HP SOA Systinet

Software Version: 2.52, Standard Edition

Developer Guide

Document Release Date: November 2007
Software Release Date: November 2007

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Third-Party Web Sites

Mercury provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. Mercury makes no representations or warranties whatsoever as to site content or availability.

Copyright Notices

Trademark Notices

Java™ is a US trademark of Sun Microsystems, Inc. Microsoft®, Windows® and Windows XP® are U.S. registered trademarks of Microsoft Corporation. IBM®, AIX® and WebSphere® are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries. BEA® and WebLogic® are registered trademarks of BEA Systems, Inc.

Contents

Welcome to This Guide

Welcome to HP SOA Systinet, the foundation of Service Oriented Architecture, providing an enterprise with a single place to organize, understand, and manage information in its SOA. The standards-based architecture of SOA Systinet maximizes interoperability with other SOA products.

How This Guide is Organized

SOA Systinet Developer Guide describes additional features and methods to enable developers to better interact with SOA Systinet.

It contains the following chapters:

- **IDE Integration on page 11 .** How to integrate SOA Systinet with IDEs
- **REST Interface on page 15 .** A guide to the REST Interface
- **REST Client on page 53 .** Using the REST Client
- **SDM Client on page 59 .** Using the SDM Client
- **Technical Security on page 65 .** A technical overview of SOA Systinet from the security point of view.
- **RSS on page 75 .** The RSS format used in SOA Systinet
- **Custom Source Parsers on page 77 .** How to write your own source parser
- **Custom Validation Handlers on page 81 .** How to write your own validation handler

Document Conventions

The typographic conventions used in this document are:

run.bat make	Script name or other executable command plus mandatory arguments.
[--help]	A command-line option.
either | or	A choice of arguments.
replace_value	A command-line argument that should be replaced with an actual value.
{arg1 | arg2}	A choice between two command-line arguments where one or the other is mandatory.
`rmdir /S /Q System32`	Operating system commands and other user input that you can type on the command line and press **Enter** to invoke. Items in *italics* should be replaced by actual values.
`C:\System.ini`	Filenames, directory names, paths and package names.
`a.append(b);`	Program source code.
`server.Version`	An inline Java or C++ class name.
`getVersion()`	An inline Java method name.
Shift-N	A combination of keystrokes.
Service View	A label, word or phrase in a GUI window, often clickable.
OK	A button in a GUI window.
New->Service	Menu choice.

Documentation Updates

This manual's title page contains the following identifying information:

- Software version number
- 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://ovweb.external.hp.com/lpe/doc_serv/

Support

Mercury Product Support

You can obtain support information for products formerly produced by Mercury as follows:

- If you work with an HP Software Services Integrator (SVI) partner (**http://h20230.www2.hp.-com/svi_partner_list.jsp**), contact your SVI agent.
- If you have an active HP Software support contract, visit the HP Software Support Web site and use the Self-Solve Knowledge Search to find answers to technical questions.
- For the latest information about support processes and tools available for products formerly produced by Mercury, we encourage you to visit the Mercury Customer Support Web site at: **http://hp.-com/go/hpsoftwaresupport**.
- For the latest information about support processes and tools available for products formerly produced by Systinet, we encourage you to visit the Systinet Online Support Web site at: **http://www.systinet.-com/support/index**.
- If you have additional questions, contact your HP Sales Representative.

HP Software Support

You can visit the HP Software Support Web site at:

http://www.hp.com/go/hpsoftwaresupport

HP Software online support provides an efficient way to access interactive technical support tools. As a valued support customer, you can benefit by using the support 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 find more information about access levels, go to: http://h20230.www2.hp.-com/new_access_levels.jsp

To register for an HP Passport ID, go to: http://h20229.www2.hp.com/passport-registration.html

1 IDE Integration

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

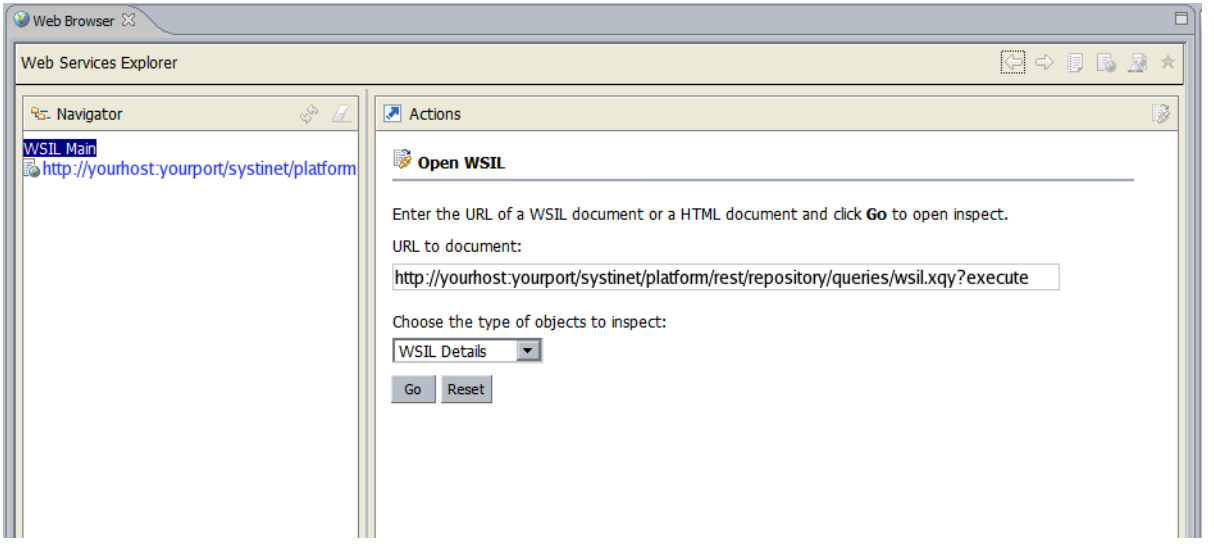
There are the following sections:

- **WSIL Report – IBM RAD and Eclipse on page 11.** How to use the WSIL query include with SOA Systinet to add it to an IDE
- **Microsoft Visual Studio on page 12.** How to add SOA Systinet as a Web Reference in MS 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 SOA 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 **Search** in the Menu on all pages or at the referenced location `http://yourhost:yourport/soa/systinet/platform/rest/service/system/wsil`

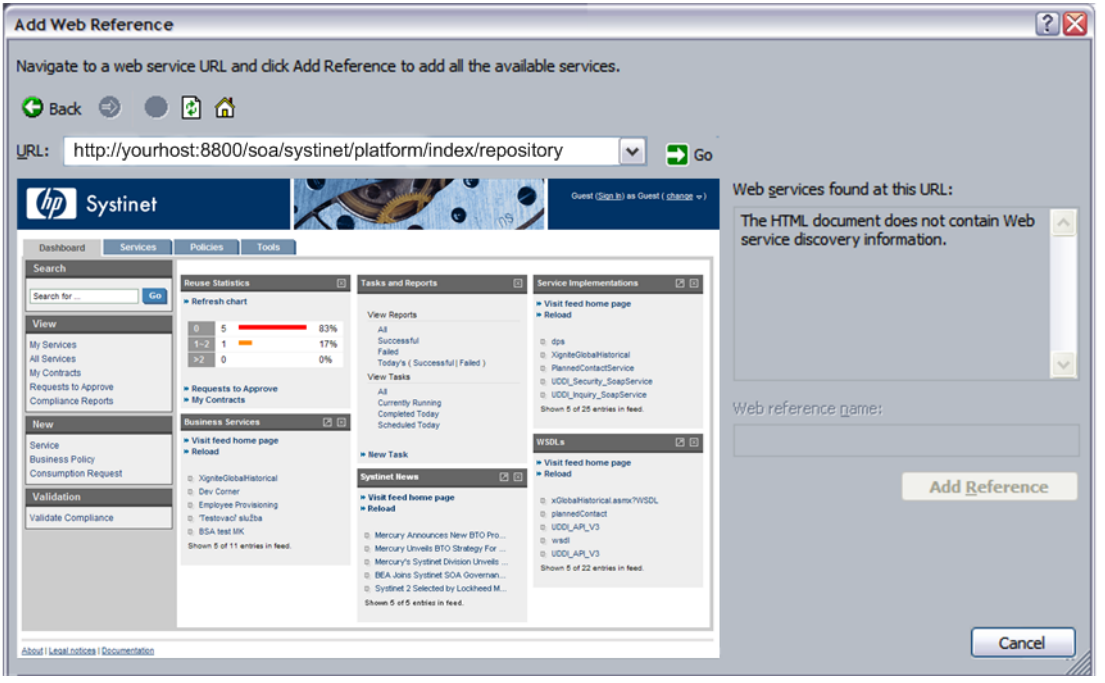
Launch IBM RAD 6.0's Web Services Explorer and enter the WSIL report URL (the page that is generated by the WSIL link of **Search**.



From there you will be able to 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 SOA Systinet installation (for example `http://yourserver:8080/soa/`) to access SOA 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 SOA 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 the Full Text Search section in the HP Systinet User Guide.

Select the WSDL artifact for the service.

From this page you can access the WSDL document by clicking **Cached version**. At this point Microsoft Visual Studio's Solution Explorer recognizes that the document accessed is WSDL. You can now click **Add Reference** to read the web service definition(s) into Microsoft Visual Studio.

2 REST Interface

The types of resource that the REST interface provides access to are described in the Repository Element Formats section in the HP Systinet Reference Guide. That is, a hierarchy of collections each containing:

- Collections
- Documents, of type:
  - XML
  - Binary

This chapter contains the following sections:

- **REST Interface URIs on page 15.** The set of HTTP URL paths identifying resources
- **Resource Representations on page 22.** The REST view of resources
- **REST Operations on page 28.** REST operation mapping to HTTP methods and URL query parameters
- **REST Exceptions on page 46.** HTTP response formats
- **Executable Objects on page 49.** How REST handles reports and queries

REST Interface URIs

As the interface uses HTTP, the URIs used to access it are HTTP URLs. Table 1 on page 17 explains how different parts of a URL are used for different purposes including:

- Specifying a WS endpoint provided by the REST interface
- Identifying an existing resource

- Specifying details of an operation, in conjunction with the HTTP method
- Specifying how a resource is represented, in conjunction with the HTTP `Content-type` header

Briefly, a REST interface URL has this form:

```
protocol://hostname:port/soa/systinet/platform/interface/namespace/{collection/}*document?
```

Table 1. Components of a REST Interface HTTP URL

Component	Description
`protocol`	http or https if using SSL
`hostname`	The host where the JEE server running SOA Systinet is installed. It can also be the host where the loadbalancer or proxy runs.
`port`	The port depends on the configuration of the JEE server. The default installation on JBoss uses: • `8080` if the protocol is http • `8843` if the protocol is https The port must be specified unless it is the default configured for the protocol in a browser.
`soa/systinet/platform/`	SOA Systinet context path. `soa/` is the context of the application in the JEE container.
`interface`	This completes the service path. For REST it is either: • `rest` for anonymous access or internal request using internal SSO authentication • `restBasic` for access using HTTP Basic authentication
`namespace`	This field allows more than one database storage. It can be also use as a namespace for non-persistent objects. For SOA Systinet two values are legal: • `repository` is the database storage for collections and documents • `service` is a namespace where functional resource reside.
`collection`	The remainder of the path identifies a resource in the repository. The first `/` identifies the root collection. Each subsequent `/` separates a collection from a resource it contains. Zero or more additional collections can be specified in this way.

Component	Description
`document`	A single document resource name may be present to complete the identification of the resource. If none is present then the resource is the collection specified by the preceding fields.
`query string`	An HTTP URL query string may encode additional options and parameters necessary to fully specify a REST operation: • The choice of REST operation is encoded in an HTTP request using the HTTP method and/or a query string field. See REST Operations on page 28 • Support for the creation of resources requires that the new resource is specified as a query parameter relative to an existing collection (specified in the URL path) See REST Operations on page 28 • Parameters of the representation type may appear in the query string. See Resource Representations on page 22 In the following *query parameter* is used to refer to fields of the form `name=value`. Boolean options are specified by the appearance or absence of a field containing only an option identifier, and are referred to as *query fields*. A complete list of query parameters is listed in Table 2 on page 19

Table 2. REST URL Query Parameters

Category	Query parameter	Operation	Value	Description
operation	create	N/A	no value	Denotes the CREATE operation on HTTP POST
operation	update	N/A	no value	Denotes the UPDATE operation on HTTP POST
operation	delete	N/A	no value	Denotes the DELETE operation on HTTP POST
operation	undelete	N/A	no value	Denotes the UNDELETE operation on HTTP POST
operation	purge	N/A	no value	Denotes the PURGE operation on HTTP POST
operation	get	N/A	no value	Denotes the GET operation on HTTP POST.
operation	exist	N/A	no value	Denotes the EXIST operation on HTTP GET.
resource identification	resource	C	string	Specifies the name and type of resource to create. If the name ends with "/", then a collection is created. Otherwise a document is created. If not given, then a document is created with a generated name.
resource identification	resource	G,U,D,Un,P	string	Identifies the name of a subresource in the collection given by request URI
resource identification	revision	G	integer, >=0	Identifies the resource revision to GET
resource identification	revision	U,D,P	integer, >=0	Identifies the resource revision to UPDATE/DELETE/PURGE. If the given revision is not the last resource revision, then the UPDATE/DELETE/PURGE operation fails.

Category	Query parameter	Operation	Value	Description
resource identification	datetime	G	datetime in ISO8601 based format (e.g., `2006-07-11T11:28:51.348Z`)	Datetime identifying the resource revision to GET (timeslice)
resource identification	datetime	U,D,P	datetime in ISO8601 based format (e.g., `2006-07-11T11:28:51.348Z`)	Datetime identifying the resource revision to UPDATE/DELETE/PURGE. If the given revision is not the last resource revision, then the UPDATE/DELETE/PURGE operation fails (timeslice).
resource identification	original	G	no value	Returns unresolved data, such as when a WSDL is imported and contains unresolved imports
resource identification	deleted	G	no value	Returns only deleted subresources in a collection listing. If not given, then only live subresources are listed.
request data param	contentType	C	string	Sets the content-type metadatum of a created resource. If not given, then the HTTP header Content-Type header is used (for RAW request message) or content-type metadatum in passed metadata (for REST resource serialization)
representation	meta	G	no value	If given, then the REST resource serialization is returned with a set metadata section
representation	desc	G	no value	If given, then the REST resource serialization is returned with a set descriptor section
representation	desc	C,U	no value	If given, then the REST response to a CREATE/UPDATE operation will contain the descriptor section of the created/updated resource

Category	Query parameter	Operation	Value	Description
representation	data	G	no value	If given, then the REST resource serialization is returned with a set data section
representation	acl	G	no value	If given, then the REST resource serialization is returned with a set acl section
representation	acl	C,U	no value	If given, then the REST response to CREATE/UPDATE operation will contain the ACL section of created/updated resource
representation	history	G	no value	If given, then the REST resource serialization is returned with the data section filled with all resource revisions
representation	rss	G	optional string	If given, then the RSS of the resource is returned. An optional value specifies the RSS format (for example `atom_0.3`; for the full list of values see RSS on page 75).
representation	view	G	no value	If given, then the HTTP redirect (302) to URL of the resource UI representation (for browsers)
execution	execute	G	no value	Executes the resource. It is bound to HTTP GET
execution	ex_	G	string	Prefix for execute parameters. A functional resource can accept parameters without this prefix, however the prefix can be used for backward compatibility.
execution	fulltext	G	string	Fulltext search query string
processing parameter	chp_	G,U,D,Un,P	string	Prefix for a collection handler parameter (for example, 'chp_foo=bbb' for parameter 'foo')

Category	Query parameter	Operation	Value	Description
response data parameter	style	G,U,D,Un,P	URI	URI of an XSLT stylesheet for a response transformation. The URI can be absolute or relative. A relative URI is resolved to a repository base URL.
response data parameter	st_	G,U,D,Un,P	string	Prefix for a stylesheet parameter (for example, 'st_foo=bbb' for parameter 'foo')
response data parameter	export	G,U,D,Un,P	optional string	Overrides HTTP header Content-Type. If no value is given, then application/octet-stream is used.
search	cs_	G	string	Prefix for parameter that limits result of collection listing to artifact matching the condition.
search	co_	G	(integer,)?asc|desc	Prefix for parameter that specifies order of artifact in collection listing.
paging	page	G	positive integer	If specified only the resource on the page with the number are included in collection listing.
paging	pageSize	G	positive integer	The page size. Default value is 30.

Resource Representations

REST Representations are views of resources. In SOA Systinet they are HTTP messages using one of the two models shown in the following table:

Table 3. Models of Communication

Model	Description
Raw	This is the default (if no URL query fields configure the XML model) The resource is represented by its raw data. For documents the raw data is contained in the message body (for both request and response) or in the first part of a multi-part request message resulting from the use of a file upload on an HTML form. (See RFC1867 [http://www.zvon.org/tmRFC/RFC1867/Output/chapter1.html].) In a request the type of the raw data is specified with an HTTP `Content-type` value, either in the message header or in the message part containing the data. In response the Content-Type is set to the value stored in the resource's metadatum content-type. It is *not* `application/xml`, because it is used for REST resource serialization model. For collections, an XML serialization is always used in place of a native format (see Example 1 on page 24).
XML REST resource serialization	This model is requested by specific URL query fields in the HTTP request as described in XML REST Resource Serialization Model on page 25. The resource is represented in the canonical XML format described by the XML schema in the Resource Serialization Schema section in the HP SOA Systinet Reference Guide. This supports the content types specified by the URL query fields. The namespace prefix `rest:` is used for its target namespace in HTTP responses. The HTTP `Content-type` header is `application/xml`.

Representations of resources are encoded in the HTTP message body. Type information is also given in:

- The HTTP `content-type` header
- URL query parameters

Example 1: XML Serialization of a Single-item Collection

```
<?xml version="1.0" encoding="UTF-8"?>
<rest:resource xlink:href="http://localhost:8080/soa/systinet/platform/rest/repository/test/"
    xml:base="http://localhost:8080/soa/systinet/platform/rest/repository/"
 type="collection" name="test/"
 requestURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/"
 readURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/?meta&amp;desc&amp;data"
 revisionURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/?revision=1"
 updateURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/?update"
 updateRevisionURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/?update&amp;revision=1"

 purgeURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/?purge"
 viewURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/?view"
 createURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/?create"
 aclURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/?acl"
 revision="1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:a="http://systinet.com/2005/05/soa/model/artifact"
 xmlns:r="http://systinet.com/2005/05/repository"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:pt="http://systinet.com/2005/05/soa/model/property/type"
 xmlns:rest="http://systinet.com/2005/05/soa/resource"
 xmlns:p="http://systinet.com/2005/05/soa/model/property"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:g="http://systinet.com/2005/05/soa/model/propertyGroup">
  <rest:data  representation="list">
    <rest:resource  xlink:href="http://localhost:8080/soa/systinet/platform/rest/repository/test/a"
     type="document"
  name="a"
  readURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?meta&amp;desc&amp;data"

  revisionURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?revision=1"
  updateURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?update"

updateRevisionURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?update&amp;revision=1"

  deleteURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?delete"

deleteRevisionURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?delete&amp;revision=1"

  undeleteURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?undelete"
  purgeURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?purge"
  purgeRevisionURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?purge&amp;revision=1"
```

```
   viewURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?view"
   viewRevisionURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?view&amp;revision=1"

   aclURI="http://localhost:8080/soa/systinet/platform/rest/repository/test/a?acl"
   revision="1"/>
   </rest:data>
 </rest:resource>
```

XML REST Resource Serialization Model

Messages that use the XML model (see Table 3 on page 23) have XML content in the format described in the Repository Element Formats section in the HP SOA Systinet Reference Guide. This has a number of optional elements meeting the various requirements of the REST interface.

The following messages are in the XML REST resource serialization model:

- Responses
  - CREATE, UPDATE, DELETE, UNDELETE and PURGE response messages

    Returned `Resource` contains the metadata of the requested resource.
  - GET response message

    Requests, which have set `data`, `meta`, `acl`, `desc` in the request URL. Returned `Resource` contains parts according to the request.
  - GET on a collection response message

    If there is no `data`, `meta`, `acl` nor `desc` in the request URL, then the default is `data`. Returned `Resource` contains parts according to the request.

    If the request for collection's data contains `meta` and/or `desc` representation option, the metadata or descriptor of the documents are also included. Note, however, that these data are not fully processed and for example information about relations is not included.

- Request
  - CREATE and UPDATE request message

`Resource` in a request message has set resource parts.

For full details see the Repository Element Formats section in the HP SOA Systinet Reference Guide. The root element of the representation is described and has a number of attributes designed for the REST interface.

Resource Revision Identification

Each resource modification creates a new resource revision. Each revision is identified by these parameters:

- Revision number

  The revision number can be specified by a URI parameter `revision` (Table 2 on page 19), where the value is the revision number. The first revision (a new resource) is `1`. Value `0` represents the latest revision.

- Timeslice (datetime)

  Timeslice identifies the revision using the value of parameter `datetime` (Table 2 on page 19).

Note:

- Collection and document revision semantics differ - the collection revision is incremented by a meta update, not on create/update/delete/purge of subresources. Document revision is incremented by a meta/descriptor/data update.
- Revision is not incremented by an incoming relationship creation (i.e., if another resource establishes a relationship targeting the resource)

The following table presents a combination of `revision` and `datetime` parameters.

Table 4. URL Query Parameters: revision and datetime

Parameters	GET url	relationship in CREATE, UPDATE	UPDATE, DELETE, PURGE
no `revision`, `datetime`	returns last revision	targets last revision	resource's last revision
`revision=0`	returns last revision	targets last revision	resource's last revision
`revision>=1`	returns the content of the given revision number. Relationships are as they were at datetime just before revision ended or at the current datetime.	targets given revision	resource must be in given revision
`datetime=DT`	Returns the content of the revision at the given DT. Relationships are as they were at DT (timeslice). Relationships in metadata and descriptor have set `datetime` parameter of DT	targets revision at given DT	resource revision in DT must be the last revision
`revision=0`, `datetime=DT`	see `datetime=DT`	targets last revision	see `datetime=DT`
`revision>=1`, `datetime=DT`	Returns the content of the given revision number. Relationship are as they were at given DT (if DT falls into datetime of given revision, otherwise datetime just before revision end of current datetime. Relationships in metadata and descriptor have set `datetime` parameter of DT	targets given revision (datetime is ignored)	resource must be in given revision

Note: If datetime has no specified value (DT), then it is set as follows:

- if no revision parameter is given or `revision=0`, then it DT is set to datetime when request is processed (i.e., current datetime)
- if revision>=1, then `DT` is set to datetime of the revision

REST Operations

To use the SOA Systinet REST interface applications must map each operation to an HTTP request. See Table 5 on page 29.

SOA Systinet REST operations map to HTTP GET, POST and HEAD only, because these requests can be received by servers or via proxies that do not support HTTP PUT and DELETE.

Each REST operation is executed on a resource identified by a request URL. For CREATE, the URL identifies an existing collection in which the new resource will be created. The query parameter `resource` names the new resource.

For other operations (GET, EXIST, UPDATE, DELETE, UNDELETE and PURGE) the resource is identified in two ways. Either:

- the URL is consistent with CREATE and specifies:
  - a collection, in the path;
  - a resource, in query parameter `resource`;

  or
- the complete path to the resource is specified in the path and there is no resource parameter;

The name of a collection is always followed by a `/`, whether in the path or in the resource query parameter. When a resource is created the new resource is a collection if and only if the resource parameter includes a trailing `/`.

Table 5. Summary of REST Operations

REST Operation	HTTP method	Query Field	Notes
CREATE	POST	`create`	The path specifies the containing collection and the `resource` URL parameter contains the name of the resource to create (if omitted, then a unique name is generated by the server)
GET	GET	None	The data represented in the response depends on the request.
GET	POST	`get`	It is possible to convert all GET requests to POST requests using `get` and moving parameters from query part to a body of type `multipart/form-data`. However this is not recommended unless the size of parameters is too high.
EXIST	GET	`exist`	Used to check if the resource exists.
EXIST	HEAD	none	Used to check if the resource exists.
UPDATE	POST	`update`	Updates the resource.
DELETE	POST	`delete`	Only for documents. GET, UNDELETE and PURGE operations can be run on deleted resources.
UNDELETE	POST	`undelete`	Undeletes the deleted resource, it can then be updated again.
PURGE	POST	`purge`	Purge physically removes a resource.

For operations other than GET and EXIST the response contains a message from the XML REST resource serialization model, where `Resource` contains the metadata of the requested resource and optionally the descriptor and/or acl sections (when the request URL contains desc and/or acl query parameters).

CREATE

The request message can contain any model of representation.

For collections, it is not possible to specify the data and so only metadata can optionally be represented.

The created document type is `XML` if the content-type of the data is `text/xml`, otherwise it is `binary`. It is not possible to change the document type using update.

Example 2: Create collection `c/` at the root collection.

```
POST http://localhost:8080/soa/systinet/platform/restBasic/repository/?resource=c/&create
```

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.

Example 3: Response to Example 2 on page 30

```
<?xml version="1.0" encoding="UTF-8"?>
<rest:resource
    xlink:href="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/"
    xml:base="http://localhost:8080/soa/systinet/platform/restBasic/repository/"
    type="collection"
    name="c/"
   requestURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/?create=&amp;resource=c%2F"

   readURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?meta&amp;desc&amp;data"

    revisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?revision=1"
    updateURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?update"

updateRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?update&amp;revision=1"

    purgeURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?purge"
    viewURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?view"
    createURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?create"
    aclURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?acl"
    revision="1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:a="http://systinet.com/2005/05/soa/model/artifact"
    xmlns:r="http://systinet.com/2005/05/repository"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:pt="http://systinet.com/2005/05/soa/model/property/type"
    xmlns:rest="http://systinet.com/2005/05/soa/resource"
    xmlns:p="http://systinet.com/2005/05/soa/model/property"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:g="http://systinet.com/2005/05/soa/model/propertyGroup">
  <rest:metadata>
    <rest:path>c/</rest:path>
    <rest:collection></rest:collection>
    <rest:binary>0</rest:binary>
    <rest:type>collection</rest:type>
    <rest:deleted>0</rest:deleted>
    <rest:owner>demouser</rest:owner>
    <rest:revision>
      <rest:number>1</rest:number>
      <rest:timestamp>2007-04-25T16:01:35.031Z</rest:timestamp>
      <rest:creator>demouser</rest:creator>
      <rest:label xsi:nil="true"/>
      <rest:last>1</rest:last>
    </rest:revision>
```

```
    <rest:relationships/>
    <rest:cached>0</rest:cached>
    <rest:checksum>0</rest:checksum>
  </rest:metadata>
</rest:resource>
```

Example 4: Create an XML document

```
POST http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?resource=d&create
```

```
POST /soa/systinet/platform/restBasic/repository/c/?create&resource=d HTTP/1.1
User-Agent: Systinet Server for Java/5.5 (Java/1.4.2_10; Windows XP/5.1; build SSJ-5.5-20070426-0008)
Host: localhost:8080
Transfer-Encoding: chunked
Connection: keep-alive
Content-type: text/xml; charset=utf-8
Authorization: Basic ZGVtb3VzZXI6Y2hhbmdlaXQ=

4
<a/>
0
```

Example 5: Response to Example 4 on page 32

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.4; JBoss-4.0.5.GA (build: CVSTag=Branch_4_0 date=200610162339)/Tomcat-5.5
Location: http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d
Content-Type: application/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Thu, 26 Apr 2007 11:35:30 GMT

ab0
<?xml version="1.0" encoding="UTF-8"?>
<rest:resource xlink:href="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d"
   xml:base="http://localhost:8080/soa/systinet/platform/restBasic/repository/" type="document" name="d"

   requestURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?create=&amp;resource=d"

   readURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?meta&amp;desc&amp;data"

    revisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?revision=1"
    updateURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update"

updateRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update&amp;revision=1"

    deleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete"

deleteRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete&amp;revision=1"

    undeleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?undelete"
    purgeURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge"

purgeRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge&amp;revision=1"

    viewURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view"

viewRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view&amp;revision=1"

    aclURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?acl" revision="1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:a="http://systinet.com/2005/05/soa/model/artifact"
    xmlns:r="http://systinet.com/2005/05/repository" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:pt="http://systinet.com/2005/05/soa/model/property/type"
    xmlns:rest="http://systinet.com/2005/05/soa/resource"
xmlns:p="http://systinet.com/2005/05/soa/model/property"
   xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:g="http://systinet.com/2005/05/soa/model/propertyGroup">
```

```
 <rest:metadata>
  <rest:path>c/d</rest:path>
  <rest:collection>c/</rest:collection>
  <rest:binary>0</rest:binary>
  <rest:contentType>text/xml; charset=utf-8</rest:contentType>
  <rest:type>document</rest:type>
  <rest:deleted>0</rest:deleted>
  <rest:owner>demouser</rest:owner>
  <rest:revision>
   <rest:number>1</rest:number>
   <rest:timestamp>2007-04-26T11:35:29.812Z</rest:timestamp>
   <rest:creator>demouser</rest:creator>
   <rest:label xsi:nil="true"/>
   <rest:last>1</rest:last>
  </rest:revision>
  <rest:relationships/>
  <rest:cached>0</rest:cached>
  <rest:checksum>0</rest:checksum>
 </rest:metadata>
</rest:resource>
0
```

GET

Table 5 on page 29 shows that the REST GET differs significantly from other operations. In particular there is no need to specify the operation with a query parameter because it is one of two operations that maps to HTTP GET.

The response contains a representation of the resource depending on the request. Parameters can specify which representation is required. The default is Raw. See Resource Representations on page 22 for details.

See also Executable Objects on page 49 about REST GET operations that execute the resource they operate on and return the execution result as a REST representation.

Example 6: Get collection `/c/`

```
GET http://localhost:8080/soa/systinet/platform/restBasic/repository/c/
```

Example 1 on page 24 shows the content of a response to a similar request.

Example 7: Get the XML serialization of a document

```
http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?meta&data&acl
```

Example 8: Response to Example 7 on page 35

```
<?xml version="1.0" encoding="UTF-8"?>
<rest:resource xlink:href="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d"
   xml:base="http://localhost:8080/soa/systinet/platform/restBasic/repository/"
   type="document"
   name="d"

requestURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?acl=&amp;data=&amp;meta="

   readURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?meta&amp;desc&amp;data"

   revisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?revision=1"
   updateURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update"

updateRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update&amp;revision=1"

   deleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete"

deleteRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete&amp;revision=1"

   undeleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?undelete"
   purgeURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge"

purgeRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge&amp;revision=1"

   viewURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view"

viewRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view&amp;revision=1"

   aclURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?acl"
   revision="1"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:a="http://systinet.com/2005/05/soa/model/artifact"
   xmlns:r="http://systinet.com/2005/05/repository"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:pt="http://systinet.com/2005/05/soa/model/property/type"
   xmlns:rest="http://systinet.com/2005/05/soa/resource"
   xmlns:p="http://systinet.com/2005/05/soa/model/property"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:g="http://systinet.com/2005/05/soa/model/propertyGroup">
  <rest:metadata>
    <rest:path>c/d</rest:path>
    <rest:collection>c/</rest:collection>
    <rest:binary>0</rest:binary>
```

```
    <rest:contentType>text/xml; charset=utf-8</rest:contentType>
    <rest:type>document</rest:type>
    <rest:deleted>0</rest:deleted>
    <rest:owner>demouser</rest:owner>
    <rest:revision>
      <rest:number>1</rest:number>
      <rest:timestamp>2007-04-26T11:35:29.812Z</rest:timestamp>
      <rest:creator>demouser</rest:creator>
      <rest:label xsi:nil="true"/>
      <rest:last>1</rest:last>
    </rest:revision>
    <rest:relationships/>
    <rest:cached>0</rest:cached>
    <rest:checksum>0</rest:checksum>
  </rest:metadata>
  <rest:acl>
    <rest:owner>demouser</rest:owner>
    <rest:ace>
      <rest:principal type="group">system#everyone</rest:principal>
      <rest:permission>read</rest:permission>
    </rest:ace>
    <rest:effectivePermissions>
      <rest:permission>read</rest:permission>
      <rest:permission>write</rest:permission>
    </rest:effectivePermissions>
  </rest:acl>
  <rest:data representation="xmldata">
    <rest:xmlData contentType="text/xml; charset=utf-8"
                  xml:base="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d">
      <a/>
    </rest:xmlData>
  </rest:data>
</rest:resource>
```

Search in collection

The collection search functionality allows to choose which documents are included in a collection listing. The listing contains only documents whose descriptor satisfies the conditions expressed in the request.

The search does not work for documents without a descriptor. A condition containing a property which is not in descriptor's schema is never satisfied.

The condition consists a name of a property prefixed with `cs_`. The value is a string. If the value is prefixed with an asterisk `*`, it is interpreted as case insensitive substring match. Otherwise it is case sensitive equals. For properties with multiple value, one matching value is enough to satisfy the condition.

The query can contain more conditions. More conditions for one property form a set of alternatives (logical OR). Finally all remaining conditions and alternatives are joined with an AND logical operation.

It is possible to set order of the documents using parameters starting with the name of the property prefixed with `co_`. For example `co_name=1,asc` or equivalently `co_name=asc`. The number is precedence of the ordering. The property cannot be of some type with (potentially) multiple values.

Example 9: All WSDLS containing substring `hello` in its name

```
GET http://localhost:8080/soa/systinet/platform/rest/repository/wsdls/?cs_name=*hello
```

Example 10: All person artifacts ordered by name

```
GET http://localhost:8080/soa/systinet/platform/rest/repository/
contactArtifacts/?data&cs_artifactType=urn:com:systinet:soa:model:artifacts:content:contact:person&co_name=asc
```

Search options can be combined with `meta`, `desc` and `rss`.

EXIST

The EXIST operation is used to check existence of a resource.

It is bound to HTTP GET or HEAD. The response does not contain an HTTP body, it contains only HTTP headers. The most important part of the response is the HTTP status code: 200 – resource exists, 404 – resource does not exist.

Example 11: Check the existence of resource `/c/a`

```
GET http://localhost:8080/soa/systinet/platform/restBasic/repository/c/a?exist
```

Example 12: Response to Example 11 on page 38

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.4; JBoss-4.0.5.GA (build: CVSTag=Branch_4_0 date=200610162339)/Tomcat-5.5
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 0
Date: Thu, 26 Apr 2007 12:17:22 GMT
```

Example 13: Check the existence of resource `/c/a`

```
HEAD http://localhost:8080/soa/systinet/platform/restBasic/repository/c/a
```

Example 14: Response to Example 13 on page 39

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.4; JBoss-4.0.5.GA (build: CVSTag=Branch_4_0 date=200610162339)/Tomcat-5.5
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 0
Date: Thu, 26 Apr 2007 12:17:22 GMT
```

UPDATE

Similar operation to CREATE. Creates a new revision of the resource.

As for CREATE, it is not possible to specify the content of a collection.

Example 15: Update a document

```
POST http://localhost:8080/soa/systinet/platform/restBasic/repository/c/?resource=d&update
```

You cannot simply enter the URL into a browser either, since this will result in a GET request.

DELETE

A successful DELETE results in a response containing resource metadata, which shows that it has been deleted:

Example 16: Delete a document and return the XML serialization

```
POST http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete
```

Note: You cannot simply enter the URL into a browser, since this will result in a GET request.

Example 17: Response to Example 16 on page 40

```
<?xml version="1.0" encoding="UTF-8"?>
<rest:resource xlink:href="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d"
    xml:base="http://localhost:8080/soa/systinet/platform/restBasic/repository/"
 type="document"
 name="d"
 requestURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete="
 readURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?meta&amp;desc&amp;data"

 revisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?revision=1"
 updateURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update"

updateRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update&amp;revision=1"

 deleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete"

deleteRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete&amp;revision=1"

 undeleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?undelete"
 purgeURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge"

purgeRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge&amp;revision=1"

 viewURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view"
 viewRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view&amp;revision=1"

 aclURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?acl"
 revision="1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:a="http://systinet.com/2005/05/soa/model/artifact"
 xmlns:r="http://systinet.com/2005/05/repository"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:pt="http://systinet.com/2005/05/soa/model/property/type"
 xmlns:rest="http://systinet.com/2005/05/soa/resource"
 xmlns:p="http://systinet.com/2005/05/soa/model/property"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:g="http://systinet.com/2005/05/soa/model/propertyGroup">
  <rest:metadata>
    <rest:path>c/d</rest:path>
    <rest:collection>c/</rest:collection>
    <rest:binary>0</rest:binary>
    <rest:contentType>text/xml; charset=utf-8</rest:contentType>
    <rest:type>document</rest:type>
    <rest:deleted>1</rest:deleted>
```

```
    <rest:owner>demouser</rest:owner>
    <rest:revision>
      <rest:number>1</rest:number>
      <rest:timestamp>2007-04-26T12:41:19.906Z</rest:timestamp>
      <rest:creator>demouser</rest:creator>
      <rest:label xsi:nil="true"/>
     <rest:last>1</rest:last>
    </rest:revision>
    <rest:relationships/>
    <rest:cached>0</rest:cached>
    <rest:checksum>0</rest:checksum>
  </rest:metadata>
</rest:resource>
```

UNDELETE

A successful UNDELETE results in a response containing resource metadata, which shows that it has been undeleted:

Example 18: Undelete a document and return the XML serialization

```
POST http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?undelete
```

Note: You cannot simply enter the URL into a browser, since this will result in a GET request.

Example 19: Response to Example 18 on page 42

```
<?xml version="1.0" encoding="UTF-8"?>
<rest:resource xlink:href="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d"
    xml:base="http://localhost:8080/soa/systinet/platform/restBasic/repository/"
 type="document"
 name="d"
 requestURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?undelete="
 readURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?meta&amp;desc&amp;data"

 revisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?revision=1"
 updateURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update"

updateRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update&amp;revision=1"

 deleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete"

deleteRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete&amp;revision=1"

 undeleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?undelete"
 purgeURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge"

purgeRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge&amp;revision=1"

 viewURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view"
 viewRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view&amp;revision=1"

 aclURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?acl"
 revision="1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:a="http://systinet.com/2005/05/soa/model/artifact"
 xmlns:r="http://systinet.com/2005/05/repository"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:pt="http://systinet.com/2005/05/soa/model/property/type"
 xmlns:rest="http://systinet.com/2005/05/soa/resource"
 xmlns:p="http://systinet.com/2005/05/soa/model/property"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:g="http://systinet.com/2005/05/soa/model/propertyGroup">
  <rest:metadata>
    <rest:path>c/d</rest:path>
    <rest:collection>c/</rest:collection>
    <rest:binary>0</rest:binary>
    <rest:contentType>text/xml; charset=utf-8</rest:contentType>
    <rest:type>document</rest:type>
    <rest:deleted>0</rest:deleted>
```

```
    <rest:owner>demouser</rest:owner>
    <rest:revision>
      <rest:number>1</rest:number>
      <rest:timestamp>2007-04-26T12:41:19.906Z</rest:timestamp>
      <rest:creator>demouser</rest:creator>
      <rest:label xsi:nil="true"/>
      <rest:last>1</rest:last>
    </rest:revision>
    <rest:relationships/>
    <rest:cached>0</rest:cached>
    <rest:checksum>0</rest:checksum>
  </rest:metadata>
</rest:resource>
```

PURGE

A successful PURGE results in a response containing resource metadata, which shows that it has been purged:

Example 20: Delete a document and return the XML serialization

```
POST http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge
```

Note: You cannot simply enter the URL into a browser, since this will result in a GET request.

Example 21: Response to Example 20 on page 44

```
<?xml version="1.0" encoding="UTF-8"?>
<rest:resource xlink:href="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d"
    xml:base="http://localhost:8080/soa/systinet/platform/restBasic/repository/"
 type="document"
 name="d"
 requestURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge="
 readURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?meta&amp;desc&amp;data"

 revisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?revision=1"
 updateURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update"

updateRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?update&amp;revision=1"

 deleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete"

deleteRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?delete&amp;revision=1"

 undeleteURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?undelete"
 purgeURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge"

purgeRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?purge&amp;revision=1"

 viewURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view"
 viewRevisionURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?view&amp;revision=1"

 aclURI="http://localhost:8080/soa/systinet/platform/restBasic/repository/c/d?acl"
 revision="1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:a="http://systinet.com/2005/05/soa/model/artifact"
 xmlns:r="http://systinet.com/2005/05/repository"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:pt="http://systinet.com/2005/05/soa/model/property/type"
 xmlns:rest="http://systinet.com/2005/05/soa/resource"
 xmlns:p="http://systinet.com/2005/05/soa/model/property"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:g="http://systinet.com/2005/05/soa/model/propertyGroup">
  <rest:metadata>
    <rest:path>c/d</rest:path>
    <rest:collection>c/</rest:collection>
    <rest:binary>0</rest:binary>
    <rest:contentType>text/xml;  charset=utf-8</rest:contentType>
    <rest:type>document</rest:type>
    <rest:deleted>1</rest:deleted>
```

```
    <rest:owner>demouser</rest:owner>
    <rest:revision>
      <rest:number>1</rest:number>
      <rest:timestamp>2007-04-26T12:41:19.906Z</rest:timestamp>
      <rest:creator>demouser</rest:creator>
      <rest:label  xsi:nil="true"/>
      <rest:last>1</rest:last>
    </rest:revision>
    <rest:relationships/>
    <rest:cached>0</rest:cached>
    <rest:checksum>0</rest:checksum>
  </rest:metadata>
</rest:resource>
```

REST Exceptions

Exceptions that result from a REST operation are represented in the HTTP response in XML.

Example 22: Error Response

```
<?xml version="1.0" encoding="UTF-8"?>
<rest:exception xml:base="http://localhost:8080/soa/systinet/platform/restBasic/repository/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:a="http://systinet.com/2005/05/soa/model/artifact"
   xmlns:r="http://systinet.com/2005/05/repository"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:pt="http://systinet.com/2005/05/soa/model/property/type"
   xmlns:rest="http://systinet.com/2005/05/soa/resource"
   xmlns:p="http://systinet.com/2005/05/soa/model/property"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:g="http://systinet.com/2005/05/soa/model/propertyGroup">
  <rest:code>r:document-not-found</rest:code>
  <rest:message>{http://systinet.com/2005/05/repository}document-not-found:
     REST request processing failed. Method: GET, URL:
http://localhost:8080/soa/systinet/platform/restBasic/repository/absent.xml
  </rest:message>
  <rest:stackTrace>com.systinet.platform.RepositoryException:
{http://systinet.com/2005/05/repository}document-not-found:
 REST request processing failed. Method: GET, URL:
http://localhost:8080/soa/systinet/platform/restBasic/repository/absent.xml

  at com.systinet.platform.rest.service.RestService.process(RestService.java:236)
  at com.systinet.platform.servlet.processing.RawServiceClassWrappingServlet.
   genericDo(RawServiceClassWrappingServlet.java:139)
  at com.systinet.platform.servlet.processing.RawServiceClassWrappingServlet.
   doGet(RawServiceClassWrappingServlet.java:126)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:697)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:810)
  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:252)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
  at com.systinet.platform.servlet.processing.security.HttpBasicFilter.doFilter(HttpBasicFilter.java:116)

  at
com.systinet.platform.servlet.processing.security.AbstractSecurityFilter.doFilter(AbstractSecurityFilter.java:72)

  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:202)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
  at com.systinet.platform.servlet.processing.security.SiteminderFilter.doFilter(SiteminderFilter.java:95)

  at
com.systinet.platform.servlet.processing.security.AbstractSecurityFilter.doFilter(AbstractSecurityFilter.java:72)

  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:202)
```

```
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
  at
com.systinet.platform.servlet.processing.security.InitSecurityFilter.doFilter(InitSecurityFilter.java:30)

  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:202)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
  at org.jboss.web.tomcat.filters.ReplyHeaderFilter.doFilter(ReplyHeaderFilter.java:96)
  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:202)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
  at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:213)
  at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:178)
  at org.jboss.web.tomcat.security.SecurityAssociationValve.invoke(SecurityAssociationValve.java:175)
  at org.jboss.web.tomcat.security.JaccContextValve.invoke(JaccContextValve.java:74)
  at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:126)
  at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:105)
  at org.jboss.web.tomcat.tc5.jca.CachedConnectionValve.invoke(CachedConnectionValve.java:156)
  at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:107)
  at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:148)
  at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:869)
  at
org.apache.coyote.http11.Http11BaseProtocol$Http11ConnectionHandler.processConnection(Http11BaseProtocol.java:664)

  at org.apache.tomcat.util.net.PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:527)
  at org.apache.tomcat.util.net.MasterSlaveWorkerThread.run(MasterSlaveWorkerThread.java:112)
  at java.lang.Thread.run(Thread.java:595)
 Caused by: com.systinet.platform.RepositoryException: {http://systinet.com/2005/05/repository}document-
not-found:
 The document absent.xml was not found.

  at com.systinet.platform.rdbms.runtime.impl.Resource.invokeGetResourceInternal(Resource.java:1383)
  at com.systinet.platform.rdbms.runtime.impl.Resource.getResourceInternal(Resource.java:1327)
  at com.systinet.platform.rdbms.runtime.impl.Resource.getResourceByPath(Resource.java:323)
  at com.systinet.platform.rdbms.runtime.xmldbadapter.DbSessionImpl.doGetResource(DbSessionImpl.java:109)

  at com.systinet.platform.xmldb.DbSession.getResource(DbSession.java:271)
  at com.systinet.platform.impl.SessionImpl.getDocument(SessionImpl.java:426)
  at com.systinet.platform.rest.service.RestHelper.getDocumentResource(RestHelper.java:620)
  at com.systinet.platform.rest.service.RestGETProcessing.processGetRaw(RestGETProcessing.java:751)
  at
com.systinet.platform.rest.service.RestGETProcessing.getProcessSetHttpOkNoCache(RestGETProcessing.java:206)

  at com.systinet.platform.rest.service.RestGETProcessing.processGetPreparsed(RestGETProcessing.java:147)

  at com.systinet.platform.rest.service.RestGETProcessing.processGet(RestGETProcessing.java:117)
  at com.systinet.platform.rest.service.RestService.executeOperation(RestService.java:279)
  at com.systinet.platform.rest.service.RestService.process(RestService.java:226)
  ... 34 more
```

```
    </rest:stackTrace>
</rest:exception>
```

The `Content-type` of the response is `text/xml`. The interpretation of HTTP response codes for different operations is summarized in the following table.

Table 6. Possible HTTP Response Codes

Code	CREATE	GET	UPDATE	DELETE	Meaning
400 Bad Request	yes	yes	yes	yes	Bad request if: • the request REST operation is invalid • serialization is erroneous • mime type is not supported • resource is not supported
401 Unauthorized	yes	yes	yes	yes	Authentication failure. Credentails are required or were invalid. Serialization of the exception is not provided because the Java object is not available.
403 Forbidden	yes	yes	yes	yes	The current user does not have the right to perform the requested action.
404 Not Found	yes	yes	yes	yes	The resource does not exist – the containing collection in the case of CREATE.
409 Conflict	yes	no	yes	yes	Conflict, if the resource already exists for create or concurrent modification for update and delete.

Executable Objects

In SOA Systinet there are two kinds of objects that can be executed: functinal resources and some artifacts, for example task artifacts. What these objects have in common is that they are basically a description of a function which builds its result based on (some part of) data stored on the repository.

Functional Resources

A functional resource is a piece of code that can handle a request and somehow provide a response. In SOA Systinet they are accessible though the REST interface with URIs that contains `service` namespace (see Table 1 on page 17). For example:

```
http://localhost:8080/soa/systinet/platform/rest/service/system/product-information
```

It is possible to map more resources to the same collection (e.g. `system/`). However it is not possible to map one functional resource under another one (for example it would be illegal to add a resource mapped to `system/product-information/jvm-information`; the space already belongs to the product information functional resource). A functional resource (or its programmer) can choose which HTTP operations to support and is reponsible for handling the request parameters.

The functional resource are meant to replace XQueries which in the previous versions server for the same purpose. For example the WSIL functionality is now implemented by the functional resource available at:

```
http://localhost:8080/soa/systinet/platform/rest/service/system/wsil
```

However in the previous version the XQuery providing the WSIL document was located elsewere. This problem is addressed by introduction of aliases: documents with a special content type that are located in the `repository` namespace. An execute request for the alias if forwarded to the associated functional resource. For the example the following request returns the same document as the previous one:

```
GET http://localhost:8080/soa/systinet/platform/rest/repository/queries/wsil.xqy?execute
```

Some more details about aliases are documented in the next section.

Executable Artifacts

Several kinds of repository documents can be executed: task artifacts, stored search artifacts and aliases. A document is executed by a request containing `execute` parameter.

Note: The parameter `execute` is also used combined with the `fulltext` parameter to run full text search through a given collection (and its subcollections).

The result of an execution of a task artifact is the result report document.

The representation of the result can be modified by request parameters `data` or `rss`.

The last kind executable documents are aliases. They are technically speaking not artifact but they appear in a collection for artifacts. The data they contain is a reference to associated functional resource:

```
<?xml version="1.0" encoding="UTF-8"?>
<rest:resource xlink:href="http://localhost:8080/soa/systinet/platform/rest/repository/queries/wsil.xqy"

    xml:base="http://localhost:8080/soa/systinet/platform/rest/repository/"
 type="document"
 name="wsil.xqy"
 ...
 >
  <rest:metadata>
    <rest:path>queries/wsil.xqy</rest:path>
    <rest:collection>queries/</rest:collection>
    <rest:binary>0</rest:binary>
    <rest:contentType>x-application/alias</rest:contentType>
    <rest:type>document</rest:type>
    <rest:deleted>0</rest:deleted>
    <rest:owner>systinet:admin</rest:owner>
    <rest:revision>
      <rest:number>1</rest:number>
      <rest:timestamp>2007-04-26T09:42:31.578Z</rest:timestamp>
      <rest:creator>systinet:admin</rest:creator>
      <rest:label  xsi:nil="true"/>
      <rest:last>1</rest:last>
    </rest:revision>
    <rest:relationships/>
    <rest:cached>0</rest:cached>
    <rest:checksum>0</rest:checksum>
    <rest:extensions>
      <r:alias xmlns:r="http://systinet.com/2005/05/repository">/system/wsil</r:alias>
    </rest:extensions>
  </rest:metadata>
  <rest:descriptor/>
  <rest:data representation="xmldata">
    <rest:xmlData contentType="x-application/alias"
                xml:base="http://localhost:8080/soa/systinet/platform/rest/repository/queries/wsil.xqy"/>

  </rest:data>
</rest:resource>
```

3 REST Client

The Java REST HTTP client hides the technical details of the REST protocol.

It is composed of the following base packages and classes:

`org.systinet.platform.rest`

A package containing the foundation of the client that is used through its implementation, mainly:

`org.systinet.platform.rest.Client`

REST Client implementation.

`org.systinet.platform.rest.Source`

Hides REST request data format complexity.

`org.systinet.platform.rest.ClientException`

Client exception thrown by the client.

`org.systinet.platform.rest.RestHelper`

Constants and helper methods.

`org.systinet.platform.rest.schema.model.xsd`

A package containing object representation classes of a REST resource.

`org.systinet.platform.rest.schema.model.xsd.Resource`

The root class of the REST resource serialization.

For more details please see the Javadoc located in `PLATFORM_HOME/doc/api`.

Basic Principles

This section will show you how to interact with SOA Systinet using the REST HTTP client.

For a REST GET use the following steps:

1. GET credentials.

It is possible to omit this step and access the server without any credentials - using anonymous access. However obviously only publicly visible documents can be accessed in this way.

The other option is to use HTTP Basic authentication:

```
Credentials credentials = SecurityHelper.createCredentials("demouser", "changeit",
SecurityHelper.HttpBasic);
```

2 Construct the resource URL. For HTTP Basic authentication the base URL follow this example:

```
public static String RESTBaseHttpBasicUrl =
"http://localhost:8080/soa/systinet/platform/restBasic/repository/";
```

For single sign-on it resembles the following URL:

```
public static String RESTBaseAnonymousUrl =
"http://localhost:8080/soa/systinet/platform/rest/repository/";
```

A resource URL is composed of a server base URL, collection name and the resource name:

```
String resourceUrl = RESTBaseHttpBasicUrl + "businessServiceArtifacts/test";
```

3 GET resource by invoking the GET method on the REST client. In this example we GET the REST resource representation with sections data, meta, descriptor and ACL:

```
Resource resource = Client.get(resourceUrl, AccessMode.DATA_META_DESC_ACL, credentials);
```

4 Output the result:

```
System.out.println(resource.getMetadata().getPath());
```

The complete code fragment (HTTP Basic) is shown below:

```
// Copyright 2001-2007 Systinet Corp. All rights reserved.
// Use is subject to license terms.

package example;
```

```
import org.idoox.security.Credentials;
import org.idoox.wasp.SecurityHelper;
import org.systinet.platform.rest.AccessMode;
import org.systinet.platform.rest.Client;
import org.systinet.platform.rest.Source;
import org.systinet.platform.rest.schema.model.xsd.Resource;

public class RESTExample {
 public static String RESTBaseHttpBasicUrl =
  "http://localhost:8080/soa/systinet/platform/restBasic/repository/";

 public static void main(String[] args) throws Exception {
  Credentials credentials =
   SecurityHelper.createCredentials("demouser", "changeit", SecurityHelper.HttpBasic);

  String resourceUrl = RESTBaseHttpBasicUrl + "businessServiceArtifacts/test";

  Resource resource = Client.get(resourceUrl, AccessMode.DATA_META_DESC_ACL, credentials);
  System.out.println(resource.getMetadata().getPath());
 }
}
```

To REST CREATE, follow these steps:

1 GET credentials in the same way as described in the inquiry case.

2 CREATE resource parts (in this example we provide data only):

```
Source data = new Source("example text content");
```

3 Use a secure endpoint to publish the artifact:

```
Resource resource = Client.createDocument(rootCollectionUrl, "test", data, credentials);
```

The complete code fragment (HTTP Basic) is shown below:

```
// Copyright 2001-2007 Systinet Corp. All rights reserved.
// Use is subject to license terms.

package example;

import org.idoox.security.Credentials;
```

```
import org.idoox.wasp.SecurityHelper;
import org.systinet.platform.rest.AccessMode;
import org.systinet.platform.rest.Client;
import org.systinet.platform.rest.Source;
import org.systinet.platform.rest.schema.model.xsd.Resource;

public class RESTExample {
 public static String RESTBaseHttpBasicUrl =
  "http://localhost:8080/soa/systinet/platform/restBasic/repository/";

 public static void main(String[] args) throws Exception {
  Credentials credentials =
   SecurityHelper.createCredentials("demouser", "changeit", SecurityHelper.HttpBasic);

  String rootCollectionUrl = RESTBaseHttpBasicUrl;

  Source data = new Source("example text content");
  Resource resource = Client.createDocument(rootCollectionUrl, "test", data, credentials);

  System.out.println(resource.getMetadata().getPath());
 }
}
```

For more details, please see the HP SOA Systinet Demo Guide and REST Interface on page 15.

REST Client Package

This section describes how to use the client distribution. This client allows you to access SOA Systinet through a REST HTTP interface.

The installation program creates the client distribution in subdirectory `client` of the directory in which SOA Systinet is installed. Futher in this section, the system property `CLIENT_HOME` refers to this directory.

Note: The `CLIENT_HOME` directory contains all required files and can be copied to a different location of your choice.

The `CLIENT_HOME` contains three subdirectories:

- `bin` - shell scripts for running the REST tools. Not necessary unless you want to use these tools.

- `conf` - files with client's configuration
- `lib` - jar files that compose the client

Note: Directory `conf/sdm/` including its content and file `lib/platform_sdm.jar` are not necessary for the REST client. They are used only in the SDM client.

Note: If you want to use an HTTPS connection to SOA Systinet server, you must import the server's certificate into the truststore using standard Java `keytool` command. The recommended location and name is `CLIENT_HOME/conf/client.truststore`.

Tip: You do not have to place client files to directories that have specific names. For example, all client files can be copied to the flat directory.

Client Classpath

For each Java program using the REST client, the associated `.jar` files must be added to the classpath. The classpath must contain all the `.jar` files located in the `lib` directory of the client distribution (except the `platform_sdm.jar`):

```
account_client.jar
activation.jar
builtin_serialization.jar
commons-logging-api.jar
commons-logging.jar
core_services_client.jar
core_tools.jar
group_client.jar
jaxm.jar
jaxrpc.jar
log4j.jar
platform-core.jar
runner.jar
saaj.jar
security-ng.jar
security2-ng.jar
security_providers.jar
uddiclient_api_v3.jar
```

```
uddiclient_core.jar
wasp.jar
xercesImpl.jar
xml-apis.jar
```

Client Environment

To run your HP SOA Systinet Registry client code you must set the following Java properties:

```
-Dwasp.location=CLIENT_HOME
-Dwasp.config.location=conf/clientconf.xml
-Djava.security.auth.login.config=CLIENT_HOME/conf/jaas.config
-Didoox.debug.level=1
-Didoox.debug.logger=log4j
-Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog
```

If the client will used HTTPS transport, set additionally this property:

```
-Djavax.net.ssl.trustStore=CLIENT_HOME/conf/client.truststore
```

Note: Replace `CLIENT_HOME` with an appropriate directory name or variable.

4 SDM Client

The SDM client provides a high-level abstraction of artifact descriptors and relationships. It is built on top of the REST Client on page 53 and provides methods to speed up modeling and interaction with SOA Systinet.

It is composed of the following base packages and classes:

org.systinet.platform.sdm

A package containing the foundation of the client that is used through its implementation, mainly:

org.systinet.platform.sdm.SdmClientConstants

Constants used to reference particular repository collections where artifacts are stored and taxonomy URN.

org.systinet.platform.sdm.SdmClientHelper

Helper class providing useful methods for artifacts manipulation.

org.systinet.platform.sdm.SdmClient

Core of the generic client allowed to work with any artifact type. On top of this core is a specific adaptor for each artifact. This client is not typically used directly.

org.systinet.platform.sdm.xsd.artifact

A package containing implementation classes for each artifact type. Each artifact type class contains CRUD methods allowing the creation of artifact instances and their manipulation.

org.systinet.platform.sdm.xsd.artifact.Artifact

Superclass of all artifact type classes.

org.systinet.platform.sdm.xsd.property

Implementation of all artifact properties.

For more details please see the Javadoc located in `PLATFORM_HOME/doc/api`.

Basic Principles

This section shows how to interact with SOA Systinet using the SDM Client.

For an inquiry pattern use the following steps:

1 Get credentials.

Unless you want to use anonymous access, it is necessary to create HTTP Basic credentials:

```
Credentials credentials
        =SecurityHelper.createCredentials("demouser", "changeit", SecurityHelper.HttpBasic);
```

2 Construct the artifact URL. For SSL transport HTTP Basic authentication the base URL follows this example:

```
static final String restEndpoint="https://localhost:8843/soa/systinet/platform/restBasic/repository/";
```

In the case of anonymous access use the following URL:

```
static final String restEndpoint="https://localhost:8843/soa/systinet/platform/rest/repository/";
```

An artifact URL is comprised of a server base URL, collection name and the artifact name:

```
String webServiceUrl
 = restEndpoint+"/"+SdmClientConstants.COLLECTION_NAME_WEBSERVICE+"/"+artifactName;
```

3 Get the artifact by invoking the static method on the artifact type class:

```
WebServiceArtifact webServiceArtifact
        = WebServiceArtifact.get(webServiceUrl, credentials);
```

4 Output the result:

```
SdmClientHelper.showArtifact(webServiceArtifact);
```

The complete code fragment (HTTP Basic) is shown below:

```
// Copyright 2001-2007 Systinet Corp. All rights reserved.
// Use is subject to license terms.

package example;

import org.idoox.security.Credentials;
import org.idoox.wasp.SecurityHelper;
import org.systinet.platform.sdm.SdmClientConstants;
import org.systinet.platform.sdm.SdmClientHelper;
import org.systinet.platform.sdm.xsd.artifact.WebServiceArtifact;

public class InquiryExample {
 static final String restEndpoint="https://localhost:8843/soa/systinet/platform/restBasic/repository/";

 static final String artifactName="MyWebServiceArtifact";

 public static void main(String[] args) throws Exception {
  Credentials credentials
   =SecurityHelper.createCredentials("demouser", "changeit", SecurityHelper.HttpBasic);

  String webServiceUrl
   = restEndpoint + "/" + SdmClientConstants.COLLECTION_NAME_WEBSERVICE + "/" + artifactName;

  WebServiceArtifact webServiceArtifact
   = WebServiceArtifact.get(webServiceUrl, credentials);

  SdmClientHelper.showArtifact(webServiceArtifact);
 }
}
```

To publish, follow these steps:

1. Get credentials in the same way as described in the inquiry case.
2. Build artifact:

```
WebServiceArtifact artifact = new WebServiceArtifact();

artifact.setNameGroup(new NameGroup(new Name[] {
  new Name("en","FTP Web Service")
 }));

artifact.setDescriptionGroup(new DescriptionGroup(
```

```
 new Description[] {
  new Description("en",
   "Web Service artifact representing a Web Service interface to the FTP protocol.")
 }));


AccessPoint accessPoint = new AccessPoint("http://soap.systinet.net:9080/FTPService");
accessPoint.setUseType("Unsecured endpoint");
artifact.setAccessPointGroup(new AccessPointGroup(new AccessPoint[]{
  accessPoint
 }));

// production stage
artifact.setProductionStage(
 new ProductionStage(
  "Production",
  SdmClientConstants.TAXONOMY_LIFECYCLE_STAGES,
  "uddi:systinet.com:soa:model:taxonomies:lifecycleStages:production"));
```

3 Use a secure endpoint to publish the artifact:

```
String webServiceUrl
        = WebServiceArtifact.create(
        restEndpoint,
        artifactName,
        buildWebServiceArtifact(),
        null,null,credentials);
```

The complete code fragment (HTTP Basic) is shown below:

```
// Copyright 2001-2007 Systinet Corp. All rights reserved.
// Use is subject to license terms.
package example;

import org.idoox.security.Credentials;
import org.idoox.wasp.SecurityHelper;
import org.systinet.platform.sdm.SdmClientConstants;
import org.systinet.platform.sdm.xsd.artifact.WebServiceArtifact;
import org.systinet.platform.sdm.xsd.group.AccessPointGroup;
import org.systinet.platform.sdm.xsd.group.DescriptionGroup;
import org.systinet.platform.sdm.xsd.group.NameGroup;
import org.systinet.platform.sdm.xsd.property.AccessPoint;
import org.systinet.platform.sdm.xsd.property.Description;
import org.systinet.platform.sdm.xsd.property.Name;
```

```
import org.systinet.platform.sdm.xsd.property.ProductionStage;

public class PublicationExample {
 static final String restEndpoint="https://localhost:8843/soa/systinet/platform/restBasic/repository/";

 static final String artifactName="MyWebServiceArtifact";

 public static void main(String[] args) throws Exception {
  Credentials credentials
   =SecurityHelper.createCredentials("demouser", "changeit", SecurityHelper.HttpBasic);

  String webServiceUrl = WebServiceArtifact.create(
        restEndpoint,
        artifactName,
        buildWebServiceArtifact(),
        null, null, credentials);

  System.out.println("Creates Web Service artifact: "+webServiceUrl);
 }

 private static WebServiceArtifact buildWebServiceArtifact() {
  WebServiceArtifact artifact =new WebServiceArtifact();

  artifact.setNameGroup(new NameGroup(new Name[] {
    new Name("en","FTP Web Service")
   }));

  artifact.setDescriptionGroup(new DescriptionGroup(
    new Description[] {
     new Description("en","Web Service artifact representing FTP Web Service.")
   }));


  AccessPoint accessPoint = new AccessPoint("http://soap.systinet.net:9080/FTPService");
  accessPoint.setUseType("Unsecured endpoint");
  artifact.setAccessPointGroup(new AccessPointGroup(new AccessPoint[]{
    accessPoint
   }));

  // production stage
  artifact.setProductionStage(
   new ProductionStage(
    "Production",
    SdmClientConstants.TAXONOMY_LIFECYCLE_STAGES,
    "uddi:systinet.com:soa:model:taxonomies:lifecycleStages:production"));

  return artifact;
```

```
  }
 }
```

For more details, please see the HP Systinet Demo Guide and REST Interface on page 15.

SDM Client Package

The SDM client is the same as the REST client described in REST Client Package on page 56.

Warning: Do not remove `conf/sdm` and include `lib/platform_sdm.jar` in the classpath.

5 Technical Security

This chapter provides a technical description of SOA Systinet security. It consists of the following sections:

SOA Systinet Overview

SOA Systinet has four components. From the security point of view, their functions are as follows:

- SSO service
  - User and group management.
  - Authentication service.

- SOA Systinet Platform
  - Provides data store for artifacts.
  - Exposes WEB and REST services to manage artifacts.
- Policy Manager
  - Provides UI for policy management.
  - Engine for policy validation.
  - Exposes WEB and REST services to policy management and validation.
- Reporting Service
  - Stores report definitions and data.
  - Engine for report generation.
  - Exposes REST service for report management.

Users and Groups

SOA Systinet uses the userstore provided by SSO. This userstore is reused from HP SOA Systinet Registry. It offers two backends:

- Database
  - Users and groups are stored in database.
  - Authentication is done by comparing password MD5 hash.
  - Only MD5 hash of user password is stored in database
- LDAP — Authentication is left to LDAP.

When accessing SOA Systinet, the user may have the following security attributes assigned:

- Authenticated — See Authentication on page 67 for authentication mechanisms.
- Anonymous — A user who does not pass any credentials and accesses service on access points with an anonymous authentication mechanism. His/her name used in ACL is `systinet#anonymous`.
- Resource owner — A user who owns the accessing resource. Used in ACL evaluation.
- Administrator — A user who is marked as an administrator. He has the rights to perform all actions (no ACLs are applied on resources, management tasks, etc). In the default installation, there is one user named "admin" who is an administrator. SOA Systinet Platform's user and group management can be used to mark a user as administrator and/or to define a group of administrators (in other words, make every member of a group an administrator).
- System administrator — An internal identity used in 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. Its name used in ACL is `systinet:admin`.

We use the following built-in groups:

- `system#registered` — all users who exists in the userstore. In other words, users who are authenticated.
- `system#everyone` — Both authenticated users (group `system#registered`) and the anonymous user (`systinet#anonymous`).

Transport Security

SOA 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: SOA Systinet Platform does not provide any management of SSL (certificates) because HTTPS transport is provided by the application server. For the simple configuration of the JBoss application server, we provide automatic SSL enablement (i.e., certificate generation and SSL configuration) as a part of installation.

Authentication

The services exposed in HTTP/HTTPS endpoints may be accessed using the following authentication mechanisms:

- Anonymous — No credentials are passed to the service. The anonymous identity is used for authorization to access resources.
- HTTP Basic — Consumes HTTP Basic credentials. If not passed, then challenge is returned to the client.
- SSO — Consumes SSO credentials. If not passed, processing passes to the next authentication mechanism.
- SiteMinder — Consumes SiteMinder credentials (only HTTP headers are supported). If not passed, processing passes to the next authentication mechanism.
- WEB — Used by UI only. If UI requires non-anonymous access, user is forwarded to login page.

Authorization

REST and WEB clients use their own authorization when accessing a service's resources. REST's resources are artifacts and collections, while WEB's resources are tasks, which provide access to artifacts.

REST ACLs

SOA Systinet Platform and Reporting Service provide resources in the hierarchical model accessible by REST. In this model there are collections and resources, where a collection can contain both individual resources and other collections.

SOA Systinet 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.

- An ACL is a list of ACEs, where an ACE is composed of the following:
  - User or group identification
  - Granted permission:
    - Read
      - Artifact — Permission to read any data and metadata of artifact.
      - Collection — Permission to read content and metadata of collection.

    - Write
        - Artifact — Permission to update any data and metadata of artifact.
        - Collection — Permission to create a new sub-collection and artifact, 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. In other words, a change to a collection's ACL does not change any ACLs of the collection's 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 (i.e., artifact type).
- The *resource owner* and *administrator* users always have read and write permissions.

The default ACL configuration is described in the Reference Guide. It is possible to change the default ACL configuration by exporting the configuration to an XML file, editing the configuration and then reimporting it. This procedure is described in the Administrator Guide.

WEB Security

The UI is composed of tasks mapped to URLs. Some tasks are restricted to authenticated users only (not anonymous user), so user is forced to authenticate (log in). Because the UI is composed of static tasks, this setup is part of the WEB configuration.

SSL Certificate Verification

SOA Systinet Platform allows you to disable certificate verification during installation (see the Installation Guide). After installation, you can enable or disable certificate validation by using the Setup tool or via the web UI (see the Administrator Guide). Disabling certificate validation eases deployment in a non-production environment, where it is not necessary to trust SSL certificates. Verification of SSL certificates applies to HTTPS connections that are initiated at the server side (so called back-channel) by Platform deployed in the application server.

When certificate verification is turned off:

- It is not verified that a SSL server certificate is trusted.
- it is not verified that hostnames in HTTPS URLs match their SSL server certificates.

When certificate verification is turned on:

- SSL server certificates must be trusted. The method of trusting SSL certificates depends on the application server that hosts SOA Systinet Platform.
- Hostnames are verified. The hostname in an HTTPS URL must match one of the hostnames that are present in the associated server certificate.

Important: Keep SSL certificate verification turned on for a production environment. The option to turn off certificate validation is provided only to ease communication between Platform and integrated HP SOA software during evaluation or development.

SSO Security

SSO service provides a single point of authentication and user and group management APIs. After a successful authentication, a SAML Assertion is issued for the user. This assertion is then transferred between SOA Systinet components without any further need for user authentication.

In SSO, each installed component must be a partner in an affiliation. Only partners in affiliation can transfer identity between each other. Each partner has a certificate/private key pair. They are used to generate SAML assertions and establish trust in the affiliation (for example, sign SSO messages and encrypt sensitive information in SSO messages).

During the installation of each SOA Systinet component (step **SSO partner**), the certificate and private key are created. The private key is stored in JKS (Java keystore), which is encrypted by a password given during installation. This password is stored in the SOA Systinet configuration database.

Each SSO partner must expose SSO partners' services. The SSO partners use each other's SSO partner services to transfer identity between themselves.

Platform Services

Platform provides REST and WEB services. Table 7 on page 71 shows the access points to which these services are mapped on HTTP and HTTPS transports provided by hosting application server.

Table 7. Platform Web Services

Type of service	Endpoint URLs	Authentication mechanisms
REST	`http://localhost:8080/soa/systinet/platform/rest/` `https://localhost:8443/soa/systinet/platform/rest/`	SSO, SiteMinder, Anonymous
	`http://localhost:8080/soa/systinet/platform/restBasic/` `https://localhost:8443/soa/systinet/platform/restBasic/`	HTTP Basic
WEB	`http://localhost:8080/soa/systinet/platform/web` `https://localhost:8443/soa/systinet/platform/web`	SSO, SiteMinder, WEB

The REST service does not use session management (due to nature of REST architecture). The WEB uses its own session management based on cookies and parameters passed in requests.

Reporting Server Services

Reporting provides a REST web service. The REST service is exposed at the following access points, mapped on HTTP and HTTPS transports provided by the hosting application server: `http://localhost:8080/reporting/` and `https://localhost:8443/reporting/` The rest service uses SSO, HTTP Basic and Anonymous authorization mechanisms.

Note: The REST service does not use session management due to the nature of REST architecture.

Policy Manager Services

Policy Manager provides REST and WEB services. Table 8 on page 72 shows the access points to which these services are mapped on HTTP and HTTPS transports provided by the hosting application server.

Table 8. Policy Manager Services

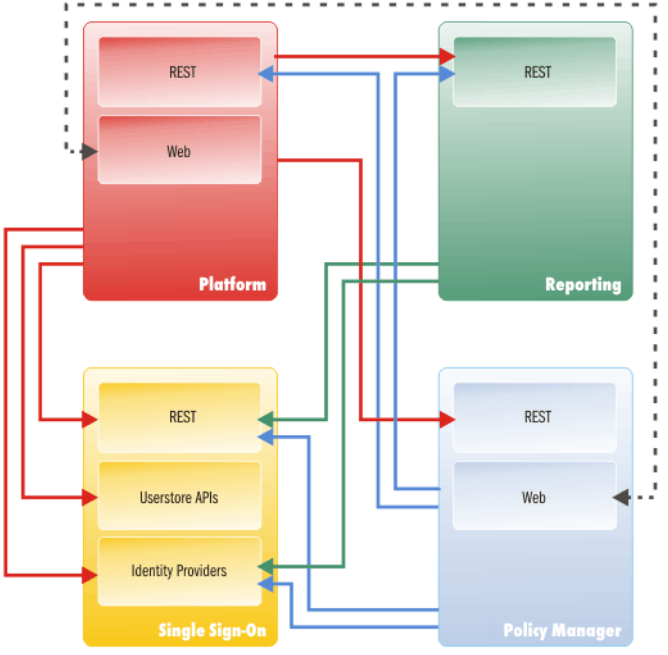
Type of service	Endpoint URLs	Authentication mechanisms
REST	`http://localhost:8080/policymgr/rest/` `https://localhost:8443/ policymgr/rest /`	SSO, HTTP-Basic
WEB	`http://localhost:8080/soa/systinet/platform/web` `https://localhost:8443/soa/systinet/platform/web`	SSO, WEB

Note: The REST service does not use session management due to the nature of REST architecture. The WEB uses the standard HTTP session mechanism.

Product Integration

Figure 1 shows the dependencies between HP SOA Systinet products. For simplicity, SSO Partner services are not shown—every partner uses every other partner's SSO Partner service. The dashed line shows the integration between Platform and Policy Manager UIs. This integration is based on SSO.

Figure 1. Product Integration and Web Services



6 RSS

RSS (Really Simple Syndication) is an XML-based system for subscribing to information sources. See http://www.rss-specifications.com/rss-specifications.htm.

SOA Systinet provides RSS feeds and can be used to subscribe to others.

This section describes what kinds of RSS Feeds are supported, where to find them and how to use the **Feed Reader** dashboard portlet.

Kinds of RSS Feed

SOA Systinet provides the following kinds of RSS feed:

document feed

contains the document revisions as separate items

collection feed

consists of recently changed documents in collections

stored search feed

the RSS representation of stored search results

Syndication Syntax

Each request for RSS must contain a URL parameter `rss`, which can be parameterized. The optional value specifies the required format of the syndicate. SOA Systinet supports the following popular syntaxes of syndicates:

- Atom v0.3 (use URL parameter `?rss=atom_0.3`)
- Atom 1.0 (use URL parameter `rss=atom_1.0`)
- RSS 0.9 (use URL parameter `rss=rss_0.9`)

- RSS 0.92 (use URL parameter `rss=rss_0.92`)
- RSS 0.93 (use URL parameter `rss=rss_0.93`)
- RSS 0.94 (use URL parameter `rss=rss_0.94`)
- RSS 1.0 (use URL parameter `rss=rss_1.0`)
- RSS 2.0 (use URL parameter `rss` or `rss=rss_2.0`), which is the default RSS format

Subscriptions over RSS

Although SOA Systinet has no abstraction of subscriptions, you can be notified of changes to repository data.

*Notification*s about new items in syndicates are a natural feature of RSS feed readers. Feed readers cache the syndicated items and inform users about new ones from the latest feed.

RSS feed readers use the item attribute `link` to recognize if the item has already been read or not. SOA Systinet's RSS feed item identifiers are based on the REST revision URL of the syndicated resource. So when the resource is created/modified, the URL of the current revision of the resource is changed. The RSS feed reader is then able to recognize that a new item has appeared in the syndication (replacing the old one) and informs the user about the changes.

The main advantage of this kind of subscription is that users need not learn any new proprietary subscription API. Users can use their favorite RSS feed readers or the one implemented as a SOA Systinet dashboard portlet.

7 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 HP SOA 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 are 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. In this step, 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 (that's 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 references 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 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.

8 Custom Validation Handlers

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

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. This not only ensures better performance, but you also do not have to modify the existing `systinet-policy.ear`. Take a look at `jboss-app.xml` for further details.
- **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 for filtering the input list of assertions–only the assertions with this namespace are passed into this handler.

Index