifacts collection:

`http://localhost:port/context/systinet/platform/rest/artifact/`

At the following locations:

- artifactBase/65a2b119-9a6b-491e-8353-3692f4b9e3e5
- implementation/65a2b119-9a6b-491e-8353-3692f4b9e3e5
- webServiceArtifact/65a2b119-9a6b-491e-8353-3692f4b9e3e5

These URLs are not user-friendly. For newly created artifacts, Systinetauto-generates a REST name which in most cases is more user-friendly than the uuid.

This REST name can be used instead of the uuid in the URL.

`http://localhost:port/context/systinet/platform/rest/artifact/webServiceArtifact/MyService`

Note: If you migrate or federate resources (for example, with UDDI Registry import/export), the user-friendly URLs are lost.

User-friendly REST names remain the same, even if you change the artifact name.

Category Documents

Atom categories are a way to categorize large amounts of data. The permitted values in Atom categories can be either fixed or unrestricted. Category documents group permitted category values.

An example of a category group with a fixed set of values is the impact level criticality category group.

```
http://host:port/context/platform/rest/category-
document/uddi:systinet.com:soa:model:taxonomies:impactLevel:criticality
```

Impact Criticality Category Document

```
<?xml version="1.0" encoding="UTF-8"?>
<app:categories xmlns:app="http://www.w3.org/2007/app"
xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns:hp="http://hp.com/2008/02/systinet/platform/model/taxonomy"
    xmlns:v355tax="http://systinet.com/uddi/taxonomy/v3/5.5"
    xmlns:v350tax="http://systinet.com/uddi/taxonomy/v3/5.0" fixed="yes"
    scheme="uddi:systinet.com:soa:model:taxonomies:impactLevel:criticality">
  <atom:category term="uddi:systinet.com:soa:model:taxonomies:impactLevel:high"
label="High"/>
  <atom:category term="uddi:systinet.com:soa:model:taxonomies:impactLevel:medium"
label="Medium"/>
  <atom:category term="uddi:systinet.com:soa:model:taxonomies:impactLevel:low"
label="Low"/>
</app:categories>
```

Systinet uses taxonomies, which are an abstraction almost identical to Atom categories. These taxonomies are sometimes transferable to Atom category documents, which can be referenced from the service document.

The categories in the taxonomy then appear as Atom categories, corresponding to the taxonomy values in artifact entries and feeds.

Atom REST Operations

To use the Atom REST interface, applications must map each operation to an HTTP request. For details, see Summary of Atom REST Operations.

Summary of Atom REST Operations

REST Operation	HTTP method	Query Field	Notes
"CREATE" on the next page	POST	`create`	The path specifies the containing collection and the POST body contains an XML representation of the artifact to create.

Summary of Atom REST Operations, continued

REST Operation	HTTP method	Query Field	Notes
GET	GET	None	Obtains the requested resources. For details, see "Feeds" on page 13 and "Entries" on page 21.
"UPDATE" below	PUT	`update`	Updates the specified resource.
"DELETE" on the next page	DELETE	`delete`	Deletes the specified resource. GET, UNDELETE, and PURGE operations can be run on deleted resources.
"UNDELETE" on the next page	POST	`undelete`	Undeletes the deleted resource. It can then be updated again.
"PURGE" on the next page	DELETE	`purge`	Purge physically removes a resource.

Note: All writable operations use a proprietary XML representation for POST and PUT operations.

CREATE

Implemented by processing a POST request to the artifact type collection space. The POST body contains a valid XML representation of the new artifact.

```
POST
http://localhost:8080/systinet/platform/restSecure/artifact/businessServiceArtifact
```

The content of the XML representation should match an artifact Atom entry. For details, see "Artifact Atom Entries" on page 22.

You can create artifacts conditionally using CREATE with Etags. For details, see "Atom REST ETags" on the next page.

Note: Since this operation requires an HTTP POST request, you cannot simply enter the URL into a browser. Typically the request is coded in an application. It is possible to use Javascript or HTTP command line clients.

UPDATE

Implemented by processing a PUT request to the specified collection and artifact identified with its UUID. The updated content is contained in the XML representation. For details, see "Artifact Atom Entries" on page 22.

```
PUT http://localhost:8080/systinet/platform/restSecure/artifact/
```

```
businessServiceArtifact/002374c1-3500-43ea-92a7-02322bdf6002
```

Note: Since this operation requires an HTTP PUT request, you cannot simply enter the URL into a browser. Typically the request is coded in an application. It is possible to use Javascript or HTTP command line clients.

DELETE

Implemented by sending a DELETE request to the specified collection and artifact identified using its UUID.

```
DELETE http://localhost:8080/systinet/platform/restSecure/artifact/
businessServiceArtifact/002374c1-3500-43ea-92a7-02322bdf6002
```

UNDELETE

Implemented by sending an empty POST request to the specific collection and deleted artifact identified using its UUID. There is no XML representation associated with the POST operation for UNDELETE.

```
POST http://localhost:8080/systinet/platform/restSecure/artifact/
businessServiceArtifact/002374c1-3500-43ea-92a7-02322bdf6002
```

PURGE

Implemented by sending a DELETE request to the specific collection and artifact identified by its UUID and its history feed URI

Caution: This operation cannot be undone.

```
DELETE http://localhost:8080/systinet/platform/restSecure/artifact/
businessServiceArtifact/002374c1-3500-43ea-92a7-02322bdf6002/history
```

Atom REST ETags

ETags enable you to perform GET, PUT, and POST operations using conditions. For example, you can use ETags to compare a response to a previously cached response to see if there are any changes to the requested resource.

Note: Using ETags requires a REST client in order to specify the parameters.

You can use both *weak* and *strong* ETags.

Weak ETags are implemented by comparing the last modified time of an artifact in the repository with the time from HTTP header attributes: `If-Modified-Since` and `If-Unmodified-Since`.

Strong ETags are used mainly for caching purposes when weak ETags based on timestamps are not sufficient. For example, when an artifact has not been modified but its representation has. This happens when there is a new, changed, or missing incoming relation. ETags are random hash-generated with every artifact update.

Use ETags as described in the following topics:

- "Conditional GET" below
- "Conditional PUT and POST" below

Conditional GET

You can apply a conditional GET to determine whether a resource has changed, and then only return the representation if there is a change.

You can use a weak ETag specifying a time or a strong ETag specifying the tag attribute used to identify the revision.

Specify the time using the *If-Modified-Since* header parameter in the HTTP request.

This time is compared to the *Last Modified* attribute in the response. The *Last Modified* attribute is always returned and can be stored for future reference.

If cases where timestamps are not sufficient, you can use ETags to compare entry or feed revisions.

Specify the ETag value using the *If-None-Match* header parameter in the HTTP request.

This speoifed ETag is compared to the *ETag* attribute in the response. The *ETag* attribute is always returned and can be stored for future reference.

If the artifact has not changed, then an HTTP standard non-modified response is created with a 304 status code and proper headers are returned.

If a header parameter is not specified the latest representation is always returned.

Conditional PUT and POST

You can apply a conditional PUT or POST to determine whether a resource has changed compared to the revision you are updating, and then only apply your update if there is no change.

You can use a weak ETag specifying a time or a strong ETag specifying the tag attribute used to identify the revision.

Specify the time using the *If-Unmodified-Since* header parameter in the HTTP request.

This time is compared to the *Last Modified* attribute in the response. The *Last Modified* attribute is always returned and can be stored for future reference.

In cases where timestamps are not sufficient, you can use ETags to compare entry or feed revisions to determine whether a resource has changed compared to the revision you are updating, and then only apply your update if there is no change.

Specify the ETag value using the *If-Match* header parameter in the HTTP request.

This speoifed ETag is compared to the *ETag* attribute in the response. The *ETag* attribute is always returned and can be stored for future reference.

If the artifact has changed, then an HTTP standard preconditions-failed response is created with a 412 status code and proper headers are returned.

If a header parameter is not specified your update is applied regardless of any other changes.

Atom REST Client

The Atom REST client is an untyped API to manipulate artifacts in the repository. It is a thin layer above the Atom REST Interface.

The client provides the following features:

Model Introspection

- Enumerate Artifact types
- Enumerate Artifact properties

CRUD

- Local operations:
  - Create Artifact instance
- Server Operations
  - Create Artifact
  - Get Artifact
  - Get Artifact Data
  - Update Artifact
  - Update Artifact Data
  - Delete Artifact
  - Purge Artifact

Search

- Search criteria - name-value pairs, same property names are "ORed"
- Lists Artifacts - initialized properties depend on the given artifact type. For example, ArtifactBase has only name, description, categoryBag,...
- Pagination and ordering is supported .

Classpath

JAR files are mixed with others in the installation `client/lib` folder.

- abdera-client-1.0.jar
- abdera-core-1.0.jar
- abdera-i18n-1.0.jar
- abdera-parser-1.0.jar
- axiom-api-1.2.5.jar
- axiom-impl-1.2.5.jar
- common-lang.jar
- commons-codec-1.3.jar
- commons-httpclient-3.1.jar
- commons-lang-2.3.jar
- commons-logging-1.1.jar
- jaxen-full-2.51.jar
- localization-1.0.0-alpha-3.jar
- pl-model-api.jar
- pl-model-impl.jar
- pl-remote-client.jar
- pl-remote-model.jar
- pl-xml-serialization.jar
- pl-xmlbeans-sdmconfig.jar
- pl-xmlbeans-serialization.jar
- saxpath-1.0-FCS.jar
- security.jar
- xmlbeans-2.3.0-patch.hp-3.jar

First Steps

This section provides code extracts that demonstrate working with the API. For more examples, see "Demos" on the next page and the Javadocs at `http://`*`host:port`*`/hp-systinet-doc/doc/api/index.html`.

1. Create a new RepositoryClient instance:

```
RepositoryClient repositoryClient =
  RepositoryClientFactory.createRepositoryClient

("http://localhost:8080/systinet",
  "demouser", "changeit", false, null, 0);
```

2. Create a new webService artifact instance and set its name:

```
ArtifactBase webService =
  repositoryClient.getArtifactFactory().newArtifact

("webServiceArtifact");
  webService.setName("Demo Webservice Name");
```

3. Store the instance on the server:

```
webService = repositoryClient.createArtifact(webService);
```

4. Get the instance from server:

```
webService = repositoryClient.getArtifact(webService.get_uuid().toString());
```

Important Classes

- **Javadoc** documentation is located at `SYSTINET_HOME/doc/api ([host]:[port]/hp-systinet-doc/doc/api/index.html)`.
- **SDM Model** documentation is located at `SYSTINET_HOME/doc/sdm ([host]:[port]/hp-systinet-doc/doc/sdm/index.html)`.
- **RepositoryClientFactory**
  - Factory used to create RepositoryClient instances.
  - The factory supports:
    - SDM Model Caching - the parameter means that the factory loads the model from the server if the cached version is older than the passed value.
    - Custom authentication (custom Abdera client factory) - see

https://cwiki.apache.org/ABDERA/client.html for more information.

    - Switching off server certificate validation when using HTTPS.

- **RepositoryClient**
    - This interface contains all the important methods and getters for supporting classes.
- **ArtifactBase**
    - To get/set a particular part of an artifact use either the get or set methods.
    - Common abstraction for the untyped view of any artifact in Service Definition Model (SDM).
- **ArtifactData** - Artifact data holder.
- **ArtifactFactory** - Factory for creating artifact instances.
- **ArtifactRegistry** - Registry of defined artifacts.
    - **ArtifactDescriptor** - Introspective info about an artifact.
    - **PropertyDescriptor** - Introspective info about an artifact's property.
- **ValuesFactory**
    - Able to create MultiplePropertyValues, Uuid, and ArtifactData.
    - Creates instances of single property values from given values.
- **PropertiesUtil**
    - Various static helper functions for manipulating properties.

Demos

The following demos provide more code examples:

- "Atom REST Client Demo" below

Atom REST Client Demo

The purpose of this demo is to introduce the Atom REST Java client and to show how to interact with Systinet using this client. The basic operations CREATE, UPDATE, DELETE, UNDELETE, PURGE, GET, search, and mupports JDK 1.7, odel introspection are demonstrated.

1. Enumerate artifact types and service properties (`enumerateArtifactsAndProperties` method).
2. Create web service artifact and business service artifact with relation to that web service (`createGetUpdateDelete` method).
3. Create service and search that service by criticality (`createSearchDelete` method).

You can find the demo source code in: `SYSTINET_HOME\demos\client\rest\src`

To run the REST API demo:

1. Ensure that the demo is properly configured and Systinet is running.
2. Change your working directory to: `SYSTINET_HOME\demos\client\rest`
3. To get help, execute: `run`
4. To build the demo, execute: `run make`
5. To run the demo, execute: `run run`

To rebuild the demo, execute `run clean` to delete the classes directory and `run make` to rebuild the demo classes.

Chapter 3: IDE Integration

This chapter explains how to allow IDEs to access the Systinet repository.

It contains the following sections:

- "Systinet IDE Integrations" below

  Introduces the plug-ins HP Software provides for development environments.

- "WSIL Report – IBM RAD and Eclipse" below

  How to use the WSIL query include with Systinet to add it to an IDE.

- "Microsoft Visual Studio" on the next page

  How to add Systinet as a Web Reference in MS Visual Studio.

Systinet IDE Integrations

HP Software provides a set of plug-ins for IDEs that embed Systinet functionality in each development environment.

HP Software provide the following IDE integration products:

- **Systinet Plugin for Eclipse**

  Enables you to search the Catalog, generate service clients and skeletons from Systinet resources, perform local resource validation against Systinet policies, and publish local resources to the Catalog. You can also make contract and lifecycle approval requests and use the Navigator feature from within Eclipse.

- **Systinet Plugin for Visual Studio**

  Enables you to search the Catalog , generate web references from Systinet resources, and publish local resources to the Catalog. You can also make contract and lifecycle approval requests and use the Navigator feature from within Visual Studio.

For details, see *Plugin for Eclipse* and *Plugin for Visual Studio*.

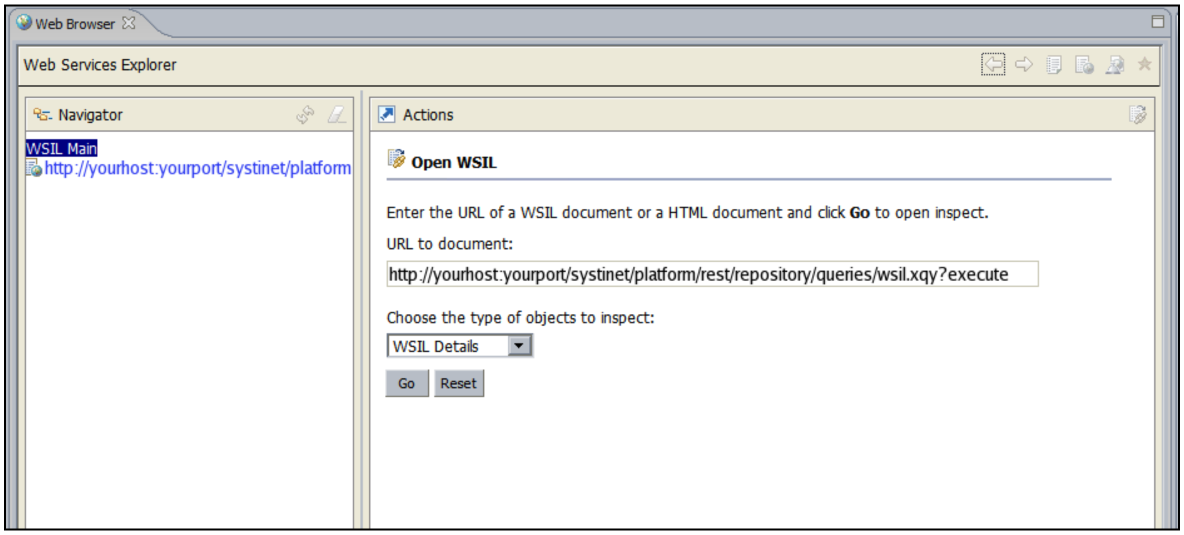
WSIL Report – IBM RAD and Eclipse

A WSIL (Web Service Inspection Language) dynamic query is included to make it easy for IDEs, like IBM RAD, to leverage the Systinet repository. This query provides a list of all web services and their

WSDLs and is used by RAD to create a service proxy. You can access this query from the Tools tab menu Generate WSIL Document, or at the referenced location:

```
http://yourhost:yourport/systinet/platform/restBasic/service/system/wsil
```

Launch IBM RAD 6.0's Web Services Explorer, and enter the WSIL report URL (the page that is generated by the WSIL link of Search.



Enter the URL of a WSIL document or a HTML document and click **Go** to open inspect.

URL to document:

http://yourhost:yourport/systinet/platform/rest/repository/queries/wsil.xqy?execute

Choose the type of objects to inspect:

WSIL Details

From there, you can access the services WSDL documents.

Microsoft Visual Studio

The **Add Web Reference** facility of Microsoft Visual Studio’s Solution Explorer is fully supported.

Enter the URL of your Systinet installation (for example, `http://yourserver:8080/systinet/`) to access Systinet within Microsoft Visual Studio.

Notice the instructions from Microsoft Visual Studio at the top. In this case, you are navigating to a WSDL file stored in Systinet. On the right, you can see that Microsoft Visual Studio does not recognize web service discovery information on the current page.

To find the service you are looking for, see "How to Search the Catalog" in the *User Guide*.

Select the WSDL artifact for the service to open the WSDL detail page.

From this page copy the link in the Data section and paste it to the Microsoft Visual Studio Solution Explorer. You can now click **Add Reference** to read the web service definition(s) into Microsoft Visual Studio.

Chapter 4: Lifecycle Remote Client

The Lifecycle Remote Client enables you to remotely manipulate lifecycle processes and manage the governance data of artifacts.

Process Management

Designed for administration of lifecycle processes remotely.

All the functionality is accessible using service `ProcessManagementService`.

An instance of `ProcessManagementService` can be created using method `createProcessManagementService()` on `GovernanceServiceFactory`

For example:

```
ProcessManagementService
service = GovernanceServiceFactory.createProcessManagementService
    ("http://localhost:8080/systinet","admin","changeit",true)
```

It contains methods for reading, creating, removing, copying, publishing, and editing lifecycle processes.

For more details, see the javadoc for `ProcessManagementService`.

Process information and process editing does not support all features.

Artifact Governance

Designed to manage governance of the artifact remotely.

All the functionality is accessible using service `ArtifactGovernanceService`.

An instance of `ArtifactGovernanceService` can be created using method `createArtifactGovernanceService ()` on `GovernanceServiceFactory`

For example:

```
ArtifactGovernanceService
service = GovernanceServiceFactory.createArtifactGovernanceService
    ("http://localhost:8080/systinet","admin","changeit",true)
```

It contains methods for the following:

- Starting governance
- Ending governance

- Changing: process, stage, or process stage and approval
- Approving the requests
- Getting:
  - Governance status for a list of UUIDs or for a governance tree identified by root artifact UUID,
  - Current stage history record
  - Voting status
  - Voting details
  - Artifacts on which voting is enabled
  - Policies and last known validation status
  - Tasks
  - StageHistoryRecords for a single artifact
  - Request approval for a root artifact UUID
- Canceling approval
- Marking a task as complete
- Voting

It also contains support for searching governed artifacts using the following methods:

- By Governance Process UUID
- By Current Stage
- By Last Approved Stage
- By Type
- By Lifecycle Status
- Conditions are always combined together

It always returns governance records. For more details see the javadoc for `ArtifactGovernanceService`.

Classpath

JAR files are mixed with others in client/lib folder.

- lifecycle-remote-api.jar
- hessian-3.1.6-patch.hp-2.jar

First Steps

This section provides code extracts that demonstrate working with the API. For more examples, see the Javadocs at `http://`*`host:port`*`/hp-systinet-doc/doc/api/index.html`.

1. Create new Artifact Governance Service instance

```
ArtifactGovernanceService
service=GovernanceServiceFactory.createArtifactGovernanceService
    ("http://localhost:8080/systinet","admin","changeit",true);
```

2. Get Governance Status of an artifact

```
String artifactUuid=...
GovernanceStatus record = service.getGovernanceStatus(artifactUuid);
```

3. Request approval

```
service.requestApproval(artifactUuid,"Requesting approval.");
```

4. Get Stage History Record and iterate over approvals

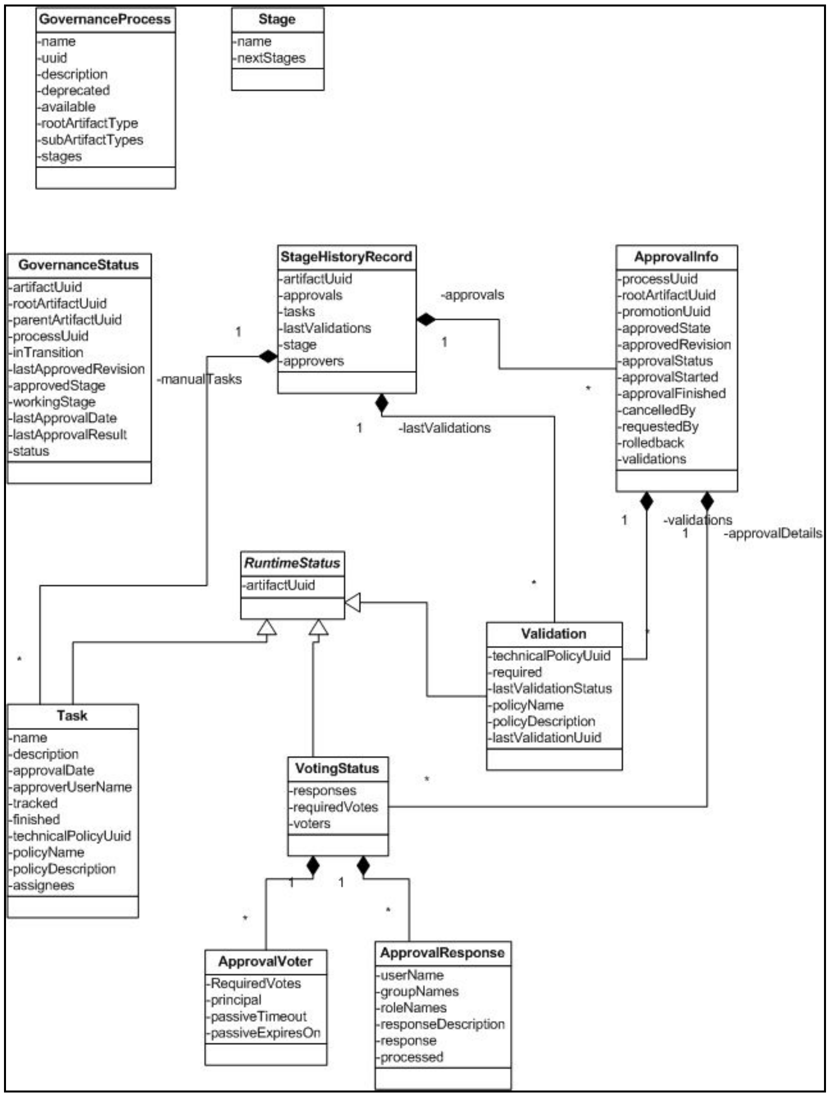
```
StageHistoryRecord
historyRecord = service.getCurrentStageHistoryRecord(artifactUuid);
for (ApprovalInfo ar : historyRecord.getApprovals()) {
  ...
}
```

Important Classes

- **Javadoc** documentation is located at `SYSTINET_HOME/doc/api` (`[host]:[port]/hp-systinet-doc/doc/api/index.html`).
- **GovernanceServiceFactory** - Factory that creates services.
- **ArtifactGovernanceService** - Service for getting governance details as well as managing governance of artifacts.
- **ProcessManagementService** - Service for managing governance process.

There is a demo available that provides some code examples at `SYSTINET_HOME/demos/client/lifecycle`.

Data Structure Diagram



Chapter 5: Executable Objects

In Systinet you can execute task artifacts remotely by accessing a proprietary remote endpoint representing the task artifact.

A document is executed by a request containing an execute parameter. The result of an execution of a task artifact is the resulting report document.

An example of a task execution is the URL to use to execute the Recycle Bin Cleaner Task:

```
http://localhost:8080/systinet/platform/restBasic/
repository/taskArtifacts/recycleBinCleanerTask?execute
```

The output of such a task execution is an XML representation of the report.

Using DQL

The DQL query language provides a simple query solution for the SOA Definition Model (SDM). It enables you to query all aspects of the model – artifacts, properties, relationships, governance, and compliance.

This chapter describes DQL in the following sections:

- "Introduction to DQL" below
- "DQL Reference" on page 56
- "DQL with Third-Party Products" on page 66

Introduction to DQL

DQL is an SQL-like language that enables you to query the repository of artifacts in Systinet defined by the SDM model. DQL preserves SQL grammar, but uses artifacts instead of tables, and artifact properties instead of table columns. As DQL is based on SQL you can apply your SQL knowledge to DQL.

A simple example is to return the name and description of all business service artifacts.

```
select name, description
  from businessServiceArtifact
```

In Systinet, you can use DQL queries in the following use cases:

- To create reports in Systinet Report Editor. For details, see the *Report Editor Guide*.
- To customize pages of the Systinet user interface. For details, see "UI Customization" in the Administration Guide.
- You can also use DQL in any SQL designer using the DQL JDBC driver. For more details, see "DQL in SQL Designers" on page 68

The following sections contain DQL examples:

- "Primitive Properties" on the next page
- "Complex Properties" on the next page
- "Artifact Inheritance" on page 49
- "Categorization Properties" on page 49

Primitive Properties

Primitive properties are simple properties, such as numbers, characters, and dates, that may occur once or multiple times for an artifact depending on the cardinality as defined in the SDM.

For example, in the SDM Model, each person is represented by a person artifact. The person artifact includes a name property with single cardinality and an email property with multiple cardinality.

The following query returns the name and all emails for each person in the repository.

```
select name, email
  from personArtifact
```

Instances of primitive properties with multiple cardinality are all returned as comma separated values. For example, all the emails for a person return as a concatenated, comma-separated string. If there is no instance of the property for an artifact, a null value is returned.

The following query returns the name, description, and version of all business service artifacts whose version is 2.0.

```
select name, description, version
  from businessServiceArtifact
  where version = '2.0'
```

Note: By default, DQL queries return the latest revisions of artifacts unless you specify revision modifiers. For details, see "Modifiers" on page 53.

Complex Properties

Complex properties are composed of one or more single or multiple-valued sub-properties (for example, address contains sub-properties addressLines in multiple cardinality, country in single cardinality, etc. The sub-property addressLines is also a complex sub-property, containing a value and useType.). It is only possible to query the sub-property components of primitive types. Components of sub-properties are separated by **.** (in MS Access you can use **$** as a separator).

```
select address.addressLines.value, address.country
  from personArtifact
  where address.city = 'Prague'
```

For a full reference of all complex properties in the default SDM, see "SOA Definition Model" in the *Reference Guide*.

Artifact Inheritance

Artifacts in Systinetform a hierarchy defined by the SDM model. Artifacts lower in the hierarchy inherit properties from higher abstract artifact types. `artifactBase` is the root abstract artifact type in the SDM hierarchy. All other artifacts are below it in the hierarchy and inherit its properties. You can query abstract artifacts and return a result set from all the instances of artifact types lower in the hierarchy.

Property groups function in a similar way, querying a property group returns results from all artifact types that inherit properties from the group.

The following query returns results from all implementation artifacts; SOAP Services, XML Services, and Web Applications.

```
select name, serviceName
  from implementationArtifact
```

Notice that in this query, serviceName is a specific property of SOAP Service artifacts. In the result set, name is returned for all implementation artifacts but serviceName is only returned for SOAP service artifacts. For other implementation types, the serviceName is NULL.

Caution: Different artifact types may define the same properties with different cardinalities. In cases where two artifact types define the same property with different cardinality, querying a shared parent abstract artifact for these properties may fail. Examples that fail include **SELECT environment FROM artifactBase** and **SELECT accessPoint FROM artifactBase**.

Categorization Properties

Categorization properties are a special case of complex properties.

Categorization properties have the following sub-properties:

- `val` - machine readable name of the category.
- `name` - human readable name of the category.
- `taxonomyURI` - identifies the taxonomy defining the category set.

  Note: The taxonomyURI is not defined for named category properties.

Systinet uses categorization properties in the following ways:

- **Named category properties** (for example, business service criticality).

  The following query returns the names, descriptions, and versions of all business service artifacts which are categorized using the named criticality categorization property with a high failure impact.

  ```
  select name, description, version
    from businessServiceArtifact
    where criticality.val =
          'uddi:systinet.com:soa:model:taxonomies:impactLevel:high'
  ```

  Note: The taxonomyURI is not defined for named category properties. The name of the category property implies the taxonomy.

- **categoryBag**

  `categoryBag` is a complex property that includes sub-property `categories` which is a categorization property and `categoryGroups`. `categoryGroups` also contains categorization sub-property `categories` and a `taxonomyURI` defining the meaning of the group. HP recommends querying `_category` instead of `categoryBag` to ensure that all categories are queried.

  The following query returns the names, descriptions, and versions of all business service artifacts which are categorized by the Gift certificate category (14111608) of the `uddi:uddi.org:ubr:categorization:unspsc` taxonomy.

  ```
  select name, description, version
    from businessServiceArtifact
    where categoryBag.categories.taxonomyURI =
          'uddi:uddi.org:ubr:categorization:unspsc'
      and categoryBag.categories.val = '14111608'
  ```

- **identifierBag**

  `identifierBag` is a complex property similar to `categoryBag` that includes sub-property `categories`. `identifierBag` does not contain the `categoryGroups` subproperty. HP recommends querying `_category` instead of `identifierBag` to ensure that all categories are queried.

- **_category**

  This generic categorization property holds all categorizations from `categoryBag`, `identifierBag`, and all named categorization properties from the given artifact type.

  The following query returns the names, descriptions, and versions of all business service artifacts which are categorized with a high failure impact.

  ```
  select name, description, version
    from businessServiceArtifact
    where _category.val =
          'uddi:systinet.com:soa:model:taxonomies:impactLevel:high'
  ```

```
   and _category.taxonomyURI =
       'uddi:systinet.com:soa:model:taxonomies:impactLevel'
```

Caution: When you use the generic `_category` property you must specify the taxonomy using the `_category.taxonomyURI` sub-property. When you use a named categorization property the taxonomy is implicitly known and does not need to be specified.

Fixing Multiple Properties

Consider a business service with keywords, 'Finance' and 'Euro'. The intuitive query for finding a 'Euro Finance' service is as follows:

```
select name, description, version
  from businessServiceArtifact b
  where b.keyword.val = 'Finance'
    and b.keyword.val = 'Euro'
```

This query does not work as a single instance of keyword can never be both 'Finance' and 'Euro'

The solution is to fix instances of multiple properties as shown in the following query:

```
select name, description, version
  from businessServiceArtifact b, b.keyword k1, b.keyword k2
  where k1.val = 'Finance'
    and k2.val = 'Euro'
```

Relationships

A relationship is a special kind of complex property pointing to another artifact. Systinet uses relationships to join artifacts.

The following queries are semantically identical and return all business services and the contact details of their provider. These queries do not return business services that do not have providers.

- The following query is an example of an *SQL92-like* join which uses the USING clause.

```
select b.name, b.version, b.keyword.name, p.name as contact, p.email
  from businessServiceArtifact b
  join personArtifact p using provides
```

  The relationship property `provides` leads from person artifacts to business service artifacts is specified after the `using` keyword.

- The following query is an example of an *SQL92-like* join which uses the ON clause.

```
select b.name, b.version, b.keyword.name, p.name as contact, p.email
```

```
  from businessServiceArtifact b
  join personArtifact p on bind(provides)
```

The relationship property `provides` leads from person artifacts to business service artifacts is specified with the `bind` predicate in the `WHERE` clause.

- The following query is an example of an *old-style* join which uses the BIND predicate.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b, personArtifact p
  where bind(p.provides, b)
```

The BIND predicate specifies that the provides relationship of the person artifact points to business service artifacts.

The following query also returns all business services and the contact details of their provider. This query is an example of a LEFT JOIN. The LEFT JOIN extends the previous queries by also returning business services that do not have providers.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b
  left join personArtifact p using provides
```

Each relationship has the following sub-properties which you can query:

- rType - the SDM QNames of the relationship type.
- useType - the values of the useType relationship property
- target - the UUIDs of the artifact the relationship points to (deprecated).

It is possible to specify a particular provider type using useType. The following queries return all business services and their contact details where the provider is an architect.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b, personArtifact p
  where bind (p.provides, b)
    and p.provides.useType = 'architect'
```

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b
  join personArtifact p on bind(p.provides, b)
    and p.provides.useType = 'architect'
```

It is possible to traverse several relationships using several old-style joins or SQL-92-like join clauses in the same query. The following example queries business services in applications, which are also part of a project.

```
select b.name, b.description, a.name as Application, p.name as Project
  from businessServiceArtifact b
  join hpsoaApplicationArtifact a using hpsoaProvidesBusinessService
  join hpsoaProjectArtifact p using contentRelationshipType
```

In cases where artifacts may be joined by multiple properties, you can use a generic _relation property together with the additional rType condition.

```
select A.name as A_name, B.name as B_name
  from hpsoaApplicationArtifact A left join artifactBase B on bind(A._relation)
    and A._relation.rType in (
        '{http://systinet.com/2005/05/soa/model/property}
hpsoaProvidesBusinessService',
        '{http://systinet.com/2005/05/soa/model/property}r_providesBusinessProcess'
      );
```

You can use the target relationship sub-property to bind the source and target of a relationship.

```
select b.name, b.version, p.name as contact, p.email
  from businessServiceArtifact b, personArtifact p
    where p.provides.target = b._uuid
```

Caution: The target property and this style of comparison is deprecated and its use is not recommended. Use the bind predicate instead.

Modifiers

Modifiers define primary sets of objects (artifacts and their revisions) to query. If no modifier is specified, the last revisions of undeleted artifacts for which the user has read access are queried.

The following modifiers are available:

- Revision related modifiers (mutually exclusive):
  - **all_rev** - queries all revisions of artifacts.
  - **last_approved_revision** - queries the last approved revisions of artifacts.
- Security related modifiers (mutually exclusive):
  - **my** - queries artifacts belong to the user.
  - **writable** - queries artifact the user has write permission for.
  - **no_acl** - queries all artifacts regardless of security.
- Other modifiers:
  - **include_deleted** - queries all instances, including deleted artifacts.

You can use multiple comma-separated modifiers.

The following query returns all business services that you own that are marked as deleted.

```
select b.name, b.version, b.keyword.name
```

```
  from businessServiceArtifact b (my, include_deleted)
  where _deleted = '1'
```

Virtual Properties

DQL defines virtual properties, that are not defined by the SDM. Systinet stores or calculates these properties enabling DQL to query meta information about artifacts. These virtual properties provide information about lifecycle, compliance, domains, etc.

The following example returns lifecycle details from the last approved revisions of all business service artifacts, ordered by lifecycle stage.

```
select name, _lastApprovedStage.name Stage, _revision
  from businessServiceArtifact(last_approved_revision)
  order by Stage
```

The following example returns the name and compliance status of last approved revisions of all business services which a compliance status of at least 80%.

```
select b.name, b._complianceStatus
  from businessServiceArtifact b (last_approved_revision)
  where b._complianceStatus >= 80
```

Systinet repository content exists within a domain structure where each artifact exists within only one domain. The default functionality of DQL queries all domains but Systinet provides virtual properties enabling you to query artifacts within a particular domain. The following example returns business service names and the domain details of all business service artifacts that exist within the EMEA domain.

```
select A.name, A._domainId, A._domainName
  from businessServiceArtifact A
  where A._domainId="EMEA"
```

DQL provides the following macros for querying within domain hierarchies:

- **#SUBDOMAINS('*domainId*')**

  Queries the specified domain and all its sub-domains.

- **#SUPERDOMAINS('*domainId*')**

  Queries the specified domain and all its parent domains.

- **#SECURITYCONTEXT('*userId*')**

  Returns current user.

  `userId` is a keyword and is case-insensitive.

- **#SECURITYCONTEXT('*workingDomain*')**

  Returns current domain.

  `workingDomain` is a keyword and is case-insensitive.

The following query returns all business services in the EMEA domain and any of all of its sub-domains.

```
select A.name
  from businessServiceArtifact A
  where A._domainId in #SUBDOMAINS('EMEA')
```

The following query returns artifacts in working domain and belongs to currently logged in user.

```
select :columns
  from artifactBase a
  where a._domainId in #SECURITYCONTEXT(‘workingDomain’)AND

  a._ownerin #SECURITYCONTEXT(‘userId’)
```

The following query returns the name and virtual properties artifactTypeName and owner from the latest revisions of consumer properties (the property group for all consuming artifact types).

```
select name, _artifactTypeName, _owner
  from consumerProperties
```

For details of all virtual properties, see "Properties in DQL" on the next page.

Embedding SQL Queries

DQL works with SDM entities (artifacts and properties) only and cannot directly access database tables. In some cases it is necessary to obtain values from outside the SDM (for example, system configuration). You can use an SQL subquery in a NATIVE clause of a DQL query. By default, DQL expects SQL to return an unnamed single column of values.

The following example returns business services owned by the administrator using the name defined during installation:

```
select name,description, version
  from businessServiceArtifact
  where _owner in (
    native {select svalue from systemConfiguration
              where name='shared.administrator.username'})
```

You can use NATIVE clauses instead of expressions, as a condition in WHERE clauses, as a column in SELECT clauses, and as a artifact reference in FROM clauses. For details, see "DQL Grammar" on page 60 .

If you use a NATIVE clause to formulate part of a FROM clause, you must specify parameters to bind columns defined by SQL to properties used by DQL.

Each parameter consists of the following:

- The property name defines how DQL addresses columns returned from the NATIVE SQL statement.
- The property type which may be returned by the metadata of a column is optional and if not specified is assumed to be a text string.

The parameters are enclosed in brackets in the native clause, delimited by commas, and the type is separated from the name using whitespace.

The following example shows a query with NATIVE SQL in a DQL FROM clause.

```
select B.p_id, B.s_val, A.name, B.state_index
  from (
    native(s_val, s_name, state_index integer, p_name, p_id)
      {select S.val as s_val, S.name as s_name, S.state_index as state_index,
              P.name as p_name, P.id as p_id
         from rylf_state S, rylf_process P
         where S.fk_rylf_process=P.id and P.name='Application Lifecycle'}) B
  left join artifactBase A on A._currentStage.val = B.s_val
  order by B.p_id, B.state_index
```

The NATIVE statement returns the following columns; `s_val`, `s_name`, `p_name`, and `p_id` of type String, and `state_index` of type Integer.

Note: Native clauses can not contain variables (? or :<variable>).

DQL Reference

This section provides a reference to properties and DQL grammar in the following sections:

- "Properties in DQL" below
- "DQL and SQL" on page 60
- "DQL Grammar" on page 60

Properties in DQL

Artifact (property group) properties hold values which may be queried in DQL expressions.

DQL recognises the following properties:

- SDM Properties

  Properties defined in the SDM Model. For details, see "SOA Definition Model" in the *Reference Guide*.

- Virtual, System, and Other Properties

  Properties holding metadata about artifact instances.

Properties may be one of the following:

Property Kind	**Description**
Primitive	Holds string, number, or boolean values. For example, name, description, version. A primitive property is defined in DQL statements by the artifact type name or alias, the property delimiter (. or $), followed by the property name. For example, personArtifact.name. The artifact name or alias is optional (with the delimiter) when the property is specific to a single artifact type in the query.
Complex	Hold complex structures such as address. Only primitive sub-properties of complex properties may be queried. Properties and sub-properties are separated by **.** or **$**. For example, `personArtifact.address.city`.
Categorization	Hold categorization data and are handled in a similar way to complex properties. Categories consist of `name`, `val`, and `taxonomyURI` components. For example, `businessServiceArtifact.criticality.name`.
Relationships	Properties that specify a directional relationship to other artifacts.

All values that you can query are of a particular data type. The following table describes these data types, and gives examples of how to use them in a query.

Data Type	**Description**	**Example**
Number	Numeric values	_revision = 1
String	Text values	_revisionCreator = 'admin'
Date Time	Date and Time (ms since 00:00 1/1/1970)	_revisionTimestamp > 1274447040124 /* revisions made since 15:04 21/5/2010 */
Boolean	True or False flags	_deleted = '1'

Properties may have the following cardinalities:

Property Cardinality	**Description**
Single	Only one instance of the property exists for an artifact and it may be optional or required.

Property Cardinality	Description
Multiple	The property is a list of values and so occurs multiple times for an artifact. In WHERE clauses, using multiple properties returns particular artifacts if any instance of the multiple property matches the condition. In SELECT clauses, using multiple properties returns all instances of the multiple property as a concatenated, comma-separated string.

DQL uses the following system properties:

System Property	Description
_artifactTypeName	The human readable name of the artifact type (the SDM label of the artifact). HP recommends using _sdmName in conditions, and _ artifactTypeName in SELECT clauses.
_category	All categorizations for an artifact with `name`, `val`, and `taxonomyURI` components.
_deleted	The deletion marker flag (boolean).
_id	The database ID of an artifact instance (number, deprecated - use _uuid).
_longDescription	Systinet supports a long description including HTML tags up to 25000 characters by default. HP recommends using the `description` property in DQL queries instead as queries using _longDescription may affect performance and the HTML tags may corrupt report outputs. `description` contains only the first 1024 text characters of `_longDescription` (may vary according to your database type). **Note:** The `platform.repository.max.description.length` property determines the maximum length of _longDescription. You can modify this property in `SYSTINET_ HOME/conf/setup/configuration-propeties.xml`.
_owner	The user, group, or role designated as the artifact owner.
_ownerName	The human readable name of the user (taken from the use profile), group, or role artifact.
_path	Legacy REST path of the artifact (string, deprecated - use _uuid).
_relation	A generic virtual property that may be used to specify all outgoing relationships.
_revision	The revision number of an artifact instance.
_revisionCreator	The user who created the revision of the artifact.
_revisionTimestamp	The date and time the revision was created.

System Property	Description
_sdmName	The local name of the artifact type. HP recommends using _sdmName in conditions, and _artifactTypeName in SELECT clauses.
_uuid	The unique artifact identifier.

DQL uses the following virtual properties:

Property Class	Property	Description
UI Property	_isFavorite	Marked by the user as favorite flag (boolean).
	_rating	The average rating of the artifact (double).
Security Property	_shared	Indicates that the artifact is shared and visible to users in the Sharing Principal role (boolean). For more details, see "How to Share Artifacts" in the *User Guide*.
	_writable	User write permission flag (boolean). **Note:** Using this property may have a performance impact. If possible, use the **writable** modifier instead. For details, see "Modifiers" on page 53.
Contract Property	_enabledConsumer	The artifact is a valid consumer artifact type.
	_enabledProvider	The artifact is a valid provider artifact type. The artifact must also be marked as 'Ready for Consumption'.
Domain Property	_domainId	Value of the `domainId` property defined for the domain artifact that the artifact belongs to.
	_domainName	The readable name of the domain.
Lifecycle Property	_currentStage	Current working stage of an artifact.
	_governanceProcess	process applicable to the artifact.
	_isApproved	Lifecycle approval flag (boolean).
	_lastApprovedRevision	Revision number of the last approved revision (number).
	_lastApprovedStage	The name of the last approved stage.
	_ lastApprovalTimestamp	Timestamp for the last approval (number, ms since 00:00 1/1/1970).
	_lifecycleStatus	The status of the current lifecycle stage.
Policy Manager Property	_complianceStatus	Compliance status as a percentage (number).

DQL and SQL

DQL supports most features of SQL with the following exceptions:

- SELECT * is not supported.
- RIGHT and FULL OUTER JOIN are not supported.
- It is not possible to use properties with multiple cardinality in GROUP BY, HAVING, or ORDER BY clauses.

DQL Grammar

A DQL query consists of the following elements with their grammar explained in the following sections:

- "Select" on the next page
- "FROM Clause" on the next page
- "Conditions" on page 62
- "Expressions" on page 64
- "Lexical Rules" on page 65

Typographical Conventions

Convention	Example	Description
KEYWORDS	**SELECT**	A reserved word in DQL (case-insensitive).
parsing rules	*expr*	Name of a parsing rule. A parsing defines a fragment of DQL which consists of keywords, lexical rules, and other parsing rules.
LEXICAL RULES	*ID*	Name of a lexical rule. A lexical rule defines a fragment of DQL which consists of letters, numbers, or special characters.
[ ]	[**AS**]	Optional content.
[...]	[, *select_item*, ...]	Iterations of optional content.
|	**ASC** | **DESC**	Alternatives.
{ }	{ + | - }	Group of alternatives.
..	0..9	A range of allowable characters.

Select

```
select :
  subquery [ ORDER BY                                    order_by_item [, order_by_
item ...]]

subquery :
  subquery [ set_operatorsubquery ...]
  | (subquery)
  | native_sql
  | subquery_base

subquery_base :
  SELECT [ DISTINCT ] select_item [, select_item ...]
  FROM                                  from_clause_list
  [ WHERE                                  condition ]
  [ GROUP BY                                  expression_list
    [ HAVING                                  condition ]
  ]

select_item :
  expr [ [ AS ] alias ]

alias :
  ID | QUOTED_ID

order_by_item :
  expr [ ASC | DESC ]

set_operator :
  UNION ALL | UNION | INTERSECT | EXCEPT

native_sql :
  NATIVE [ (column_name [ column_type ] [ , ... ] ) ]
  { sql_select }
```

Explanation:

- The { } around the sql_select are required and sql_select is an SQL query.
- The column_name and column_type specify parameters to pass from the SQL query to the DQL query.

FROM Clause

```
from_clause_list :
  { artifact_ref | subquery_ref | fixed_property | native_sql }
  [ from_clause_item ... ]

from_clause_item :
  , { artifact_ref | subquery_ref | fixed_property | native_sql }
  | [ LEFT [ OUTER ] ] JOIN
    { artifact_ref | subquery_ref } join_condition

artifact_ref :
  artifact_name [ alias ] [ (artifact_modifiers) ]

subquery_ref :
  (subquery)alias

fixed_property :
  property_refalias

artifact_modifiers :
  ID [ ,ID ... ]

artifact_name :
  ID

join_condition :
  | USINGproperty_ref
```

Conditions

```
condition :
  condition_and [ OR                                  condition_and ... ]

condition_and :
  simple_condition [ AND                                  simple_condition ... ]

simple_condition :
  (condition)
  | NOT                                   simple_condition
  | exists_condition
  | like_condition
  | null_condition
  | in_condition
  | simple_comparison_condition
  | native_sql
  | bind

simple_comparison_condition :
  exprcomparison_opexpr

comparison_op :
 = | <> | < | > | <= | >=

like_condition :
  expr [ NOT ] LIKE                                  like_expression [ ESCAPE
                             STRING ]

like_expression :
  STRING
  | variable_ref

null_condition :
  expr                                    IS [ NOT ] NULL

in_condition :
  expr [ NOT ] IN( { subquery | expression_list } )
  | macro

exists_condition :
  EXISTS(subquery)

bind :
  BIND(property_ref [ , alias ] )

macro :
  macro_name [ (expression_list) ]

macro_name :
  #ID
```

Explanation:

- Conditions can be evaluated to true, false, or N/A. *condition* consists of one or more *condition_and* that are connected by the **OR** logical operator.
- *condition_and* consists of one or more *simple_condition* connected by the **AND**
- *simple_condition* is one of following:
  - *condition* in parentheses.
  - Negation of *simple_condition*.
  - *exists_condition*
  - *like_condition*
  - *null_condition*
  - *in_condition*

- *simple_comparison_condition*
- *native_sql*

- *simple_comparison_condition* is a comparison of two expressions using one of the comparison operators: =, <>, <, >, <=, >=
- *like_condition* compares an expression with a pattern. Patterns can contain wildcards:
  - **_** means any character (including numbers and special characters).
  - **%** means zero or more characters (including numbers and special characters).
  - **ESCAPE** *STRING* is used to prefix _ and % in patterns that should represent those characters and not the wildcard.
- *alias* references the target artifact.

Expressions

```
expr :
  term [ { + | - | CONCAT } term ... ]

term :
  factor [ { * | / } factor ... ]

factor :
  (select)
  | (expr)
  | { + | - } expr
  | case_expression
  | NUMBER
  | STRING
  | NULL
  | function_call
  | variable_ref
  | property_ref
  | native_sql

case_expression :
  CASE                                      case_item [ case_item ... ]
    [ ELSE                                      expr ]
  END

case_item :
  WHEN                                      condition
 THEN                                      expr

function_call :
  ID( [ DISTINCT ] { [ * ] | [ expression_list ] } )

property_ref :
  { ID | QUOTED_ID } [ { . | $ } { ID | QUOTED_ID } ... ]

expression_list :
  expr [ ,expr ... ]

variable_ref :
  ? | :ID
```

Explanation:

- Variables are of two kinds:
  - Positional variables - ? in DQL.
  - Named variables - :<name_of_variable>
- When variables are used in DQL, each variable must have a value bound to the variable.

Lexical Rules

```
CONCAT :
  ||

STRING :
  [ N | n ] ' text '

NUMBER :
  [ [ INT ] . ] INT

INT :
  DIGIT [ DIGIT ... ]

DIGIT :
  0..9

ID :
  CHAR [ { CHAR | DIGIT } ... ]

CHAR :
  a..z | A..Z | _
```

Explanation:

- *ID* is sequence of characters, numbers and underscores beginning with a character or underscore.

- *QUOTED_ID* is text in quotes.
- *CONCAT* means a concatenation of strings - syntax **||**

DQL with Third-Party Products

DQL is provided by a JDBC driver which you can use with common SQL designers supporting third-party JDBC drivers (or ODBC with an ODBC-JDBC bridge).

The following sections describe the driver and its use with 3rd party products:

- "DQL JDBC Driver" below
- "DQL in SQL Designers" on page 68
- "DQL in MS Access" on page 68

DQL JDBC Driver

The DQL JDBC driver translates DQL queries into SQL queries and executes them using the underlying JDBC driver for the used database. The translation is provided by a remote invocation of Systinet.

All the required JAR files for the DQL driver are available in `SYSTINET_HOME/client/lib/jdbc`:

- `pl-dql-jdbc.jar`
- `hessian-`*`version`*`.jar`
- Database driver JAR files are copied here during installation (for example, `ojdbc6.jar`).

The following table describes the driver configuration required to use the driver with 3rd party products.

DQL JDBC Driver Configuration

Property	Description
Connection String	jdbc:systinet:http(s)://<username>@<host:port>/<context>((|<option>=<value>)+)? • `<username>` is the Systinet username who executes the DQL query using Systinet permissions security. • <host:port> are the connection details of your Systinet installation (for example, localhost:8080 for HTTP or secure:8443 for HTTPS. • <context> is the application server context, the default is systinet. where the options are: 1. schema : (optional) is the schema of the user who owns HP Systinet database tables. 2. model : (optional) is the list of allowed models (public,sys, or “public,sys”) 3. dqlwrapper : (optional) a comma separated character sequences that specify start and end token that wraps a DQL statement inside an SQL statement. This construct allows to execute embed a DQL command into an SQL command 4. driverclass : (optional) JDBC driver class, obtained from Systinet server by default 5. dbtype : (optional) database type (mssql, db2 or oracle), obtained from Systinet server by default 6. dburl : (optional) JDBC connection URL that used to communicate directly with the Systinet’s database, obtained from Systinet server by default For example, jdbc:systinet:http://admin@demoserver.acme.com:8080/soa|schema=SOA320
DB Credentials	The database username and credentials used for direct access to the Systinet database. In most cases it is the user who owns all tables for Systinet - called the *power user*. In case of "Manual Database Arrangement" with a power user and a common user (who has only read/write access to tables, but can not create other tables), use the common user account. In case the common user is still too powerful to be shared, the DB administrator can create another - "read-only user" with read-only access to Systinet tables. Note that the read-only user must also have created synonyms/aliases for Systinet tables to pretend that Systinet tables are in the schema of the read-only user. For more details, see "Database Installation Types" in the *Installation and Deployment Guide*.
DQL JDBC Classname	com.hp.systinet.dql.jdbc.DqlDriver

Note: The DQL JDBC driver must be able to connect to the database from the client. Use the full hostname for your database used during installation or setup. In the event of connection problems, verify the firewall settings between the local server and the database server.

Sample:

When using JDBC connection string:
`jdbc:systinet:http://admin@systinet:8080/soa|dqlWrapper=DQL{,}DQL|dbUrl=jdbc:oracle:thin:@systinet:1521/XE`

Virtual DB Connection is obtained where:

- Systinet gets connected at URL http://systinet:8080/soa.
- DQL runs as "**admin**"
- When there is a sequence DQL{….}DQL detected. Then its inside contents will be translated from DQL to SQL and rest (outside of the sequence) will be handled as plain SQL.
- Systinet will overwrite database connection URL to `jdbc:oracle:thin:@systinet:1521/XE`

Sample:

Select embedding DQL: `select count(*) from (DQL{select a._uuid, a.name from hpsoaApplicationArtifact a}DQL)`

DQL in SQL Designers

SQL Designer software can use the DQL driver if the designer is JDBC-aware.

To configure a JDBC-aware SQL Designer:

1. Add the DQL JDBC JAR files to the classpath.
2. Create a JDBC connection using the properties described in "DQL JDBC Driver Configuration" on page 66.

After you establish the DQL JDBC connection, the following functionality should be available in your SQL Designer:

- Schema introspection, browsing the list of artifact types and property groups as tables, and their properties as columns.
- DQL query execution.

DQL in MS Access

MS Access 2007 can execute DQL queries using an ODBC-JDBC bridge. Before using MS Access, you must configure the ODBC datasource in Windows.

To configure an ODBC-JDBC bridge:

1. Download and install an ODBC-JDBC bridge. For example, *Easysoft ODBC-JDBC Gateway*.

2. Configuration typically consists of:

   - JDBC driver configuration using the properties described in "DQL JDBC Driver Configuration" on page 66.

   - Bridge configuration. For details, see the documentation for the bridge software.

DQL syntax varies from the examples given in "Introduction to DQL" on page 47 in the following cases:

- Complex properties must use $ notation and be enclosed by [ ].

  ```
  personArtifact.[address$addressLines$value], personArtifact.[address$country]
  ```

- To use modifiers such as (include_deleted) use the Pass-Through option in MS Access.

- Left Joins do not work. Use plain joins instead.

- For fixed properties, use the Pass-Through option in MS Access.

- For timestamps, use the Pass-Through option in MS Access.

- Native queries do not work in MS Access.

- For property aliases, do not use quoted aliases.

Chapter 7: WebDAV Compliant Publishing

Systinet uses a WebDAV compliant workspace to store data content uploaded to the repository using the publishing functionality described in "How to Publish Content" in the *User Guide*.

Systinet supports WebDAV Level 1 (no locking). For details, see http://www.ietf.org/rfc/rfc4918.txt.

Caution: WebDAV functionality is unavailable for Systinet integrated with Siteminder because Siteminder does not support the WebDAV protocol.

The WebDAV protocol enables document access in a file-system manner. You can access, create, modify, and delete documents using a WebDAV compliant client.

The publishing location is available at the following URL which varies depending on the authentication and transport security you use:

- Authenticated (username/password required)

  `http://`*`SERVER:PORT`*`/systinet/platform/restSecure/location`

  `https://`*`SERVER:SSLPORT`*`/systinet/platform/restSecure/location`

- Anonymous (username/password not required)

  `http://`*`SERVER:PORT`*`/systinet/platform/rest/location`

  `https://`*`SERVER:SSLPORT`*`/systinet/platform/rest/location`

Tip: In Linux clients you may need to use `webdav` or `davs` as the protocol instead of `http(s)`.

HP recommends using the authenticated URL. Systinet permissions apply to operations performed in the publishing location using WebDAV.

You can use the URL in your WebDAV client, for example, in any of the following ways:

- As a publishing location in your IDE.

  For example, Eclipse or Visual Studio with appropriate WebDAV plugins, specifically, Plugin for Eclipse and Plugin for Visual Studio.

- As a mapped web folder in Windows.

  Note: Windows requires the KB907306 patch for the correct client functionality:

  http://www.microsoft.com/downloads/details.aspx?FamilyId=17C36612-632E-4C04-9382-987622ED1D64&displaylang=en

HP recommends deploying Systinet using standard HTTP/HTTPS ports (80/443) to ensure the correct client functionality.

In Windows Vista, a file from the publishing workspace opened in MS Office applications may appear as read-only. In this case, make a local copy and resubmit it to the server after you make your changes.

- Using a 3rd party file manager program with the appropriate plugin. For example, Total Commander with the plugin available at http://ghisler.fileburst.com/fsplugins/webdav.zip.

  Note: `Use multi-step upload method` must be disabled in Total Commander or any file is published as a documentation artifact. Restart Total Commander after changing any plugin settings.

Consult your WebDAV client documentation for details of their WebDAV functionality.

WebDAV access enables you to work with documents published to the repository using the publishing location like a file system (depending on the client). Systinet handles create and update operations using its publishing functionality, so relationships between documents are established and maintained with respect to the document content (for example, when a WSDL references an XSD, Systinet publishes the XSD and a relationship between them is established). These details are available in the Systinet UI in the document artifact details.

WebDAV publishing is an alternative to UI-based publishing. Unlike the configuration of UI publishing (for example, what artifacts to create), WebDAV publishing can only be configured globally using the configuration described in "Configuration Management" in the *Administration Guide*.

The most common WebDAV client operations are:

- Retrieving the content of published documents.

  For example, import a WSDL to your IDE client for service implementation development.

- Publishing new documents.

  For example, publish a WSDL to the repository from your IDE client. Systinet uses its publishing feature to create the document and associated artifacts. Relationships are automatically maintained.

- Republishing documents.

  For example, importing a WSDL to your IDE client, modifying it, and then republishing. Systinet uses its publishing functionality to update the document and maintain associated artifacts and relationships.

- Deleting documents.

  For example, using your IDE client to delete an obsolete WSDL. Systinet uses Delete instead of Purge enabling retrieval of the document if required.

- Changing document locations.

  WebDAV clients can use the MOVE operation to change the server location for an artifact in the repository. Systinet maintains metadata and history. This functionality enables remote management of the publishing location.

- Creating, renaming, and deleting directories.

  The publishing location is effectively a file system, enabling you to organize your documents in the publishing location using your WebDAV client.

- Copying documents or whole directories.

  Create duplicates of publishing folders or documents in the publishing location.

Chapter 8: Technical Security

This chapter provides a technical description of Systinet security.

Security is described in the following sections:

Systinet Overview

Systinet consists of the following components:

- **Web UI**

  Exposes the WEB service providing the Systinet UI.

- **Platform**

  Provides a repository (data store) for artifacts.

  Exposes WEB and REST services to manage artifacts.

- **Policy Manager**

  Engine for policy validation.

  Exposes REST services to policy management and validation.

- **Reporting**

Store for report definitions and data.

Engine for report generation.

Exposes REST service for report management.

These components are deployed as a single EAR file which is generated by the installation.

Users and Groups

Systinet delegates authentication to the J2EE container. The userstore is not managed by Systinet, but by the application server or LDAP/AD tools.

Systinet uses the following definitions:

- **User**

  A user represents the identity accessing Systinet.

  Use your application or LDAP/AD tools to manage users.

- **User Profile**

  Profiles provide additional information for Systinet. For example, a contact email used for mail notifications and a primary group used for collective ownership.

- **Role**

  Roles are defined by functional security. They define the actions permitted to a user. Currently, only the *administrator* role is defined.

- **Group**

  Groups are defined by organizational security following the company structure.

  Systinet uses the following types of groups:

  - **external**

    Groups defined by LDAP. These must be managed within LDAP.

  - **internal (local)**

    Groups managed within Systinet by the administrator.

Systinet uses the following user types for processing:

- **authenticated**

  A user authenticated by J2EE. For example, a user/password for HTTP.

  See the "Authentication" on the next page for authentication mechanisms.

- **anonymous**

  A user who does not pass any credentials and accesses service on access points with an anonymous authentication mechanism.

  The name used in ACL is `systinet#anonymous`.

- **resource owner**

  A user who owns the accessed resource. Used in ACL evaluation.

- **administrator**

  A user with the administrator role. The administrator has the rights to perform all actions (no ACLs are applied on resources, management tasks, and so on).

  During installation, you must define an administrator.

  Systinet user and group management enables you to assign the administrator role to users or entire user groups.

- **system administrator**

  An internal identity used for the execution of internal tasks. It is not possible to authenticate (log in) with this identity. This user has the same capabilities as an administrator.

  The name used in ACL is `systinet:admin`.

Systinet uses the following built-in groups:

- `system#registered`

  All users who exist in the userstore. In other words, users who are authenticated.

- `system#everyone`

  Both authenticated users (group `system#registered`) and anonymous users (`systinet#anonymous`).

Transport Security

Systinet provides several REST and WEB services. They are exposed at access points mapped on the HTTP and HTTPS transports provided by the hosting application server. It also provides installation scenarios where you can enable or disable HTTP or HTTPS.

Systinet does not provide SSL management (certificates) because HTTPS transport is provided by the application server.

For simple JBoss configuration, Systinet provides automatic SSL enablement (certificate generation and SSL configuration) during installation.

On the client side (for example, Systinet accesses HTTPS URLs to upload WSDLs), the handling of SSL certificates is configurable (for example, the selection of truststores, enable/disable hostname verification).

Authentication

Authentication is provided by the J2EE application server. The application server capability determines which method is used (for example, HTTP Basic, SiteMinder). For backward compatibility, it is possible to configure Systinet authentication (SiteMinder and client SSL certificates) but the preferred authentication is via J2EE application servers.

J2EE session management is used for both WEB and REST services.

Resource ACL

Systinet does not use J2EE authorization to access service resources (for example, REST resources are artifacts and collection or WEB resources are tasks).

Platform, Policy Manager, and Reporting Service components provide hierarchical resource models accessible by REST. In these models there are collections and resources, where a collection can contain both individual resources and other collections.

Platform and Reporting Service use the same ACL model.

When access to a resource is requested, ACL is used to authorize access for a user using the following model:

- An ACL is a list of ACEs, where an ACE is composed of the following model:
  - **resource owner**

    Can be either a user or a group.

    resource owner and **administrator** always have read and write permission granted so ACLs are not evaluated in these cases.
  - ACL is a list of ACEs, where an ACE is composed of:

- user or group identification
- granted permission:
  - **read**:
    - `artifact/resource` — permission to read any data and metadata of the artifact.
    - `collection` — permission to read the content and metadata of the collection.
  - **write**:
    - artifact/resource — permission to update any data and metadata of the artifact.
    - collection — permission to create new artifacts, resources, and sub-collections, and to update the metadata of the collection.

- No negative ACE.

  It is not possible to deny permission to a user or group.

- No inheritance or propagation of ACL.

  Only the ACL of the accessed artifact is used for authorization.

  A change to a collection ACL does not change any ACLs of collection members.

  To read or update an artifact, it is sufficient to have read or write permission on the resource.

- When a resource is created, its default ACL is set by artifact. It is possible to configure default ACLs per collection (for example, artifact type).

For details about changing the default ACL configuration, see "How to Manage Default Access Rights" in the *Administration Guide*.

WEB Security

The UI is composed of *tasks* mapped on URLs. All UI tasks require an authenticated user who must sign in to Systinet.

The UI is composed of static tasks, so this setup is part of the WEB configuration.

WEB uses J2EE session management, provided by the application server.

Platform Services

Platform provides a REST service, exposed at the following access points, mapped on HTTP and HTTPS transports provided by the hosting application server:

- **Proprietary REST**

  `http://host:port/context/systinet/platform/rest/` and `https://host:port/context/systinet/platform/rest/` operate with the *anonymous* authentication mechanism.

  `http://host:port/context/systinet/platform/restBasic/` and `https://host:port/context/systinet/platform/restBasic/` operate with the default HTTP Basic authentication mechanism, specified by the application server.

- **Atom-Based REST**

  `http://host:port/context/platform/rest/` and `https://host:port/context/platform/rest/` operate with the *anonymous* authentication mechanism.

  `http://host:port/context/platform/restSecure/` and `https://host:port/context/platform/restSecure/` operate with the default HTTP Basic authentication mechanism, specified by the application server.

The REST service uses J2EE session management, provided by the application server.

Reporting Services

Reporting provides a REST service. It is exposed on the following access points, mapped on HTTP and HTTPS transports provided by the hosting application server:

- **Atom-Based REST**

  `http://host:port/context/reporting/rest/` and `https://host:port/context/reporting/rest/` operate with the *anonymous* authentication mechanism.

  `http://host:port/context/reporting/restSecure/` and `https://host:port/context/reporting/restSecure/` operate with the default HTTP Basic authentication mechanism, specified by the application server.

The REST service uses J2EE session management, provided by the application server.

Policy Manager Services

Policy Manager provides a REST service. It is exposed on the following access points, mapped on HTTP and HTTPS transports provided by the hosting application server:

- **Atom-Based REST**

  `http://host:port/context/policymgr/rest/` and `https://host:port/context/policymgr/rest/` operate with the *anonymous* authentication mechanism.

  `http://host:port/context/policymgr/restSecure/` and `https://host:port/context/policymgr/restSecure/` operate with the default HTTP Basic authentication mechanism, specified by the application server.

The REST service uses J2EE session management, provided by the application server.

Default Endpoint Authentication

By default, Systinet performs the following authentication on Systinetendpoints:

- **FORM authentication**:
  - /web/service/catalog/*
  - /web/policy-manager/*
  - /web/shared/*
  - /web/artifactIconList.htm
- **HTTP basic authentication**:
  - /systinet/platform/restBasic/*
  - /platform/restSecure/*
  - /policymgr/restSecure/*
  - /reporting/restSecure/*
  - /remote/navigator/*
  - /remote/upload/*
- **Unauthenticated URL patterns:**
  - /systinet/platform/rest/*
  - /platform/rest/*
  - /policymgr/rest/*
  - /reporting/rest/*

- /web/design/*
- /remote/dql/*

Note: All endpoints are preceded by `http(s):/host:port/context` as set during installation.

Chapter 9: Custom Source Parsers

The source parser you write creates an object representation of a log of messages. When your input source is only a single message, it creates a log of one message.

The following list specifies a mapping between concepts and classes in Systinet Policy Manager API:

- A log of messages corresponds to an instance of `org.systinet.policy.validation.ValidationSourceCollection`. It can contain both inline request/response messages and references to external messages. As credentials are passed along, the external messages can be secured with HTTP basic authentication.

- A request/response conversation (or a single message, if it is one-way) corresponds to an instance of `org.systinet.policy.validation.ValidationSource`. When creating an instance of this class, make sure you set up:

  - SourceType – this should be set to `org.systinet.policy.validation.ValidationConstants#Elements.SOURCE_CONVERSATION`, in case of request/response conversation, or `soap:Envelope` for single-message validation.

  - One (for one-way) or two (for request-response conversation) messages.

- A message corresponds to an instance of `org.systinet.policy.validation.ValidationSourceDocument`.

  You should set up:

  - `content`

    The SOAP payload of the message.

  - `contentURL`

    The url of the SOAP payload. If the SOAP message is inline in the parsed source, you can use `org.systinet.xml.XPointerHelper.appendToURL (java.lang.String,java.lang.String)`, together with `org.systinet.xml.DOMHelper.getXPointer(org.w3c.dom.Element)` to create a URL pointing directly to the payload.

  - `contentBOM` (optional)

    The BOM signature of the content.

  - `description` (optional)

    The WSDL description of the message.

- `descriptionURL` (optional)

  URL of the WSDL description of the message.

- `metadata` (optional)

  Metadata associated with the message. Anything which is `java.io.Serializable` can be added to the metadata. The built-in handlers understand only `org.systinet.policy.validation.SOAPMetadataConstants.METADATA_MESSAGE_HEADERS`, which is used as a key to access transport headers.

- `sourceType`

  This field should be either `soap:Envelope` to indicate that only a SOAP content is available, or `org.systinet.policy.validation.ValidationConstants#Elements.SOURCE_MESSAGE`, to indicate that additional metadata is available.

- `sourceDocumentURL`

  This field should be set to the URL of the whole message; that is, the container for the SOAP payload and metadata. If this container is inlined in a bigger structure, you may use the XPointerHelper class mentioned above to get a more detailed URL. If there is no URL, rather than leaving this field empty, use the URL of the SOAP payload or of the whole request/response conversation.

The parser's main method is `public ValidationSourceCollection parse(String uri, String rootElementNamespaceURI, String rootElementLocalName, SourceResolver resolver, CredentialsList credentials) throws SourceParseException, CredentialsException`. Usually, the parser follows these steps:

1. The parser inspects the `rootElementNamespaceURI` and `rootElementLocalName` to determine if the document should be handled by this parser. If not, it returns immediately with `null` and the parsing framework continues with the next parser.

2. The parser retrieves the parsed document from the source resolver: `Source source = resolver.getSource(uri, credentials)`. This call fetches the document if this is the first time the document was accessed (this is why credentials must be passed) or uses a cached version if the document has been fetched already. The cache expires when the validation of this source ends.

3. The source parser should either create an instance of `ValidationSourceDocument`, pass a reference to another document, or do both. For example, a WSDL source parser creates an instance of `ValidationSource`, adds the parsed WSDL as a new `ValidationSourceDocument`, and then includes each contained/referenced xml schema via `ValidationSource.addReferencedDocument`. All the referenced documents are parsed before the validation starts.

4. If the resource being parsed is a collection, the parser should create a `ValidationSourceCollection` and add the references via `addReferencedSource`.

The URL which goes to the `addReferencedXXX` methods might point inside the parsed resource if XPointer is used. You can use `DOMHelper.getXPointer()` and `XPointerHelper.appendToURL()` to create such a URL.

To be recognized by the source parsing framework, the parser must be bound to the `/systinet/policy/validation/sources/` JNDI context.

Chapter 10: Custom Validation Handlers

In addition to the built-in handlers described in the "Assertion Schema" section in the *Reference Guide*, you can write and deploy your own validation handlers without further changes to the Systinet Policy Manager.

The following points should be kept in mind:

- **Home and remote interfaces**

  The handler must have `org.systinet.policy.validation.handlers.DialectValidator` as its remote interface and `org.systinet.policy.validation.handlers.DialectValidatorHome` as its remote home interface.

- **Classloaders**

  The handler should be deployed within the same classloader. For further details, see `jboss-app.xml`.

- **Deployment path**

  The handler must be deployed to the `systinet/policy/validation/handlers/` JNDI context.

- **Exceptions**

  The handler should never throw an exception, apart from `org.systinet.http.CredentialsException`. If an error occurs, the handler should always create a report saying that there has been an error.

- **Incoming assertions**

  The incoming list of assertions contains instances of `org.systinet.policy.validation.handlers.DialectValidator#AssertionRecord`.

- **Return value**

  The return value must be a list of `org.systinet.policy.model.report.Result`. In this list, there is one result for each of the assertions in the incoming list, placed in the same order.

- `getDialect()`

  This method returns the URI of the dialect this handler accepts. It must be the same as the namespace URI of the first element in the `pe:Enforcement` section of the assertion definition. It is used to filter the input list of assertions. Only the assertions with this namespace are passed into this handler.

Chapter 11: Validation Client

Policy Manager includes a command-line validation client that you can copy to another computer on the network. The validation client is designed for the following uses:

- Validating local and/or remote documents against local policies. These validations run on the client.
- Validating remote documents against policies located on a server. These validations run on the server.

The validation client is located at `SYSTINET_HOME/client`. To install the client, copy this folder to the location of your choice.

The validation client command-line tools are located in `SYSTINET_HOME/client/bin`. The tools and their functions are described in the following sections:

Downloading Policies and Assertions (sync)

To perform validations locally, you need local copies of the policies and assertions in the Systinet repository. To download these policies and assertions, run the `sync` tool. Your computer must be connected to the Systinetserver/cluster when you run `sync`.

To run `sync`, simply enter **sync -u** ***username*** **-p** ***password***. If Systinet does not require any credentials, enter **sync -noauth**. The `sync` tool gets the hostname and port of the Systinet host from the `SYSTINET_HOME/client/conf/policy-manager.properties` file, created automatically when Systinet is installed.

The property used is determined by the `shared.https.use` property and is either:

- shared.http.urlbase=http\://host\:port/context
- shared.https.urlbase=https\://host\:8443/context

Local Validations (validate)

Validate documents against local copies of technical policies by running the `validate` tool. The syntax is:

`validate` [OPTIONS] {--policy local_technical_policy_name,_file_or_uri...} {--source source_file_or_uri...}

For a full list of options and examples of commands, enter **validate --help**.

Caution: Before you can validate a set of documents, download policies and assertions from the server to your local directory using the sync tool.

Policy Formats

You can specify technical policies in the following ways:

- As the plain text name of the policy, in quotation marks. For example, `"SOA Systinet Best Practices"`.
- As the file name (full or relative) of the policy file. For example, `C:/opt/systinet/policymgr/client/data/policies/systinet-best-practices.xml`.
- As the full URI of the policy. For example, `file:///opt/systinet/policymgr/client/data/policies/systinet-best-practices.xml`.

Source Formats

You can write source document locations in the following formats:

- As the file name (full or relative) of the document. For example, `C:/tmp/services/service1.wsdl`.
- As the full URI of the document. For example, `http://host:port/services/service1.wsdl`.

To validate one source against one policy it is not necessary to include any options in the command line. For example, to validate a local copy of `service1.wsdl` against a local copy of the `HP SystinetBest Practices` technical policy, you can run **validate "HP SystinetBest Practices" C:/tmp/services/service1.wsdl**.

Validating Multiple Sources With Multiple Policies

You can validate multiple source documents and/or use multiple technical policies using the `-p` or `--policy` and `-d` or `--source` options. For example, **validate -p "HP SystinetBest Practices" -p file:///opt/systinet/policymgr/client/data/policies/wsdl-validity.xml -d C:/tmp/services/service1.wsdl -d C:/tmp/services/service2.wsdl** validates `service1.wsdl` and `service2.wsdl` against the HP Systinett Best Practices and WSDL Validitytechnical policies.

You can make the validation stop the first time a policy is violated. Use the `-c` or `--stop` option. For example, the validation launched by **validate --stop -p "WSDL Validity" -p "HP SystinetBest Practices" -d C:/tmp/services/service1.wsdl -d C:/tmp/services/service2.wsdl** would stop when either `service1.wsdl` or `service2.wsdl` violated either Systinet Best Practices or WSDL Validity.

Selecting Sources By Wildcard

Instead of specifying every source document to be validated, you can specify a directory of documents and pass a wildcard so all matching documents in that directory will be validated. Specify the directory with the `-d` or `--source` option and use the `-e` or `--pattern` to pass the wildcard. For example, **validate -p "HP SystinetBest Practices" -d C:/tmp/services -e service*.wsdl** would validate `service1.wsdl`, `service2.wsdl`, etc, against the HP SystinetBest Practices technical policy.

Setting Up Output

By default, validation reports are created in text format and printed in the console window. You can save the report as a file by using the `-o` or `-outputDir`option and the file location. For example, **validate -o C:/tmp/reports "HP Systinet Best Practices" C:/tmp/services/service1.wsdl** would create the file `C:/tmp/reports/service1.txt`.

Report names are based on source names by default. To give a report a different name, use the `-n` or `--name` option.

You can produce output in HTML or XML format instead of text. Use the `--format html` or `--format xml` option, respectively. When producing HTML or XML output, specify an output location with the `-o` or `-outputDir`option. Otherwise the raw HTML or XML is only printed out to the console.

When the `validate` tool produces HTML output, it uses a template combining XSL and graphics. The validation client comes with a default template that reproduces the Policy Manager report style. You can add additional templates by saving them in the `../client/templates` folder. Specify the template to be used by using the `-m` or `--template` option. For example, if you saved a custom template in `.../client/templates/MyCustomTemplate`, use it to produce HTML output by running **validate.sh --format html --template MyCustomTemplate [-p policy] [-d source]**. If you do not specify a template, the default template is used.

ANT Task Automation of `validate`

You can automate the execution of the `validate` tool as an ANT task. Write an ANT script to launch `validate` and save the script in `.../client/bin`. Launch it with the **ant** command. For example, if you create an ANT script called `/client/bin/validatetask.xml`, launch it with **ant -f validatetask.xml**.

The elements of the ANT task are given in Table, "`validate` ANT Task Elements". Example, "`validate` ANT Task" is an example of an ANT task script for launching `validate`.

`validate` ANT Task Elements

Element name	Attributes	
`taskdef`	`name`	Must be `validate`.
	`classname`	Must be `com.systinet.policy.tools.ant.ValidateTask`
`validate` (Child of `target`)	`format`	Output format. Takes one of `xml`, `html`, or `txt`
	`policyPropsFile`	Specifies Policy Manager properties file. Usually `../conf/policy-manager.properties`
	`output`	Output file path, such as `C:/opt/reports/` or `C:/tmp/myreport.html`. If file name is not specified, it will match the validated source's name or `summary.txt|xml|html` if it is a summary report.
	`cancel`	Boolean. `true` stops the validation at the first failure.
`policies` (Child of `validate`)	No attributes. Contains a list of all policies to be used for the validation, in nested ANT elements (`fileset/include`).	
`sources` (Child of `validate`)	No attributes. Contains a list of sources to be validated, in nested ANT elements (`uri`, `fileset/include`).	

`validate` ANT Task

```
<?xml version="1.0"?>
<project name="validatetool" default="main">
  <taskdef name="validate" classname="com.systinet.policy.tools.ant.ValidateTask"/>
  <target name="main">
    <validate format="html" policyPropsFile="../conf/policy-manager.properties"
        output="C:/tmp/out">
      <policies>
        <fileset dir="../data/policies/">
          <include name="wsdl-validity.xml"/>
          <include name="systinet-best-practices.xml"/>
        </fileset>
      </policies>
      <sources>
        <uri value="http://api.google.com/GoogleSearch.wsdl"/>
        <fileset dir="../data/policies/">
          <include name="wsdl-validity.xml"/>
        </fileset>
      </sources>
    </validate>
  </target>
</project>
```

Validating Against Policy On Server (server-validate)

Validate a document against a technical policy in an Systinet repository, or remotely run a business policy validation, by running the `server-validate` tool. The tool publishes a report in the same Systinet repository that contains the policy. The URL of the report is printed on the command-line console.

The syntax for validating a document against a technical policy is

`server-validate` [OPTION] {-u *`Systinet username`*} {-p *`Systinet password`*} [-s *`Systinet server URL`*] { *`POLICY_URI`*} {SOURCE_FILE_OR_URI}

. The syntax for running a business policy validation is

`server-validate` [OPTION] {-u *`username`*} {-p *`password`*} [-s *`server URL`*] {-b *`BUSINESS_POLICY_URI`*}

For a full list of options and examples of commands, enter **server-validate --help**.

Policy URIs

Policy URIs are in the following formats:

- Technical policy URI:
  `http|https://`*`host:port`*`/systinet/platform/rest/repository/wsPolicies/`*`policy-name`*
- Business policy URI:
  `http|https://`*`host:port`*`/systinet/platform/rest/repository/businessPolicies/`*`policy-name`*

Source Formats

Only specify a source document if you are validating one against a technical policy. You can write source document locations in the following format:

- As the full URI of the document. For example, `http://api.google.com/GoogleSearch.wsdl`.

Selecting the Systinet Server

By default, the `server-validate` tool communicates with the installation of Systinet from which the validation client was copied. It can use a policy in a different Systinet repository. Specify the Systinet repository with the `-s|--server` option and the URL of the Systinet host. Be careful to use the authorization credentials for that server.

Validation and Report Rendering Demo

This demo shows how to use the Policy Manager REST API to validate a resource. The demo utilizes the `ValidationClient` class. See the Javadoc for a full description of this class.

In this demo, you will learn how to:

- Create a service.
- Create a policy report which uses a technical policy.
- Use this policy report to validate a service.
- View the report.

You can find the demo source code in `SYSTINET_HOME\demos\policymgr\validation\src`

To run the validation demo:

1. Ensure that Systinet is running.
2. Open a command prompt at `SYSTINET_HOME\demos\policymgr\validation`.
3. Enter **run make** to compile the demo source code.
4. Enter **run run** to create the artifacts and run the validation. A link to the HTML report page is printed to the console.

Chapter 12: Publishing Extensibility

This chapter describes how to extend the built-in publishing functionality of Systinet to enable the publishing of custom document types. Specifically, how to create a custom extension that provides support for publishing XML files containing Spring context definitions and then to extend that publishing to provide a decomposition of Spring context definitions into Spring Beans.

Tip: API Docs to accompany the classes described in this chapter are available at `SYSTINET_HOME/doc/publishing/apidocs.zip`. Extract the archive and open `/apidocs/index.html`.

This chapter is split into the following sections:

- "Spring Context Publishing" below
- "Spring Context Decomposition" on page 98

Spring Context Publishing

This section describes how to create an extension for Systinet which enables you to publish Spring Context files using the publishing functionality of Systinet.

The process consists of the following parts:

- Extending the SDM Model.
- Providing custom code components.

The first stage is to create an extension with a custom artifact type definition.

To create a Spring publishing extension:

1. In Workbench Customization Editor, create a new extension project.

   - Select mixed as the project type to be able to make both model and code changes.

     Note: For forward compatibility you should create a model extension for the new artifacts and properties described in this procedure, and a separate code extension for the custom components. Forward compatibility is only guaranteed on model extensions.

   - Select an appropriate name (for example, Spring Publishing), and leave all other options as defaults.

   For more details, see "Creating an Extension Project" in the *Customization Editor Guide*.

2. Create a `Spring Context` artifact type in the extension.

   - Use name Spring Context, Customization Editor populates the artifact properties with default values. If you use different parameters, make sure that you use the same values when you provide the `SpringContextDocType` implementation.
   - Select a suitable package, for example Content.
   - Select XML as the Data Attachment type.
   - For demonstration purposes, make the artifact available in the Service Catalog.
   - Use `springContexts` and `c_springContexts` for the Collection Name and Database Table Name respectively to avoid table name length restrictions.

   For more details, see "Creating an Artifact Type or Package" in the *Customization Editor Guide*.

3. Create an `Imported Spring Context` relationship property in the extension.

   - Use name Imported Spring Context and inverse name Spring Context Imported By.
   - Use Spring Context artifact as both the from and to artifact types.
   - Create the relationship with optional cardinality.
   - Leave the remaining options as defaults.

   For more details, see "Creating a Property" in the *Customization Editor Guide*.

4. Add properties Imported Spring Context and Spring Context Imported By to the Spring Context artifact with multiple cardinality. Make the properties visible in the UI.

5. Create a Transaction Annotation Driven primitive property of boolean type with optional cardinality and add it to the Spring Context artifact. Make the property visible in the UI.

   For more details, see "Creating a Property" in the *Customization Editor Guide*.

6. Create a Context Annotation Config primitive property of boolean type with optional cardinality and add it to the Spring Context artifact. Make the property visible in the UI.

7. Build and apply the extension to Systinet.

   For details, see "Exporting the Extension Project" and "Applying Extensions" in the *Customization Editor Guide*.

   The new artifact type Spring Context should now be visible in the Systinet Service Catalog.

The next step is to extend the publishing framework to handle Spring Context artifacts automatically, for example when using the Upload Data Content functionality in Systinet. This requires a custom implementation of the DocType abstract class.

The following approaches for implementing such a class for XML content are available:

- Extend the XmlDocType class.
- Extend the DocType class and implement an XML interface.

And to provide further support for publishing non-XML content:

- Extend the DocType class and implement a Binary interface.

Note: It is also possible to mark your DocType with the Binary interface even if the content is XML-based. In this case you are provided with both SAX- and binary- callbacks. In particular, it injects the file extension value into the handler. While this may be useful (for example, for type detection based on a file extension if no content is provided in the first place), it is generally not encouraged as extension-based solutions are not as reliable.

Any DocType that is available for WebDAV-based publishing must also be annotated with the DavEnabled notation. As an alternative for XML-based content, you can base your implementation on the DavEnabledXmlDocType class that is already annotated accordingly. Be aware that in the case of WebDAV-based publishing, the source URL is NULL. As a result, should the DocType need to determine the name of the resource from the filename, its target location is more appropriate.

For XML content, extending XmlDocType (or DavEnabledXmlDocType) is the recommended approach as shown in the following example:

SpringContextDocType.java

```
//(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
package demo.publishing.spring;

import com.hp.systinet.publishing.types.DavEnabledXmlDocType;
import com.hp.systinet.publishing.types.ReferenceDefinition;
import com.hp.systinet.repository.sdm.properties.PropertyValue;
import com.hp.systinet.repository.sdm.propertytypes.BooleanProperty;
import com.hp.systinet.repository.sdm.propertytypes.StringProperty;
import com.hp.systinet.repository.sdm.ValuesFactory;
import com.hp.systinet.repository.sdm.ArtifactBase;
import com.hp.systinet.repository.sdm.SdmConstants;
import com.hp.systinet.repository.criteria.filtering.ArtifactFilter;
import com.hp.systinet.repository.criteria.filtering.PropertyFilter;

import javax.xml.namespace.QName;
import java.util.Map;
import java.util.HashMap;
import java.util.List;

import org.springframework.schema.beans.BeansDocument;
import org.apache.xmlbeans.XmlObject;
import org.apache.xmlbeans.XmlAnySimpleType;
import org.apache.xmlbeans.XmlException;
import org.apache.xmlbeans.XmlOptions;
import org.apache.xmlbeans.XmlSaxHandler;

/**
  * DocType for handling XML files with definition of a spring context.
  * <p> *
  * Following is recognized:
  * <ul>
    * <li>import of another context files</li>
    * <li>transaction:annotation-driven property<li>
    * <li>context:annotation-config property<li>
  * </ul>
*/

public class SpringContextDocType extends DavEnabledXmlDocType {
  private static final String NAMESPACE_BEANS =
      "http://www.springframework.org/schema/beans";
  private static final String NAMESPACE_TX =
      "http://www.springframework.org/schema/tx";
  private static final String NAMESPACE_CONTEXT =
      "http://www.springframework.org/schema/context";
  private static final QName ANNOTATION_DRIVEN =
      new QName(NAMESPACE_TX, "annotation-driven");
  private static final QName ANNOTATION_CONFIG =
      new QName(NAMESPACE_CONTEXT, "annotation-config");
  private static final String SDM_TYPE_SPRING_CONTEXT = "c_springContextArtifact";
  private static final String SDM_COLLECTION_SPRING_CONTEXT = "/springContexts/";
  private static final String SDM_RELATION_IMPORT_SPRING_CONTEXT =
      "c_importedSpringContext";
  private static final String SDM_PROPERTY_TRANSACTION =
      "c_transactionAnnotationDriven";
  private static final String SDM_PROPERTY_CONTEXT = "c_contextAnnotationConfig";

  /**
    * The XML describing spring context can include another file using
    * <code>{http://www.springframework.org/schema/beans}import</code> element.
    * File pointed to by the <code>resource</code> attribute will be treated as
    * spring context and it will be linked to its referee by the
    * <code>c_importedSpringContext</code> relationship.
  */
  private static Map<QName, ReferenceDefinition> imports =
      new HashMap<QName, ReferenceDefinition>();
  static {
    imports.put(new QName(NAMESPACE_BEANS, "import"),
    new ReferenceDefinition("resource", SDM_RELATION_IMPORT_SPRING_CONTEXT,
        SDM_TYPE_SPRING_CONTEXT));
  }

  private XmlSaxHandler saxHandler;
  private BeansDocument.Beans beans;
```

```
public SpringContextDocType() {
  super(SDM_COLLECTION_SPRING_CONTEXT, SDM_TYPE_SPRING_CONTEXT, imports);
  // writing SAX-based parser is somewhat tedious, we will use XmlBeans instead
  // register appropriate content&lexical handler for the spring schema type
  XmlOptions xmlOptions = new XmlOptions();
  xmlOptions.setDocumentType(BeansDocument.type);
  saxHandler = XmlObject.Factory.newXmlSaxHandler(xmlOptions);
  setContentHandler(saxHandler.getContentHandler());
  setLexicalHandler(saxHandler.getLexicalHandler());
}

/**
  * Input is recognized as spring context if its XmlBeans representation
  * is built successfully
  * @return true if content is spring context
*/
public boolean recognized() {
  return getBeans() != null;
}

/**
  * Get relevant information gathered from the input.
  * @param valuesFactory factory to be used for property instantiation
  * @return relevant information gathered from the input
*/
public Map<String, PropertyValue>
    getArtifactProperties(ValuesFactory valuesFactory) {
  Map<String, PropertyValue> ret = new HashMap<String, PropertyValue>();
  // get the XmlBeans representation of the content
  BeansDocument.Beans beans = getBeans();
  if(beans != null) {
    if(beans.isSetDescription()) {
      // set description property if it is set in the XML file
      ret.put(SdmConstants.PROPERTY_DESCRIPTION,
          new StringProperty(getTextValue(beans.getDescription())));
    }
    // strip the extension from the resource name
    ret.put(SdmConstants.PROPERTY_NAME,
        new StringProperty(filenameWithoutExtension(getLocation())));
    // set transaction annotation driven property to true if
        there is tx:annotation-driven element
    ret.put(SDM_PROPERTY_TRANSACTION,
        new BooleanProperty(beans.selectChildren(ANNOTATION_DRIVEN).length > 0));
    // set context annotation config property to true if
        there is context:annotation-config element
    ret.put(SDM_PROPERTY_CONTEXT,
        new BooleanProperty(beans.selectChildren(ANNOTATION_CONFIG).length > 0));
  }
```

```
    return ret;
  }

  /**
    * Get find filter to be used when searching for duplicates of given resource.
    * @param artifact artifact used as a source for the duplicate filter
    * @return find filter to be used when searching for duplicates of given
resource
  */
  public ArtifactFilter getDuplicateFilter(ArtifactBase artifact) {
    // default filter defined in DocType requires the last segment of the location
        (i.e. filename) to match
    ArtifactFilter filter = super.getDuplicateFilter(artifact);
    // let's add the requirement that both extra properties we define match too
    filter = filter.combineAnd(new PropertyFilter(SDM_PROPERTY_TRANSACTION,
        artifact.getBooleanProperty(SDM_PROPERTY_TRANSACTION)));
    filter = filter.combineAnd(new PropertyFilter(SDM_PROPERTY_CONTEXT,
        artifact.getBooleanProperty(SDM_PROPERTY_CONTEXT)));
    return filter;
  }

  /**
    * Get list of properties that must be initialized in artifact for construction
    * of the duplicate filter.
    * @return list of properties that must be initialized in artifact for
construction
    * of the duplicate filter
  */
  public List<String> getProperties4DuplicateFilter() {
    // we call super in getDuplicateFilter(), we musn't forget to
        include super properties too
    List<String> list = super.getProperties4DuplicateFilter();
    // list both properties we acquire during filter creation
    list.add(SDM_PROPERTY_TRANSACTION);
    list.add(SDM_PROPERTY_CONTEXT);
    return list;
  }

  public String getFileDescription() {
    return "XML Spring Context definition";
  }

  private String filenameWithoutExtension(String url) {
    return url.replaceFirst("^(.*/)?([^/]*?)(\\.[^/\\.]*)?$", "$2");
  }

  private String getTextValue(XmlObject o) {
    try {
      return XmlAnySimpleType.Factory.parse(o.xmlText()).getStringValue();
    } catch (XmlException e) {
      throw new RuntimeException(e);
    }
  }

  private BeansDocument.Beans getBeans() {
    if(beans == null) {
      try {
        BeansDocument doc = (BeansDocument)saxHandler.getObject();
        if(doc != null) {
          beans = doc.getBeans();
        }
      } catch (Exception e) {
        // couldn't parse
      }
    }
    return beans;
  }
}
```

Place the class file into the `EXTENSION-INF/ui/src/...` folder (maintaining Java package conventions).

XmlBeans are used for in-memory representations of the Spring Context XML files, so you must also provide appropriate schema types with the extension. Make sure you install an XmlBeans tool on your system and run the following command to generate the appropriate schema types:

scomp -out xmlbeans_spring_beans.jar spring-beans-2.5.xsd

Place the resulting xmlbeans_spring_beans.jar into the `EXTENSION-INF/lib/` directory.

Tip: Download `spring-beans-2.5.xsd` from http://www.springframework.org/schema/beans/spring-beans-2.5.xsd.

The final step is register the SpringContextDocType implementation into the system. This is done by registering the implementation of the DocTypeFactory that serves as the factory for creating instances of SpringContextDocType objects.

As SpringContextDocType provides a public parameter-less constructor, you do not need to provide a custom factory implementation and can re-use DocTypeFactoryImpl instead.

extensionContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
  <!-- note that bean ID is not important unless we want to enforce
      order of the factories -->
  <bean id="demo.publishing.docTypeFactory"
```

```
      class="com.hp.systinet.publishing.types.DocTypeFactoryImpl">
    <property name="types">
      <list>
        <!-- list of DocType classes (must have default public ctor)
            this factory instantiates, list our SpringContextDocType here -->
        <value>demo.publishing.spring.SpringContextDocType</value>
      </list>
    </property>
  </bean>
</beans>
```

Place `extensionContext.xml` to the `EXTENSION-INF/ui/src/META-INF` directory.

Build and apply the extension.

To verify that everything works you can publish an archive file containing two files with Spring Contexts available at `SYSTINET_HOME/doc/publishing/upload.zip`

upload.zip.

A built version of this extension is available at `SYSTINET_HOME/doc/publishing/com.systinet.ext.cust.Spring_Publishing.1.1.jar`.

Spring Context Decomposition

This section extends the process described in "Spring Context Publishing" on page 91 with the following features:

- Create artifacts for every named (top-level) bean in the Spring context file.
- Take names and other name properties from the bean ID as defined in the Spring context.
- Establish relationships to other beans.

This additional functionality is also optional, allowing the user to decide whether to create bean artifacts during the upload of the Spring context file.

Note: For demonstration purposes this example does not support:

- Bean aliasing.
- Inner beans.
- Abstract beans and inheritance.
- Anonymous beans.

To automate bean creation from the Spring context file, requires the following overall process:

1. Extend the SDM model.
2. Extend SpringContextDocType to collect information about the beans defined in the context file.
3. Provide custom options for turning on/off bean decomposition.
4. Provide a decomposer that builds a decomposition graph for each context file.
5. Provide a post-processor to establish relationships between decomposed beans.

To extend the model:

1. Create a new artifact, Spring Bean Artifact, to represent beans.

   Use name, Spring Bean, and leave other inputs as defaults.
2. Add a plain text property, `c_otherName`, with multiple cardinality to store alternative bean names.
3. Add a relationship property from Spring Bean to Spring Context, `c_beanDecomposition`, with multiple cardinality and inverse relationship, `c_beanDecompositionOf`.

The beans and relevant properties are obtained from the Spring Context file. To make this data available during decomposition, override the `getCollectedData` method by extending `SpringContextDocType` as shown in the following source code examples:

SpringContextDecomposingDocType.java

```
//(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
package demo.publishing.spring;

import com.hp.systinet.publishing.types.DavEnabledXmlDocType;
import com.hp.systinet.publishing.types.ReferenceDefinition;
import com.hp.systinet.repository.sdm.properties.PropertyValue;
import com.hp.systinet.repository.sdm.propertytypes.BooleanProperty;
import com.hp.systinet.repository.sdm.propertytypes.StringProperty;
import com.hp.systinet.repository.sdm.ValuesFactory;
import com.hp.systinet.repository.sdm.ArtifactBase;
import com.hp.systinet.repository.sdm.SdmConstants;
import com.hp.systinet.repository.criteria.filtering.ArtifactFilter;
import com.hp.systinet.repository.criteria.filtering.PropertyFilter;

import javax.xml.namespace.QName;
import java.util.Map;
import java.util.HashMap;
import java.util.List;
import java.util.ArrayList;

import org.springframework.schema.beans.BeansDocument;
import org.springframework.schema.beans.BeanDocument;
import org.springframework.schema.beans.PropertyType;
import org.apache.xmlbeans.XmlObject;
import org.apache.xmlbeans.XmlAnySimpleType;
import org.apache.xmlbeans.XmlException;
import org.apache.xmlbeans.XmlOptions;
import org.apache.xmlbeans.XmlSaxHandler;

/**
  * DocType for handling XML files with definition of a spring context.
  * <p>
  *
  * Following is recognized:
  * <ul>
    * <li>import of another context files</li>
    * <li>transaction:annotation-driven property<li>
    * <li>context:annotation-config property<li>
  * </ul> */
public class SpringContextDecomposingDocType extends DavEnabledXmlDocType {
  private static final String NAMESPACE_BEANS =
      "http://www.springframework.org/schema/beans";
  private static final String NAMESPACE_TX =
      "http://www.springframework.org/schema/tx";
  private static final String NAMESPACE_CONTEXT =
      "http://www.springframework.org/schema/context";
  private static final QName ANNOTATION_DRIVEN =
      new QName(NAMESPACE_TX, "annotation-driven");
  private static final QName ANNOTATION_CONFIG =
      new QName(NAMESPACE_CONTEXT, "annotation-config");
  private static final String SDM_TYPE_SPRING_CONTEXT = "c_springContextArtifact";
  private static final String SDM_COLLECTION_SPRING_CONTEXT = "/springContexts/";
  private static final String SDM_RELATION_IMPORT_SPRING_CONTEXT =
      "c_importedSpringContext";
  private static final String SDM_PROPERTY_TRANSACTION =
      "c_transactionAnnotationDriven";
  private static final String SDM_PROPERTY_CONTEXT = "c_contextAnnotationConfig";
  static final String BEAN_LIST = "spring.beans";

  /**
    * The XML describing spring context can include another file using
    * <code>{http://www.springframework.org/schema/beans}import</code> element.
    * File pointed to by the <code>resource</code> attribute will be treated as
    * spring context and it will be linked to its referee by the
    * <code>c_importedSpringContext</code> relationship.
  */
  private static Map<QName, ReferenceDefinition> imports =
      new HashMap<QName, ReferenceDefinition>();
  static {
    imports.put(new QName(NAMESPACE_BEANS, "import"),
        new ReferenceDefinition("resource", SDM_RELATION_IMPORT_SPRING_CONTEXT,
            SDM_TYPE_SPRING_CONTEXT));
```

```
}
private XmlSaxHandler saxHandler;
private BeansDocument.Beans beans;
public SpringContextDecomposingDocType() {
  super(SDM_COLLECTION_SPRING_CONTEXT, SDM_TYPE_SPRING_CONTEXT, imports);
  // writing SAX-based parser is somewhat tedious, we will use XmlBeans instead
  // register appropriate content&lexical handler for the spring schema type
  XmlOptions xmlOptions = new XmlOptions();
  xmlOptions.setDocumentType(BeansDocument.type);
  saxHandler = XmlObject.Factory.newXmlSaxHandler(xmlOptions);
  setContentHandler(saxHandler.getContentHandler());
  setLexicalHandler(saxHandler.getLexicalHandler());
}
/**
  * Input is recognized as spring context if its XmlBeans representation
  * is built successfully
  * @return true if content is spring context
*/
public boolean recognized() {
  return getBeans() != null;
}
/**
  * Get relevant information gathered from the input.
  * @param valuesFactory factory to be used for property instantiation
  * @return relevant information gathered from the input
*/
public Map<String, PropertyValue>
    getArtifactProperties(ValuesFactory valuesFactory) {
  Map<String, PropertyValue> ret = new HashMap<String, PropertyValue>();
  // get the XmlBeans representation of the content
  BeansDocument.Beans beans = getBeans();
  if(beans != null) {
    if(beans.isSetDescription()) {
      // set description property if it is set in the XML file
      ret.put(SdmConstants.PROPERTY_DESCRIPTION,
          new StringProperty(getTextValue(beans.getDescription())));
    }
    // strip the extension from the resource name
    ret.put(SdmConstants.PROPERTY_NAME,
        new StringProperty(filenameWithoutExtension(getLocation())));
    // set transaction annotation driven property to true if
        there is tx:annotation-driven element
    ret.put(SDM_PROPERTY_TRANSACTION,
        new BooleanProperty(beans.selectChildren(ANNOTATION_DRIVEN).length > 0));
    // set context annotation config property to true if
        there is context:annotation-config element
    ret.put(SDM_PROPERTY_CONTEXT,
        new BooleanProperty(beans.selectChildren(ANNOTATION_CONFIG).length > 0));
  }
```

```
    return ret;
  }
  /**
    * Get find filter to be used when searching for duplicates of given resource.
    * @param artifact artifact used as a source for the duplicate filter
    * @return find filter to be used when searching for duplicates of given
resource
  */
  public ArtifactFilter getDuplicateFilter(ArtifactBase artifact) {
    // default filter defined in DocType requires the last segment of the location
        (i.e. filename) to match
    ArtifactFilter filter = super.getDuplicateFilter(artifact);
    // let's add the requirement that both extra properties we define match too
    filter = filter.combineAnd(new PropertyFilter(SDM_PROPERTY_TRANSACTION,
        artifact.getBooleanProperty(SDM_PROPERTY_TRANSACTION)));
    filter = filter.combineAnd(new PropertyFilter(SDM_PROPERTY_CONTEXT,
        artifact.getBooleanProperty(SDM_PROPERTY_CONTEXT)));
    return filter;
  }
  /**
    * Get list of properties that must be initialized in artifact for construction
    * of the duplicate filter.
    * @return list of properties that must be initialized in artifact for
construction
    * of the duplicate filter
  */
  public List<String> getProperties4DuplicateFilter() {
    // we call super in getDuplicateFilter(), we musn't forget to
        include super properties too
    List<String> list = super.getProperties4DuplicateFilter();
    // list both properties we acquire during filter creation
    list.add(SDM_PROPERTY_TRANSACTION);
    list.add(SDM_PROPERTY_CONTEXT);
    return list;
  }
  public String getFileDescription() {
    return "XML Spring Context definition";
  }
  /**
    * Collects data about beans defined in the spring context.
    * This data will be later used for decomposition.
  */
  public Map getCollectedData() {
    Map ret = new HashMap();
    BeansDocument.Beans beans = getBeans();
    if(beans != null) {
      List<Bean> list = new ArrayList<Bean>();
      ret.put(BEAN_LIST, list);
      BeanDocument.Bean[] bs = beans.getBeanArray();
      for(BeanDocument.Bean b: bs) {
        Bean bb = new Bean(b.getId(), b.getName());
        if(bb.getId() == null) {
          // we ignore anonymous beans, because we wouldn't be able to identify
              them during update
          continue;
        }
        PropertyType[] ps = b.getPropertyArray();
        for(PropertyType p: ps) {
          if(p.getRef2() != null) {
            bb.addRef(p.getRef2());
          }
        }
        list.add(bb);
      }
    }
    return ret;
  }
  private String filenameWithoutExtension(String url) {
    return url.replaceFirst("^(.*/)?([^/]*?)(\\.[^/\\.]*)?$", "$2");
  }
  private String getTextValue(XmlObject o) {
    try {
      return XmlAnySimpleType.Factory.parse(o.xmlText()).getStringValue();
    } catch (XmlException e) {
      throw new RuntimeException(e);
    }
  }
  private BeansDocument.Beans getBeans() {
    if(beans == null) {
      try {
        BeansDocument doc = (BeansDocument)saxHandler.getObject();
        if(doc != null) {
          beans = doc.getBeans();
        }
      } catch (Exception e) {
        // couldn't parse
      }
    }
    return beans;
  }
}
```

Bean.java

```
//(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
package demo.publishing.spring;

import java.util.List;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Collections;

public class Bean {
  private String id;
  private List<String> names;
  private List<String> refs;
  public Bean(String id, String names) {
    this.names = splitNames(names);
    this.id = (id == null && !this.names.isEmpty())? this.names.remove(0): id;
    this.refs = new ArrayList<String>();
  }
  public String getId() {
    return id;
  }
  public List<String> getNames() {
    return names;
  }
  public List<String> getRefs() {
    return refs;
  }
  public void addRef(String ref) {
    if(!refs.contains(ref)) {
      refs.add(ref);
    }
  }
  private List<String> splitNames(String s) {
    if(s == null) {
      return Collections.emptyList();
    } else {
      return Arrays.asList(s.split("[,;\\s]"));
    }
  }
}
```

Define custom options containing a radio button for turning on/off bean decomposition, as shown in the following source code example:

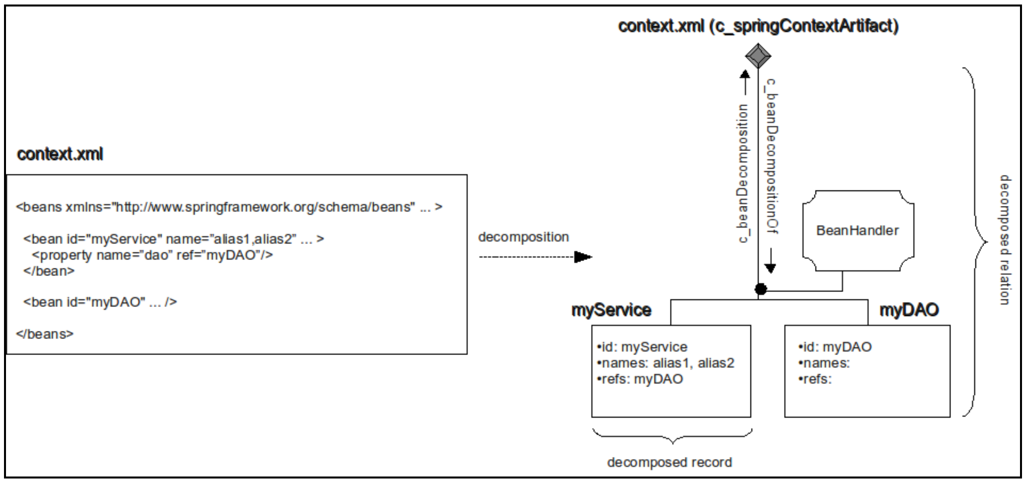
SpringContextOptions.java

```
//(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
package demo.publishing.spring;

import com.hp.systinet.publishing.extensibility.base.AbstractCustomOptions;
import com.hp.systinet.publishing.options.Options;
import com.hp.systinet.publishing.options.ElementaryOption;
import com.hp.systinet.publishing.options.RadioOption;
import com.hp.systinet.repository.sdm.generated.artifacts.C_springContextArtifact;

import java.util.Arrays;
import java.util.HashSet;
import java.util.List;

public class SpringContextOptions extends AbstractCustomOptions {
  public Options getDefaults() {
    // we define single radio option for turning on/off bean decomposition
    RadioOption opt = new RadioOption("spring.decompostion", "Spring
Decomposition",
        "Artifacts to decompose from spring context",
      Arrays.asList("beans", "none"), // keys
      Arrays.asList("Beans", "None"), // captions
      Arrays.asList("Decompose Spring Beans", "Decompose nothing"), // hints
        "beans"); // default key
    return new Options("spring.options", "Spring Publishing",
        Arrays.<ElementaryOption>asList(opt),
        new HashSet<String>(Arrays.asList(C_springContextArtifact.SDM_NAME)));
  }

  /**
    * Utility method for reading the value of the spring decomposition settings.
    * @param list all available publishing options specified for this publishing
process
    * @return true if spring bean decomposition is turned on
  */
  public static boolean isDecomposeBeans(List<Options> list) {
    for(Options options: list) {
      if(options.getId().equals("spring.options")) {
        // we have located our options, check the value
        return "beans".equals(((RadioOption)options.list().get(0)).getSelectedKey
());
      }
    }
    return true; // default if our options were not found
  }
}
```

The option value (selected by the user in the UI advanced options) must be acquired in both the decomposer and post-processor, so `SpringContextOptions`includes a method for obtaining the value.

The decomposer takes information (bean definitions) collected from the content (Spring context files) and builds a decomposition tree describing the subordinate components and their relationships. Information captured in the decomposition tree includes the relational property towards the decomposed resource, as well as the handler for mapping the decomposed resources to artifacts in the repository. In this example, the tree structure is relatively simple, because there is only a single level of decomposition, Spring beans are directly connected to the encapsulating Spring context as shown in the following diagram:

Decomposition



```
<beans xmlns="http://www.springframework.org/schema/beans" ... >

  <bean id="myService" name="alias1,alias2" ... >
    <property name="dao" ref="myDAO"/>
  </bean>

  <bean id="myDAO" ... />

</beans>
```

For details of the decomposer, see the following source code examples:

SpringContextOptions.java

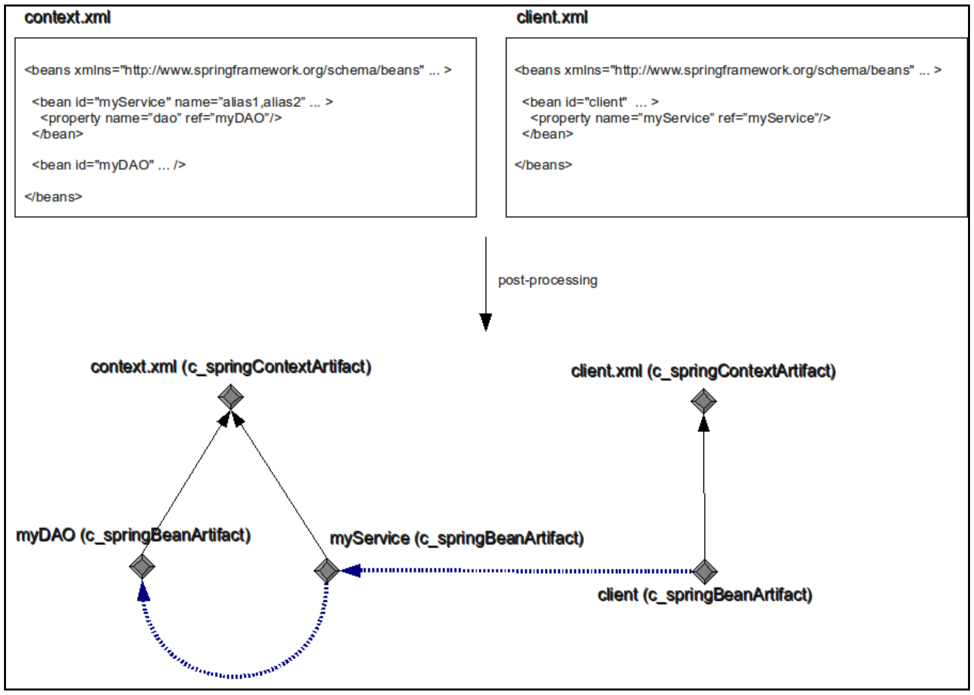
```
//(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
package demo.publishing.spring;

import com.hp.systinet.publishing.extensibility.base.AbstractDecomposer;
import com.hp.systinet.publishing.struct.DecomposedRelation;
import com.hp.systinet.publishing.struct.DecomposedRecord;
import com.hp.systinet.publishing.options.Options;
import com.hp.systinet.publishing.NamedIdentity;

import java.util.Map;
import java.util.List;
import java.util.Arrays;

public class SpringContextDecomposer extends AbstractDecomposer implements
NamedIdentity {
  public List<DecomposedRelation> decompose(Map collectedData,
      List<Options> optionsList) {
    if(collectedData != null) {
      List<Bean> beans =
          (List)collectedData.get(SpringContextDecomposingDocType.BEAN_LIST);
      if(beans != null && SpringContextOptions.isDecomposeBeans(optionsList)) {
        return decompose(beans);
      }
    }
    return null;
  }
  public String getId() {
    return "spring.decomposer";
  }
  public String getName() {
    return "Spring publishing";
  }
  private List<DecomposedRelation> decompose(List<Bean> beans) {
    DecomposedRelation<Bean> ret =
        new DecomposedRelation<Bean>("c_beanDecompositionOf",
            new BeanHandler(), false);
    for(Bean bean: beans) {
      // add decomposition record for every bean we found in the content
      ret.add(new DecomposedRecord<Bean>(bean));
    }
    return Arrays.<DecomposedRelation>asList(ret);
  }
}
```

BeanHandler.java

```
//(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
package demo.publishing.spring;

import com.hp.systinet.publishing.extensibility.base.AbstractArtifactHandler;
import com.hp.systinet.publishing.struct.DecomposedRelation;
import com.hp.systinet.publishing.options.Options;
import com.hp.systinet.repository.sdm.generated.artifacts.C_springBeanArtifact;

import java.util.List;
import java.util.Collections;
import java.util.Arrays;
import java.util.HashSet;

import org.apache.commons.collections.bag.HashBag;

public class BeanHandler extends AbstractArtifactHandler<C_springBeanArtifact,
Bean> {
  public Class<C_springBeanArtifact> getArtifactClass() {
    return C_springBeanArtifact.class;
  }
  public List<String> getPropertyNames() {
    // this is optimization that avoids necessity to explicitly
        fetch these properties later in code:
    // we use c_otherNames and c_dataReference in this handler and
        c_beanReference later in post-processor
    return Arrays.asList("c_otherNames", "c_beanReference", "c_dataReference");
  }
  public boolean setArtifactProperties(C_springBeanArtifact art, Bean item) {
    boolean changed = false;
    if(!item.getId().equals(art.getName())) {
      art.setName(item.getId());
      changed = true;
    }
    if(!new HashBag(art.getC_otherNamesGroup()).equals(new HashBag(item.getNames
())))) {
      art.setC_otherNamesGroup(item.getNames());
      changed = true;
    }
    if(!new HashSet<String>(item.getRefs()).equals
        (new HashSet<String>(art.getC_dataReferenceGroup()))) {
      art.setC_dataReferenceGroup(item.getRefs());
      changed = true;
    }
    return changed;
  }
  public Matcher<C_springBeanArtifact> getMatcherForItem(final Bean item) {
    return new Matcher<C_springBeanArtifact>() {
      public boolean matches(C_springBeanArtifact candidate) {
        // two beans match if their primary name matches
        return item.getId().equals(candidate.getName());
      }
    }
  }
}
```

Relationships between beans cannot be established during the decomposition process, because one bean may reference another bean defined in a separate context file. Referencing may only take place after the decomposition of all context files being published, using publishing post-processor, as shown the following diagram:

Post-Processing


context.xml
client.xml
post-processing
context.xml (c_springContextArtifact)
client.xml (c_springContextArtifact)
myDAO (c_springBeanArtifact)
myService (c_springBeanArtifact)
client (c_springBeanArtifact)

```
<beans xmlns="http://www.springframework.org/schema/beans" ... >

  <bean id="myService" name="alias1,alias2" ... >
    <property name="dao" ref="myDAO"/>
  </bean>

  <bean id="myDAO" ... />

</beans>
```

```
<beans xmlns="http://www.springframework.org/schema/beans" ... >

  <bean id="client"  ... >
    <property name="myService" ref="myService"/>
  </bean>

</beans>
```

The relationships between beans and their encapsulating context (solid black arrows) are established during decomposition. Relationships between beans (dashed blue lines) are added by the post-processor. The complexity of the code depends largely on how smart the process is required to be. For example, the code could be simplified by not requiring late-binding (client.xml published prior to context.xml) to work or alternatively, limit the code to relationship creation and do not remove relationships no longer represented in the data, etc.

For details of the decomposer, see the following source code example:

SpringContextProcessor.java

```
//(C) Copyright 2003-2010 Hewlett-Packard Development Company, L.P.
package demo.publishing.spring;

import com.hp.systinet.publishing.extensibility.base.AbstractProcessor;
import com.hp.systinet.publishing.extensibility.Processor;
import com.hp.systinet.publishing.ProcessorDescriptor;
import com.hp.systinet.publishing.PublisherRecord;
import com.hp.systinet.publishing.NamedIdentity;
import com.hp.systinet.publishing.PublishingLib;
import com.hp.systinet.publishing.struct.ResourceStatus;
import com.hp.systinet.publishing.options.Options;
import com.hp.systinet.repository.sdm.generated.artifacts.C_springBeanArtifact;
import com.hp.systinet.repository.sdm.generated.artifacts.C_springContextArtifact;
import com.hp.systinet.repository.sdm.SdmConstants;
import com.hp.systinet.repository.sdm.propertytypes.DocumentRelationship;
import com.hp.systinet.repository.sdm.properties.Uuid;
import com.hp.systinet.repository.command.FindCommand;
import com.hp.systinet.repository.structures.ArtifactPartSelector;
import com.hp.systinet.repository.RepositoryService;
import com.hp.systinet.repository.criteria.filtering.ArtifactFilter;
import com.hp.systinet.repository.criteria.filtering.PropertyFilter;

import java.util.*;

import org.springframework.beans.factory.annotation.Required;

@ProcessorDescriptor(type = Processor.Type.POST)
public class SpringContextProcessor extends AbstractProcessor implements
NamedIdentity {
  private PublishingLib publishingLib;
  private RepositoryService repositoryService;
  @Required
  public void setPublishingLib(PublishingLib publishingLib) {
    this.publishingLib = publishingLib;
  }
  @Required
  public void setRepositoryService(RepositoryService repositoryService) {
    this.repositoryService = repositoryService;
  }
  public List<PublisherRecord> process(List<PublisherRecord> records,
    List<Options> optionsList, Type type) {
    if(!SpringContextOptions.isDecomposeBeans(optionsList)) {
      // do nothing if bean decomposition is turned off
    }
    // collect all references to other beans (bean ID -> list of referenced bean
IDs)
    Map<String, List<String>> refs = new HashMap<String,
        List<String>>();
    // collect all contexts that are already part of publishing
    Set<Uuid> ctxs = new HashSet<Uuid>();
    for(PublisherRecord record: records) {
      if(record.getArtifact() instanceof C_springContextArtifact) {
        ctxs.add(record.getArtifact().get_uuid());
        List<Bean> beans =
            (List)record.get(SpringContextDecomposingDocType.BEAN_LIST);
        if(beans != null) {
          for(Bean bean: beans) {
            // references
            refs.put(bean.getId(), bean.getRefs());
          }
        }
      }
    }
    // find out beans that we either point to or point to us (if we are newly
created)
    Set<String> allReferences = new HashSet<String>();
    for(List<String> list: refs.values()) {
      allReferences.addAll(list);
    }
    ArtifactFilter filter = getNameFilter(allReferences); // we point to
    ArtifactFilter referenceFilter = getReferenceFilter(records); // point to us
    if(referenceFilter != null) {
      if(filter != null) {
        filter = filter.combineOr(referenceFilter);
      } else {
        filter = referenceFilter;
      }
    }
    List<PublisherRecord> createdRecords =
        new ArrayList<PublisherRecord>();
    Map<String, List<String>> dataRefs =
        new HashMap<String, List<String>>();
    if(filter != null) {
      // find them in repository
      FindCommand find = new FindCommand(C_springBeanArtifact.SDM_NAME);
      find.setArtifactPartSelector(new ArtifactPartSelector("c_
beanDecomposition"));
      find.setFilter(filter.combineAnd(new PropertyFilter
          (SdmConstants.PROPERTY_DELETED, false)));
      List<C_springBeanArtifact> arts = (List)repositoryService.findArtifacts
(find);
      for(C_springBeanArtifact art: arts) {
        if(art.getC_beanDecomposition() ==
            null || art.getC_beanDecomposition().getTargetId() == null) {
          // ignore beans without context (never link to/from them)
          continue;
        }
        if(!ctxs.add(art.getC_beanDecomposition().getTargetId())) {
          // ignore beans of contexts that we are processing
              (they are already loaded or doesn't exist anymore)
          continue;
        }
        // we have found context that contains bean we want to link to/from,
            add context + all its beans to the result
        C_springContextArtifact ctx = (C_springContextArtifact)
            repositoryService.getArtifact(art.getC_beanDecomposition().getTargetId
(),
            new ArtifactPartSelector(SdmConstants.PROPERTY_DATA_SYNC_LAST_CHECKED,
            SdmConstants.PROPERTY_DATA_LOCATION, SdmConstants.PROPERTY_DATA_ORIGIN_
URL,
            "c_beanDecompositionOf"));
        PublisherRecord createdContext = publishingLib.createRecord(ctx,
            ResourceStatus.IDENTICAL);
        createdRecords.add(createdContext);
        for(DocumentRelationship rel: ctx.getC_beanDecompositionOfGroup()) {
          if(rel.getTargetId() != null) {
            C_springBeanArtifact bean = (C_springBeanArtifact)
                repositoryService.getArtifact(rel.getSourceId(),
                new ArtifactPartSelector("c_otherNames", "c_dataReference"));
            PublisherRecord createdBean = publishingLib.createRecord(bean,
                ResourceStatus.IDENTICAL, createdContext);
            publishingLib.addReference(createdBean, createdContext,
                "c_beanDecomposition");
            createdRecords.add(createdBean);
            dataRefs.put(bean.getName(), bean.getC_dataReferenceGroup());
          }
        }
      }
    }
    // make map of all beans that we have
    Map<String, PublisherRecord> beanMap = new HashMap<String,
        PublisherRecord>();
    List<PublisherRecord> list = new ArrayList<PublisherRecord>(records);
    list.addAll(createdRecords);
    for(PublisherRecord record: list) {
      if(record.getArtifact() instanceof C_springBeanArtifact) {
        beanMap.put(record.getArtifact().getName(), record);
        for(String otherName:
```

```
        ((C_springBeanArtifact)record.getArtifact()).getC_otherNamesGroup()) {
          beanMap.put(otherName, record);
        }
      }
    }
    // set bean references...
    for(String beanName: refs.keySet()) {
      PublisherRecord source = beanMap.get(beanName);
      Set<Uuid> targetUids = new HashSet<Uuid>();
      // clear existing reference if any
      ((C_springBeanArtifact)source.getArtifact()).setC_beanReferenceGroup
          (Collections.<DocumentRelationship>emptyList());
      for(String targetName: refs.get(beanName)) {
        PublisherRecord target = beanMap.get(targetName);
        if(target != null) {
          publishingLib.addReference(source, target, "c_beanReference");
          targetUids.add(target.getUuid());
        }
      }
      // check if bean changed due to references
      if(ResourceStatus.IDENTICAL.equals(source.getStatus())) {
        if(targetUids.contains(null)) {
          // referencing something new -> changed
          source.markArtifactChanged();
          continue;
        }
        // relations we have in repository
        HashSet<Uuid> repositoryUuids = new HashSet<Uuid>();
        C_springBeanArtifact art = (C_springBeanArtifact)source.getArtifact();
        List<DocumentRelationship> l = art.getC_beanReferenceGroup();
        for(DocumentRelationship rel: l) {
          if(rel.getTargetId() != null) {
            repositoryUuids.add(rel.getTargetId());
          }
        }
        if(!repositoryUuids.equals(targetUids)) {
          source.markArtifactChanged();
        }
      }
    }
    // set references on implicitly loaded
    for(String beanName: dataRefs.keySet()) {
      PublisherRecord source = beanMap.get(beanName);
      for(String targetName: dataRefs.get(beanName)) {
        PublisherRecord target = beanMap.get(targetName);
        if(target != null && target.getUuid() == null) {
          publishingLib.addReference(source, target, "c_beanReference");
          publishingLib.setIncremental(source, "c_beanReference", true);
          // only adding references
          source.markArtifactChanged();
        }
      }
    }
    // for all unsynchronized beans, remove relations to other beans
    for(PublisherRecord record: records) {
      if(record.getArtifact() instanceof C_springBeanArtifact &&
        ResourceStatus.UNSYNCHRONIZED.equals(record.getDisplayStatus())) {
          // clear outgoing references if any
          ((C_springBeanArtifact)record.getArtifact()).setC_beanReferenceGroup
              (Collections.<DocumentRelationship>emptyList());
        }
        // NOTE: we don't remove relations to UNSYNCHRONIZED
    }
    return createdRecords;
  }
  private ArtifactFilter getReferenceFilter(List<PublisherRecord> records) {
    ArtifactFilter ret = null;
    for(PublisherRecord record: records) {
      if(ResourceStatus.NEW.equals(record.getStatus()) &&
          record.getArtifact() instanceof C_springBeanArtifact) {
        if(ret == null) {
          ret = new PropertyFilter("c_dataReference", record.getArtifact().getName
());
        } else {
          ret = ret.combineOr(new PropertyFilter("c_dataReference",
              record.getArtifact().getName()));
        }
        for(String otherName:
            ((C_springBeanArtifact)record.getArtifact()).getC_otherNamesGroup()) {
          ret = ret.combineOr(new PropertyFilter("c_dataReference", otherName));
        }
      }
    }
    return ret;
  }
  private ArtifactFilter getNameFilter(Set<String> names) {
    ArtifactFilter ret = null;
    for(String name: names) {
      if(ret == null) {
        ret = new PropertyFilter(SdmConstants.PROPERTY_NAME, name).combineOr
            (new PropertyFilter("c_otherNames", name));
      } else {
        ret = ret.combineOr(new PropertyFilter(SdmConstants.PROPERTY_NAME,
            name).combineOr(new PropertyFilter("c_otherNames", name)));
      }
    }
    return ret;
  }
  public String getId() {
    return "spring.processor";
  }
  public String getName() {
    return "Spring publishing";
  }
}
```

When you build and apply the new extension (available at `SYSTINET_HOME/doc/publishing/com.systinet.soa.ext.cust.Spring_Publishing.1.2.jar`Spring Decomposition Extension), you can verify bean decomposition by uploading the archive, `SYSTINET_HOME/doc/publishing/uploadAdvanced.zip` uploadAdvanced.zip, which includes the files described in the Figure, "Post-Processing" on page 108.

Send Documentation Feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Developer Guide (Systinet 10.01)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to docteam_systinet@hp.com.

We appreciate your feedback!