JasperServer Web Services Guide

Version 2.0.1

Table of Contents

1 Introduction

This document describes the JasperSoft BI suite's SOAP (Simple Object Access Protocol) web services Application Programming Interface (API). It describes the functionality provided in both Open Source and Professional editions.

You can use the web services to explore the repository, modify report units and several other resource types, and run reports with or without input parameters.

The API is simple, comprised of five methods: list, get, put, delete, and runReport.

These methods take as input an XML document (the request), and return another XML document as the result (the operation result). Because they use XML, the web services provide easy, natural integration with the most common programming languages and environments.

JasperSoft provides two complete sample applications that demonstrate the API: a simple J2EE (Java 2 Enterprise Edition) web application and the same application written in PHP (PHP Hypertext Preprocessor).

By default, the web services are available at the following URL:

Edition	URL
JasperSoft Open Source	http://localhost:8080/jasperserver/services/repository
JasperSoft Professional	http://localhost:8080/ji-pro/services/repository

where localhost is the name of the computer hosting JasperSoft and 8080 is the port you specified during installation.

Note: The context name (jasperserver or ji-pro) may also depend on the specific installation of JasperServer.

You must supply a valid account to access the. The web services accept the same accounts and credentials as the JasperSoft web interface.

The WSDL document that describes the services is generated dynamically by invoking the URL of the services with the string `?wsdl` concatenated at the end. For example:
http://localhost:8080/jasperserver/services/repository?wsdl

The complete WSDL is detailed in Appendix A "WSDL" on page 26.

To simplify life for Java developers, JasperSoft provides a set of helper classes, including a ready-to-use client that can make it easier to integrate an external application with JasperServer, be it web- or desktop-based. These classes include an object model that represents resources and creates requests and operation results, along with a Marshaller and an Unmarshaller class to quickly move between XML and the Java object model. The presentation of each service includes code samples that show how to use these classes.

1.1 ResourceDescriptor

Resources (such as reports, images, and queries) are stored in a repository, which is organized like a file system, with a root and a hierarchical set of folders. Each object in the repository is considered a resource: a folder is a resource of type folder, a JRXML resource is a resource of type file, just as images and JAR files are of type file. Some resources are more abstract, such as connection definitions and an input controls. The API methods operate on all resources, which the web services API describes using XML.

A resource is identified by:

- A name.
- A label.
- A unique Uniform Resource Identifier (URI) that defines the location of the resource in the repository. An URI is similar to a Unix path (for example, /samples/reports/AllAccounts).

A `resourceDescriptor` tag is defined by the following DTD (Document Type Definition):

```
<!ELEMENT resourceDescriptor (label, description?, resourceProperty*, resourceDescriptor*,
parameter*)>
```

```
<!ATTLIST resourceDescriptor
name CDATA #REQUIRED
wsType CDATA #REQUIRED
uriString CDATA #REQUIRED
isNew ( true | false ) false
>

<!ELEMENT resourceProperty (value?, resourceProperty*)>
<!ATTLIST resourceProperty
name CDATA #REQUIRED
>

<!ELEMENT value (#PCDATA)>

<!ELEMENT parameter (#PCDATA)>
<!ATTLIST parameter
name CDATA #REQUIRED
isListItem ( true | false ) false
>
```

The `wsType` attribute defines the nature of the resource. Currently, the possible values for this attribute are:

Value	**Description**
bean	Data source of type JavaBean
datasource	Generic data source. This type is normally used for a data source ReportUnit child resource when the is not defined locally to the ReportUnit.
dataType	Datatype (used with the input controls)
folder	Folder
font	Font file (normally a True Type font)
img	Image file
inputControl	Input control
jar	JAR file
jdbc	Data source of type JDBC
jndi	Data source of type JNDI
jrxml	JRXML source file
lov	List of values (used with the input controls)
olapMondrianCon	OLAP Mondrian connection – a direct connection to an OKAP source.
olapMondrianSchema	OLAP Mondrian Schema
olapXmlaCon	OLAP XMLA connection – a remote connection to an OLAP source.
prop	Resource bundle file (normally ending with .properties) for specific reports

Value	Description
query	Query used to retrieve data from a data source
reference	Reference to another resource. References are present only into report units
reportUnit	A complete report that can be run in JasperServer
xmlaConnection	XMLA Connection

For all the other resource types found in the repository, the web services set the attribute `wsType` to `unknown`.

The `isNew` attribute is used with the `put` service to indicate whether the resource being uploaded is new or replaces an existing resource in the repository.

A resource can have a set of *properties* depending on its type and a set of *children resources*. Finally, a resource descriptor can contain one or more *parameters*: their use is not intended to describe the resource; they store the values users select when the `runReport` service is invoked.

A `resourceProperty` is normally a simple pair of a name and a value. The Java class `com.jaspersoft.jasperserver.api.metadata.xml.domain.impl.ResourceDescriptor` contains constants for each property name. JasperSoft may add further constants in future releases.

The following resourceDescriptor sample contains a set of simple properties; it describes a JDBC connection resource:

```
<resourceDescriptor
name="JServerJdbcDS"
wsType="jdbc"
uriString="/datasources/JServerJdbcDS"
isNew="false">

<label>JServer Jdbc data source</label>

<description>JServer Jdbc data source</description>

<resourceProperty name="PROP_PARENT_FOLDER">
<value>/datasources</value>
</resourceProperty>

<resourceProperty name="PROP_VERSION">
<value>0</value>
</resourceProperty>

<resourceProperty name="PROP_DATASOURCE_DRIVER_CLASS">
<value>com.mysql.jdbc.Driver</value>
</resourceProperty>
<resourceProperty name="PROP_DATASOURCE_PASSWORD">
<value>password</value>
</resourceProperty>

<resourceProperty name="PROP_DATASOURCE_USERNAME">
<value>username</value>
</resourceProperty>

<resourceProperty name="PROP_DATASOURCE_CONNECTION_URL">
<value>jdbc:mysql://localhost/test?autoReconnect=true</value>
</resourceProperty>

</resourceDescriptor>
```

Some properties cannot be represented by a simple value. To accommodate more complicated properties, a resourceProperty can recursively contain other resourceProperties. This is the case for a List of Values resource: the values are contained in the resourceProperty named `PROP_LOV` and are represented by sub-resourceProperties. For example:

```
<resourceDescriptor name="SampleLOV" wsType="lov" uriString="/datatypes/SampleLOV" isNew="false">
<label>Sample List of Values</label>
<resourceProperty name="PROP_RESOURCE_TYPE">
<value>com.jaspersoft.jasperserver.api.metadata.common.domain.ListOfValues</value>
</resourceProperty>
<resourceProperty name="PROP_PARENT_FOLDER">
<value>/datatypes</value>
</resourceProperty>
<resourceProperty name="PROP_VERSION">
<value>-1</value>
</resourceProperty>
<resourceProperty name="PROP_HAS_DATA">
<value>false</value>
</resourceProperty>
<resourceProperty name="PROP_IS_REFERENCE">
<value>false</value>
</resourceProperty>
<resourceProperty name="PROP_LOV">
<resourceProperty name="US">
<value>United States</value>
</resourceProperty>
<resourceProperty name="CA">
<value>Canada</value>
</resourceProperty>
<resourceProperty name="IN">
<value>India</value>
</resourceProperty>
<resourceProperty name="IT">
<value>Italy</value>
</resourceProperty>
<resourceProperty name="DE">
<value>Germany</value>
</resourceProperty>
<resourceProperty name="RO">
<value>Romania</value>
</resourceProperty>
</resourceProperty>
</resourceDescriptor>
```

Notice that, for each list item, the resourceProperty name represents the item value, and the resourceProperty value contains the item label.

When a resourceDescriptor is used as an input parameter in a request document (for example, to specify a folder to list or a file to download), the description includes only a small portion of the entire resource descriptor definition: the part that describes the specific details of the resource in question

be omitted. In many cases, the only information required to identify a resource in the repository are the `wsType`, the `name`, and the `URI`.

The resource descriptor is a complex structure that transfers data regarding a specific resource between the server and the client. A request can include only one resource descriptor. Often, the request only includes a small portion of the entire resource descriptor definition: the part that describes the specific details of the resource in question.

On the contrary, the resource descriptors that the server sends are completely populated with all the data about the resources being described.

1.2 Request and Operation Result

The web services exposed by the server take a single input parameter of type string. This XML document represents the request. The following shows its DTD:

```
<!ELEMENT request (argument*, resourceDescriptor?)>
<!ATTLIST request
operationName (get | list | put | runReport ) "list"
locale #IMPLIED
>


<!ELEMENT argument (#PCDATA)>
<!ATTLIST argument
name CDATA #REQUIRED
>
```

A request is a very simple document that contains:

- The operation to execute (list, get, put, delete, or runReport).
- A set of optional arguments. Each argument is a pair of a key and a value that is used to achieve very particular results; arguments are only used rarely.
- A resource descriptor.

While the operation name is redundant, as the operation to execute is intrinsic in the invoked service, it is useful for clarity when people working with the request document.

The services act on a single resource at time. The resource that is the subject of the request is described by a resourceDescriptor.

To get error messages in a particular locale supported by the server, specify the locale code with the `locale` attribute. Locale codes are in the form <language code>[_<country>[_<variant>]. Valid examples include en, en_US, it_IT, and ro. For a list of Java-compliant locales, please refer to Sun's Java web site.

The following sample request lists the repository root:

```
<?xml version="1.0" encoding="UTF-8"?>
<request operationName="list" locale="en">
```

```
<resourceDescriptor name="" wsType="folder" uriString="/">
<label>null</label>
</resourceDescriptor>
</request>
```

Executing a service produces the operationResult.in the form of a new XML document.

The DTD is very simple:

```
<!ELEMENT operationResult (code, message?, resourceDescriptor*)>
<!ATTLIST operationResult
version NMTOKEN #REQUIRED
>


<!ELEMENT code (#PCDATA)>
<!ELEMENT message (#PCDATA)>
```

The operation result contains a return `code`, an optional return message and zero or more resource descriptors. A return code other than 0 indicates an error, which is normally described in the message tag.

The operation result always includes the `version` attribute: it can be used to detect the server version. For example, you can list the repository root and read the version set by the server in the response. In this case, we aren’t interested in the root folder’s content: we just want the version information from the response object itself.

The operation result of such a request is:

```
<operationResult version="1.2.1">
<returnCode>0</returnCode>
....
several resource descriptors....
....
</operationResult>
```

2 Services in the API

2.1 List

This service lists the contents of the specified folder or report unit. The following sample request lists the contents of the /ContentFiles folder in the repository:

```
<request operationName="list" locale="en">
<resourceDescriptor name="" wsType="folder" uriString="/ContentFiles" isNew="false">
<label>null</label>
</resourceDescriptor>
</request>
```

Sample response:

```
<operationResult version="1.2.1">
<returnCode>0</returnCode>
<resourceDescriptor name="html" wsType="folder" uriString="/ContentFiles/html" isNew="false">
<label>html</label>
<resourceProperty name="PROP_RESOURCE_TYPE">
<value>com.jaspersoft.jasperserver.api.metadata.common.domain.Folder</value>
</resourceProperty>
<resourceProperty name="PROP_PARENT_FOLDER">
<value>/ContentFiles</value>
</resourceProperty>
<resourceProperty name="PROP_VERSION">
<value>0</value>
</resourceProperty>
</resourceDescriptor>
<resourceDescriptor name="pdf" wsType="folder" uriString="/ContentFiles/pdf" isNew="false">
<label>pdf</label>
<resourceProperty name="PROP_RESOURCE_TYPE">
<value>com.jaspersoft.jasperserver.api.metadata.common.domain.Folder</value>
</resourceProperty>
<resourceProperty name="PROP_PARENT_FOLDER">
<value>/ContentFiles</value>
</resourceProperty>
<resourceProperty name="PROP_VERSION">
<value>0</value>
</resourceProperty>
</resourceDescriptor>
<resourceDescriptor name="xls" wsType="folder" uriString="/ContentFiles/xls" isNew="false">
<label>xls</label>
```

```
<resourceProperty name="PROP_RESOURCE_TYPE">

<value>com.jaspersoft.jasperserver.api.metadata.common.domain.Folder</value>

</resourceProperty>

<resourceProperty name="PROP_PARENT_FOLDER">

<value>/ContentFiles</value>

</resourceProperty>

<resourceProperty name="PROP_VERSION">

<value>0</value>

</resourceProperty>

</resourceDescriptor>

</operationResult>
```

When it lists a folder, the service returns a set of resource descriptors: one for each resource that resides in the specified folder. Use / as the URI to identify the root folder.

Similarly, when it lists a report unit, the service returns a set of resource descriptors that contain (at a minimum) the main JRXML source file. Since a resource in a report unit can be either a local resource or a reference to another repository resource, you should keep a few details in mind:

If a report unit data source is not defined locally, its `wsType` is set to `datasource`, which does not indicate the exact nature of the resource. Its type should simply be `reference`, but since the data source used by the report unit is a special child resource, it's easy to recognize. The URI of the referenced resource is available in the `PROP_REFERENCE_URI` property.

The main JRXML resource's `wsType` is always set to `jrxml`, even if it's a reference to an external JRXML resource. By looking at the `PROP_IS_REFERENCE` and `PROP_REFERENCE_URI` properties, you can determine where the resource is actually stored. The `PROP_RU_IS_MAIN_REPORT` property identifies the main JRXML source file of the report unit, even if the order of its children is altered.

The purpose of listing a report unit is to get the list of the resources contained in the report unit. To retrieve the entire report unit (report unit resource as well as its children) at the same time, use the `get` service.

You can use the list service as shortcut to get the list of data sources available in the repository. To do so, set the request's `LIST_DATASOURCES` argument `true`.

```
<request operationName="list" locale="en">

<argument name="LIST_DATASOURCES">true</argument>

</request>
```

Note: This is the only case in which a request doesn't require resource descriptor.

Java sample:

```
ResourceDescriptor rd = new ResourceDescriptor();

rd.setWsType( ResourceDescriptor.TYPE_FOLDER );

rd.setUriString("/");

List lst = wsclient.list(rd);
```

In the above sample, wsclient is an instance of `com.jaspersoft.jasperserver.irplugin.wsclient.WSClient.`

PHP sample:

```
$result = ws_list("/");

if (get_class($result) == 'SOAP_Fault')

{

$errorMessage = $result->getFault()->faultstring;

}

else

{

$folders = getResourceDescriptors($result);

}
```

This PHP sample uses the client.php file found in the PHP sample provided with JasperServer. This file defines the most important constants you may find useful when integrating with the JasperServer web services, as well as useful functions that wrap the `list`, `get` and the `runReport` web services.

2.2 Get

The get service is used to obtain information about a resource. In the case of file resources, such as images, fonts, JRXML files, and JAR files, the resource file is attached to the response message.

This method's behavior differs, depending on the type of object specified:

- Generally, a simple resource descriptor is returned.
- If you `get` a resource file, the file content is attached to the response; if you do not want the server to attach files to the response, set the request's `NO_ATTACHMENT` argument to `true`.
- If you `get` a report unit, all the related resources are added as child resource descriptors to the report unit descriptor.
- If you `get` an input control that is based on a query, and you set the `IC_GET_QUERY_DATA` argument to the valid URI of a data source, that data source is used to execute the query and populate the resource descriptor. This can be useful when the input control must be rendered (for example, on a web page) in order to capture a value to pass when executing a report.

Sample request to get a file resource:

```
<request operationName="get" locale="en">

<resourceDescriptor name="JRLogo" wsType="img" uriString="/images/JRLogo" isNew="false">

<label>JR logo</label>

<description>JR logo</description>

</resourceDescriptor>

</request>
```

The service only uses the `uriString` to identify the resource to get and check for access permissions. This means that other information present in the resource description (such as resource properties, label, and description) are not actually used or required.

If a file is attached to the response, the returned resource descriptor has the `PROP_HAS_DATA` property set to `true`. The attachments format is MIME.

A call to the get service always returns a resource descriptor. If the specified resource is not found, or the user cannot access it, an error with code 2 is returned.

Java sample:

```
String imgUri = "/images/JRLogo";

ResourceDescriptor rdis = new ResourceDescriptor();

rdis.setParentFolder("/images");

rdis.setUriString(imgUri);


ResourceDescriptor result = wsclient.get(rdis, null);
```

PHP sample:

```
$result = ws_get($someInputControlUri, array( IC_GET_QUERY_DATA => $someDatasourceUri ) );
```

The resource descriptor of an input control that includes data obtained by setting the `IC_GET_QUERY_DATA` argument to `true` would be similar to the following XML:

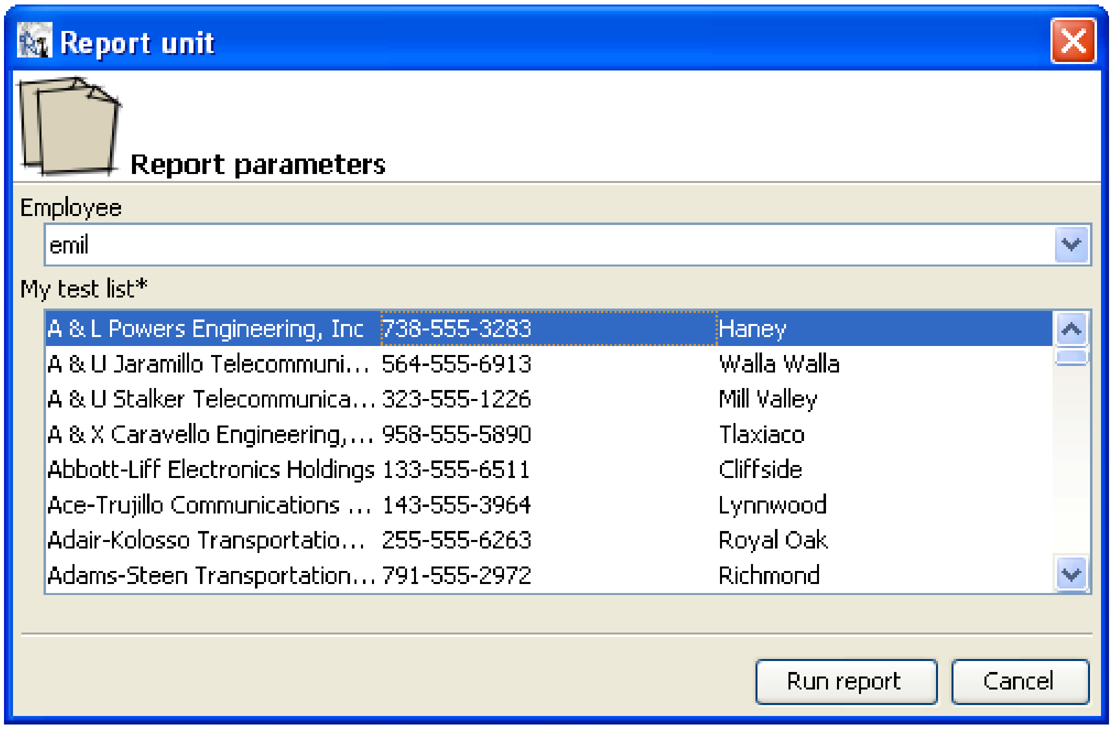
```
<resourceDescriptor name="TEST_LIST" wsType="inputControl" uriString="/MyInputControls/TEST_LIST"
isNew="false">

<label>My test list</label>

<description>My test list</description>

<resourceProperty name="PROP_RESOURCE_TYPE">

<value>com.jaspersoft.jasperserver.api.metadata.common.domain.InputControl</value>

</resourceProperty>

<resourceProperty name="PROP_PARENT_FOLDER">

<value>/MyInputControls</value>

</resourceProperty>

<resourceProperty name="PROP_VERSION">

<value>6</value>

</resourceProperty>

<resourceProperty name="PROP_HAS_DATA">

<value>false</value>

</resourceProperty>

<resourceProperty name="PROP_IS_REFERENCE">

<value>false</value>

</resourceProperty>

<resourceProperty name="PROP_INPUTCONTROL_IS_MANDATORY">

<value>true</value>

</resourceProperty>

<resourceProperty name="PROP_INPUTCONTROL_IS_READONLY">

<value>false</value>

</resourceProperty>

<resourceProperty name="PROP_INPUTCONTROL_TYPE">

<value>7</value>

</resourceProperty>

<resourceProperty name="PROP_QUERY_VALUE_COLUMN">
```

```
<value>name</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_VISIBLE_COLUMNS">
<resourceProperty name="PROP_QUERY_VISIBLE_COLUMN_NAME">
<value>name</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_VISIBLE_COLUMN_NAME">
<value>phone_office</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_VISIBLE_COLUMN_NAME">
<value>billing_address_city</value>
</resourceProperty>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA">
<resourceProperty name="PROP_QUERY_DATA_ROW">
<value>A &amp; L Powers Engineering, Inc</value>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
<value>A &amp; L Powers Engineering, Inc</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
<value>738-555-3283</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
<value>Haney</value>
</resourceProperty>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA_ROW">
<value>A &amp; U Jaramillo Telecommunications, Inc</value>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
<value>A &amp; U Jaramillo Telecommunications, Inc</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
<value>564-555-6913</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
<value>Walla Walla</value>
</resourceProperty>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA_ROW">
<value>A &amp; U Stalker Telecommunications, Inc</value>
```

```
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">

<value>A &amp; U Stalker Telecommunications, Inc</value>

</resourceProperty>

<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">

<value>323-555-1226</value>

</resourceProperty>

<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">

<value>Mill Valley</value>

</resourceProperty>

</resourceProperty>

<resourceProperty name="PROP_QUERY_DATA_ROW">

<value>A &amp; X Caravello Engineering, Inc</value>

<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">

<value>A &amp; X Caravello Engineering, Inc</value>

</resourceProperty>

<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">

<value>958-555-5890</value>

</resourceProperty>

<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">

<value>Tlaxiaco</value>

</resourceProperty>

</resourceProperty>

</resourceProperty>

<resourceDescriptor name="query" wsType="query"
uriString="/MyInputControls/TEST_LIST_files/query" isNew="false">

<label>query</label>

<description>query</description>

<resourceProperty name="PROP_RESOURCE_TYPE">

<value>com.jaspersoft.jasperserver.api.metadata.common.domain.Query</value>

</resourceProperty>

<resourceProperty name="PROP_PARENT_FOLDER">

<value>/MyInputControls/TEST_LIST_files</value>

</resourceProperty>

<resourceProperty name="PROP_VERSION">

<value>1</value>

</resourceProperty>

<resourceProperty name="PROP_HAS_DATA">

<value>false</value>

</resourceProperty>

<resourceProperty name="PROP_IS_REFERENCE">
```

```
<value>false</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY">
<value>SELECT name, phone_office, billing_address_city FROM accounts order by name</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_LANGUAGE">
<value>sql</value>
</resourceProperty>
<resourceDescriptor name="" wsType="datasource" uriString="" isNew="false">
<label>null</label>
<resourceProperty name="PROP_REFERENCE_URI">
<value>/datasources/JServerJdbcDS</value>
</resourceProperty>
<resourceProperty name="PROP_IS_REFERENCE">
<value>true</value>
</resourceProperty>
</resourceDescriptor>
</resourceDescriptor>
</resourceDescriptor>
```

The query result is a set of rows that represents the full set of possible values for the input control. For each row, the service returns the value that the `runReport` service expects for that particular option in the input control. Each row also includes the column values that should be displayed in the input control when prompting users.



Employee: emil

My test list*

A & L Powers Engineering, Inc	738-555-3283	Haney
A & U Jaramillo Telecommuni...	564-555-6913	Walla Walla
A & U Stalker Telecommunica...	323-555-1226	Mill Valley
A & X Caravello Engineering,...	958-555-5890	Tlaxiaco
Abbott-Liff Electronics Holdings	133-555-6511	Cliffside
Ace-Trujillo Communications ...	143-555-3964	Lynnwood
Adair-Kolosso Transportatio...	255-555-6263	Royal Oak
Adams-Steen Transportation...	791-555-2972	Richmond

This screenshot shows single and multiple select input controls based on queries, as they are rendered by the iReport plugin for JasperServer. When the web services run report units, the rendering of input controls is left to the client application. The best way to proceed is:

- `get` the report unit;
- check for query-based input controls by looking at the `PROP_INPUTCONTROL_TYPE` resource property or each child resource descriptor where `wsType` is equal to `inputControl`;
- `get` each query based input control by setting the `IC_GET_QUERY_DATA` argument to `true`;
- render the input controls (if used in the report unit), being mindful of the input control properties (such as read only and mandatory).
- call the `runReport` service and pass the values by user selected (refer to section 2.5 “runReport” on page 19 for details).

The rows are stored in the `PROP_QUERY_DATA` resource property: for each row, a child resource property named `PROP_QUERY_DATA_ROW` contains the value and a set of children that contain the column values; these last resource properties are named `PROP_QUERY_DATA_ROW_COLUMN`.

The following schema should may elucidate the whole data structure.

```
PROP_QUERY_DATA
(
PROP_QUERY_DATA_ROW, value
(
PROP_QUERY_DATA_ROW_COLUMN, value
PROP_QUERY_DATA_ROW_COLUMN, value
PROP_QUERY_DATA_ROW_COLUMN, value
...
)


PROP_QUERY_DATA_ROW, value
(
PROP_QUERY_DATA_ROW_COLUMN, value
PROP_QUERY_DATA_ROW_COLUMN, value
PROP_QUERY_DATA_ROW_COLUMN, value
...
)


PROP_QUERY_DATA_ROW, value
(
PROP_QUERY_DATA_ROW_COLUMN, value
PROP_QUERY_DATA_ROW_COLUMN, value
PROP_QUERY_DATA_ROW_COLUMN, value
...
)
)
```

In Java, to simplify the response processing, the `ResourceDescriptor` class provides the `getQueryData()` method that returns a list of In`putControlQueryDataRow`, which is a convenient class containing all the row information (row and column values).

2.3 Put

The `put` service adds new resources to the repository or modifies existing ones. Whether the service adds or modifies a resource depends on whether the request's `isNew` resource descriptor attribute is set to `true`. The parent URI of the new resource must exist, and can be the repository root (/). When modifying a resource, you must provide the whole resource descriptor; the changes do not effect child resources.

The following XML code creates a folder called `test` inside the folder `/reports/samples`:

```
<request operationName="put" locale="en">
<resourceDescriptor name="test" wsType="folder" uriString="/reports/samples/test" isNew="true">
<label>Test</label>
<description>This is a test</description>
<resourceProperty name="PROP_PARENT_FOLDER">
<value>/reports/samples</value>
</resourceProperty>
</resourceDescriptor>
</request>
```

When adding a file resource, the data must be added as attachment to the SOAP request, and the `PROP_HAS_DATA` property must be set to `true`. When modifying a file resource, you only need to attach the file if it must be replaced; otherwise `PROP_HAS_DATA` can be set to `false`. In this case, the properties you provide are changed (for example, the label and the description).

The following Java sample creates a new image resource in the repository using the sample classes provided by JasperSoft:

```
ResourceDescriptor rdis = new ResourceDescriptor();
rdis.setResourceType(ResourceDescriptor.TYPE_IMAGE);
rdis.setName("testImageName");
rdis.setLabel("TestImageLabel");
rdis.setDescription("Test Image Description");
rdis.setParentFolder("/images");
rdis.setUriString(rdis.getParentFolder() + "/" + rdis.getName());
rdis.setWsType(ResourceDescriptor.TYPE_IMAGE);


File img = new File("/some/file/logo.jpg"));


rdis.setHasData(true);
rdis.setIsNew(true);


ResourceDescriptor result = wsclient.addOrModifyResource(rdis, img);
```

Working with report units is a bit more difficult. When creating a new report unit, the request must contain a child JRXML resource descriptor where the `PROP_RU_IS_MAIN_REPORT` property is set to `true`. This resource becomes the main JRXML of the report unit. If it is defined locally to the report, the file must be attached to the SOAP request (in this case, the parent URI for report unit's children is not relevant, and can be set to something like `<report unit parent uri>/<report unit name>_files`).

The the report unit's main JRXML already resides in the repository, the descriptor is still defined as a JRXML resource (that is, the `wsType` property must be set to `jrxml`), and the PROP_FILERESOURCE_REFERENCE_URI property must be set to the URI of the correct JRXML resource in the repository.

A second child resource is recognized during creation: a data source descriptor of the data source that the server will use to run the report. This resource is optional, and can be defined either locally to the report unit or as a reference to another resource in the repository:

- When the data source is defined locally, the resource's `wsType` must be a valid data source type, such as `jdbc`, `jndi` or `bean`.
- If the data source is defined elsewhere in the repository, its `wsType` must be set to `datasource`, which indicates an undefined resource that can be used as data source, and its `PROP_FILERESOURCE_IS_REFERENCE` property must be set to `true`. The resource's actual URI must be set in the `PROP_FILERESOURCE_REFERENCE_URI` property. Other resources such as input controls and subreports, must be added separately using the `put` service to modify the report unit.

Creating, modifying, and removing resources in a report unit is similar to working with resources in a folder. The main difference is that you must set the request's `MODIFY_REPORTUNIT_URI` argument to the URI of the report unit you want to modify. You cannot remove the JRXML resource flagged as main JRXML, but can replace or modify it. The web services don't allow you to add more than a single data source to the report unit; the report unit is always run against this data source.

2.4 Delete

This service is used to delete resources from the repository. If the specified resource is located in a report unit, you must set the request's `MODIFY_REPORTUNIT_URI` argument to the URI of the report unit you want to modify.

If you are deleting a folder, all the content is removed recursively. There is no way to recover a deleted resource or folder, so use caution when calling this service.

The following sample request deletes a resource from a report unit:

```
<request operationName="delete" locale="en">

<argument name="MODIFY_REPORTUNIT_URI">/reports/JD_New_report</argument>

<resourceDescriptor name="test_img" wsType="img"
uriString="/reports/JD_New_report_files/test_img" isNew="false">

<label>test image</label>

<description>test image</description>

</resourceDescriptor>

</request>
```

2.5 runReport

This service executes a report on the server, and returns the report's results in the specified format. The client application is responsible for prompting users for values to pass to any input controls defined in the report, as shown in the following sample request XML.

```
<request operationName="runReport" locale="en">

<argument name="RUN_OUTPUT_FORMAT">JRPRINT</argument>

<resourceDescriptor name="" wsType=""
```

```
uriString="/reports/samples/EmployeeAccounts"
isNew="false">
<label>null</label>
<parameter name="EmployeeID">emil_id</parameter>
<parameter name="TEST_LIST" isListItem="true">A &amp; L Powers Engineering, Inc</parameter>
<parameter name="TEST_LIST" isListItem="true">A &amp; U Jaramillo Telecom, Inc</parameter>
<parameter name="TEST_LIST" isListItem="true">A &amp; U Stalker Telecom, Inc</parameter>
</resourceDescriptor>
</request>
```

This example shows a parameter tag:

```
1  <!ELEMENT parameter (#PCDATA)>
2
3  <!ATTLIST parameter
4  name CDATA #REQUIRED
5  isListItem ( true | false ) false
```

`Name` (on line 4) refers to the input control to set. If the input control is of type multi-select, the list of selected values is composed of a set of parameter tags that have the same names and have the `isListItem` attribute set to `true`, indicating that the parameter is part of a list.

Notes:

- Parameter values are all treated as strings; only number, string, and date/time values are allowed.
- Numbers cannot include punctuation for the digit grouping symbol (thousands separator) and must use a period (.) as the decimal separator (if the relative parameter is not an integer).
- Dates and date/times must be represented as the number of milliseconds since January 1, 1970, 00:00:00 GMT.

The files produced are attached to the response. Specify the final format of the report by setting the request `RUN_OUTPUT_FORMAT` argument. Its possible values are: `PDF`, `JRPRINT`, `HTML`, `XLS`, `XML`, `CSV` and `RTF`. The default is `PDF`.

If the final format is set to `JRPRINT`, the data attached to the response contains a serialized instance of a JasperPrint object. This is the best format to use for clients in Java environments, because it provides the Java client with access to all of JasperReports' export options, and only relies on the server to fill the report

The following Java code shows how to access the serialized object and get the JasperPrint object:

```
FileContent content = null;
if (attachments != null && !attachments.isEmpty()) {

content = (FileContent)(attachments.values().toArray()[0]);
}


if (content == null) {
throw new Exception("No JasperPrint");
}
```

```
InputStream is = new ByteArrayInputStream(content.getData());


JasperPrint print = (JasperPrint) JRLoader.loadObject(is);
```

If the specified output format is HTML, the URI for images can be set using the `RUN_OUTPUT_IMAGES_URI` argument: the default value is `images/`. If images are found, they are attached to the response.

If only a single page needs to be printed, use the `RUN_OUTPUT_PAGE` argument, which must contain the number of pages to fill.

Reports having resource bundles associated with them can be generated in a specific languages if the locale is passed using the REPORT_LOCALE built-in report parameter. But when this parameter is not mentioned in the request, the report locale defaults to the request locale. If no locale was specified for the reuqest using the locale attribute, then the service will use the server's default locale, when generating the report.

In the following XML fragment you can see a request for running a report using the Italian locale, passed as the value of the REPORT_LOCALE built-in report parameter:

```
<request operationName="runReport" locale="fr">
<argument name="RUN_OUTPUT_FORMAT">JRPRINT</argument>
<resourceDescriptor name="" wsType=""
uriString="/reports/samples/EmployeeAccounts"
isNew="false">
<label>null</label>
<parameter name="REPORT_LOCALE"><![CDATA[it]]></parameter>
<parameter name="EmployeeID">emil_id</parameter>
</resourceDescriptor>
</request>
```

If the built-in report parameter is removed in the above request, the report would be generated in French, based on the locale attribute of the request.

2.6 Errors

When a service return an code other than 0, an error has occurred during the server execution. The exact error is described in the message field of the operation result.

If the problem is environmental, such as an incorrect service URL, an incorrect user name or password, network errors, or an unavailable service, a web services error is retuned.

The following shows an Axis connection refused error:

```
AxisFault
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException
faultSubcode:
faultString: java.net.ConnectException: Connection refused: connect
faultActor:
faultNode:
faultDetail:
{http://xml.apache.org/axis/}stackTrace:java.net.ConnectException: Connection refused: connect
```

```
at java.net.PlainSocketImpl.socketConnect(Native Method)
at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:333)
at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:195)
at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:182)
at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:366)
```

The following shows an Axis user name/password error:

```
AxisFault
faultCode: {http://xml.apache.org/axis/}HTTP
faultSubcode:
faultString: (401)Bad credentials
faultActor:
faultNode:
faultDetail:
{}:return code: 401
&lt;html&gt;&lt;head&gt;&lt;title&gt;Apache Tomcat/5.5.16 - Error re
port&lt;/title&gt;&lt;style&gt;&lt;!--H1 {font-family:Tahoma,Arial,sans-serif;co
lor:white;background-color:#525D76;font-size:22px;} H2 {font-family:Tahoma,Arial
,sans-serif;color:white;background-color:#525D76;font-size:16px;} H3 {font-famil
y:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;}
BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;} B
{font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} P {
font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;
}A {color : black;}A.name {color : black;}HR {color : #525D76;}--&gt;&lt;/style&
gt; &lt;/head&gt;&lt;body&gt;&lt;h1&gt;HTTP Status 401 - Bad credentials&lt;/h1&
gt;&lt;HR size=&quot;1&quot; noshade=&quot;noshade&quot;&gt;&lt;p&gt;&lt;b&gt;ty
pe&lt;/b&gt; Status report&lt;/p&gt;&lt;p&gt;&lt;b&gt;message&lt;/b&gt; &lt;u&gt
;Bad credentials&lt;/u&gt;&lt;/p&gt;&lt;p&gt;&lt;b&gt;description&lt;/b&gt; &lt;
u&gt;This request requires HTTP authentication (Bad credentials).&lt;/u&gt;&lt;/
p&gt;&lt;HR size=&quot;1&quot; noshade=&quot;noshade&quot;&gt;&lt;h3&gt;Apache T
omcat/5.5.16&lt;/h3&gt;&lt;/body&gt;&lt;/html&gt;
{http://xml.apache.org/axis/}HttpErrorCode:401


(401)Bad credentials
at org.apache.axis.transport.http.HTTPSender.readFromSocket(HTTPSender.java:744)
at org.apache.axis.transport.http.HTTPSender.invoke(HTTPSender.java:144)
at org.apache.axis.strategies.InvocationStrategy.visit(InvocationStrategy.java:32)
at org.apache.axis.SimpleChain.doVisiting(SimpleChain.java:118)
at org.apache.axis.SimpleChain.invoke(SimpleChain.java:83)
at org.apache.axis.client.AxisClient.invoke(AxisClient.java:165)
```

2.7 Implementation Suggestions

The iReport plugin for JasperServer relies on the SOAP web services API described in this document.

If you use Java, JasperSoft recommends that you familiarize yourself the plugin's source code, as it can help you understand how best to implement your own web services client. It is included in JasperServer professional and is also available on SourceForge at:
http://sourceforge.net/project/showfiles.php?group_id=162962

Download the ZIP file that contains the iReport plugin.

In particular, the following classes can be very illuminating:

`com.jaspersoft.jasperserver.irplugin.wsclient.WSClient`

`com.jaspersoft.jasperserver.irplugin.JServer`

The first class implements the majority of the client code, including invoking the web services, attaching files to requests, and reading files from a response. The second class stores the URL, user name, and password to connect to the web service. With these two classes, it is easy to write an entire client.

For example:

```
import net.sf.jasperreports.engine.JasperPrint;
import com.jaspersoft.jasperserver.irplugin.JServer;
import com.jaspersoft.jasperserver.api.metadata.xml.domain.impl.*;

public class MyJIClient {

private JServer server = null;

public MyJIClient(String webServiceUrl, String username, String password)
{
server = new JServer();
server.setUsername(username);
server.setPassword(password);
server.setUrl(webServiceUrl);
}
public java.util.List list(String uri) throws Exception
{
ResourceDescriptor rd = new ResourceDescriptor();
rd.setWsType( ResourceDescriptor.TYPE_FOLDER);
rd.setUriString(uri);
return server.getWSClient().list(rd);
}
public ResourceDescriptor get(String uri) throws Exception
{
return get(uri, null);
}
public ResourceDescriptor get(String uri, java.util.List args) throws Exception
{
ResourceDescriptor rd = new ResourceDescriptor();
rd.setWsType( ResourceDescriptor.TYPE_REPORTUNIT);
rd.setUriString(uri);
return server.getWSClient().get(rd, null,args);
}
public JasperPrint runReport(String reportUri, java.util.Map parameters) throws
Exception
{
ResourceDescriptor rd = new ResourceDescriptor();
rd.setWsType( ResourceDescriptor.TYPE_REPORTUNIT);
rd.setUriString(reportUri);
return server.getWSClient().runReport(rd, parameters);
}
}
import net.sf.jasperreports.engine.JasperPrint;
import com.jaspersoft.jasperserver.irplugin.JServer;
import com.jaspersoft.jasperserver.api.metadata.xml.domain.impl.*;

public class MyJIClient {
```

This example class allows you to execute the list, get, and runReport web services. Please refer to the iReport plugin for a more extensive implementation of the web services.

To compile and use the class, you need the following JAR files (which can be taken from the iReport plugin archive):

- activation-1.1.jar
- axis-1.4.jar
- commons-discovery-0.2.jar
- commons-codec-1.3.jar
- jasperserver-common-ws-1.2.1.jar
- jasperserver-ireport-plugin-1.2.1.jar
- jaxrpc.jar
- mail-1.4.jar
- saaj.jar
- wsdl4j-1.5.1.jar

Most of these JAR files are from the Axis2 package, with these exceptions ():

- activation-1.1.jar
- mail-1.4.jar

If necessary, you can marshal and unmarshal request and response objects by using the following classes:

- `com.jaspersoft.jasperserver.ws.xml.Marshaller`
- `com.jaspersoft.jasperserver.ws.xml.Unmarshaller`

If you use a development environment other than Java (such as .NET), you can easily generate a client from the WSDL.

3 Contacting JasperSoft

If you have the evaluation license and would like to purchase the commercial license, please contact JasperSoft Sales at 415-348-2319.

If you have purchased JasperSoft Professional, and have questions, suggestions, or problems, please contact JasperSoft Technical Support:

Phone: 415-677-2245
Toll-free Phone (within the U.S.): 877-600-5767
Email: support@jaspersoft.com
Web site: http://www.jaspersoft.com/ss_overview.html

Appendix AWSDL

The following example shows the WSDL document for the JasperServer web services:

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions targetNamespace="http://127.0.0.1:8080/jasperserver/services/repository"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://127.0.0.1:8080/jasperserver/services/repository"
xmlns:intf="http://127.0.0.1:8080/jasperserver/services/repository"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!--WSDL created by Apache Axis version: 1.3

Built on Oct 05, 2005 (05:23:37 EDT)-->

<wsdl:message name="putRequest">

<wsdl:part name="requestXmlString" type="xsd:string"/>

</wsdl:message>

<wsdl:message name="getRequest">

<wsdl:part name="requestXmlString" type="xsd:string"/>

</wsdl:message>

<wsdl:message name="deleteResponse">

<wsdl:part name="deleteReturn" type="xsd:string"/>

</wsdl:message>

<wsdl:message name="listRequest">

<wsdl:part name="requestXmlString" type="xsd:string"/>

</wsdl:message>

<wsdl:message name="listResponse">

<wsdl:part name="listReturn" type="xsd:string"/>

</wsdl:message>

<wsdl:message name="deleteRequest">

<wsdl:part name="requestXmlString" type="xsd:string"/>

</wsdl:message>

<wsdl:message name="getResponse">

<wsdl:part name="getReturn" type="xsd:string"/>

</wsdl:message>

<wsdl:message name="putResponse">

<wsdl:part name="putReturn" type="xsd:string"/>

</wsdl:message>

<wsdl:message name="runReportResponse">

<wsdl:part name="runReportReturn" type="xsd:string"/>

</wsdl:message>

<wsdl:message name="runReportRequest">

<wsdl:part name="requestXmlString" type="xsd:string"/>

</wsdl:message>
```

```
<wsdl:portType name="ManagementService">

<wsdl:operation name="runReport" parameterOrder="requestXmlString">

<wsdl:input message="impl:runReportRequest" name="runReportRequest"/>

<wsdl:output message="impl:runReportResponse" name="runReportResponse"/>

</wsdl:operation>

<wsdl:operation name="put" parameterOrder="requestXmlString">

<wsdl:input message="impl:putRequest" name="putRequest"/>

<wsdl:output message="impl:putResponse" name="putResponse"/>

</wsdl:operation>

<wsdl:operation name="get" parameterOrder="requestXmlString">

<wsdl:input message="impl:getRequest" name="getRequest"/>

<wsdl:output message="impl:getResponse" name="getResponse"/>

</wsdl:operation>

<wsdl:operation name="list" parameterOrder="requestXmlString">

<wsdl:input message="impl:listRequest" name="listRequest"/>

<wsdl:output message="impl:listResponse" name="listResponse"/>

</wsdl:operation>

<wsdl:operation name="delete" parameterOrder="requestXmlString">

<wsdl:input message="impl:deleteRequest" name="deleteRequest"/>

<wsdl:output message="impl:deleteResponse" name="deleteResponse"/>

</wsdl:operation>

</wsdl:portType>

<wsdl:binding name="repositorySoapBinding" type="impl:ManagementService">

<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

<wsdl:operation name="runReport">

<wsdlsoap:operation soapAction=""/>

<wsdl:input name="runReportRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://axis2.ws.jasperserver.jaspersoft.com" use="encoded"/>

</wsdl:input>

<wsdl:output name="runReportResponse">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://127.0.0.1:8080/jasperserver/services/repository" use="encoded"/>

</wsdl:output>

</wsdl:operation>

<wsdl:operation name="put">

<wsdlsoap:operation soapAction=""/>

<wsdl:input name="putRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://axis2.ws.jasperserver.jaspersoft.com" use="encoded"/>

</wsdl:input>

<wsdl:output name="putResponse">
```

```
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://127.0.0.1:8080/jasperserver/services/repository" use="encoded"/>

</wsdl:output>

</wsdl:operation>

<wsdl:operation name="get">

<wsdlsoap:operation soapAction=""/>

<wsdl:input name="getRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://axis2.ws.jasperserver.jaspersoft.com" use="encoded"/>

</wsdl:input>

<wsdl:output name="getResponse">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://127.0.0.1:8080/jasperserver/services/repository" use="encoded"/>

</wsdl:output>

</wsdl:operation>

<wsdl:operation name="list">

<wsdlsoap:operation soapAction=""/>

<wsdl:input name="listRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://axis2.ws.jasperserver.jaspersoft.com" use="encoded"/>

</wsdl:input>

<wsdl:output name="listResponse">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://127.0.0.1:8080/jasperserver/services/repository" use="encoded"/>

</wsdl:output>

</wsdl:operation>

<wsdl:operation name="delete">

<wsdlsoap:operation soapAction=""/>

<wsdl:input name="deleteRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://axis2.ws.jasperserver.jaspersoft.com" use="encoded"/>

</wsdl:input>

<wsdl:output name="deleteResponse">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://127.0.0.1:8080/jasperserver/services/repository" use="encoded"/>

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:service name="ManagementServiceService">

<wsdl:port binding="impl:repositorySoapBinding" name="repository">

<wsdlsoap:address location="http://127.0.0.1:8080/jasperserver/services/repository"/>

</wsdl:port>

</wsdl:service>

</wsdl:definitions>
```

Appendix BAPI constants

The constants that the services require are defined in the following classes:

- `com.jaspersoft.jasperserver.api.metadata.xml.domain.impl.Argument`
- `com.jaspersoft.jasperserver.ws.xml.ResourceDescriptor`

The following shows the constant names in quotation marks:

```
// resource wsTypes
TYPE_FOLDER = "folder";
TYPE_REPORTUNIT = "reportUnit";
TYPE_DATASOURCE = "datasource";
TYPE_DATASOURCE_JDBC = "jdbc";
TYPE_DATASOURCE_JNDI = "jndi";
TYPE_DATASOURCE_BEAN = "bean";
TYPE_IMAGE = "img";
TYPE_FONT = "font";
TYPE_JRXML = "jrxml";
TYPE_CLASS_JAR = "jar";
TYPE_RESOURCE_BUNDLE = "prop";
TYPE_REFERENCE = "reference";
TYPE_INPUT_CONTROL = "inputControl";
TYPE_DATA_TYPE = "dataType";
TYPE_OLAP_MONDRIAN_CONNECTION = "olapMondrianCon";
TYPE_OLAP_XMLA_CONNECTION = "olapXmlaCon";
TYPE_MONDRIAN_SCHEMA = "olapMondrianSchema";
TYPE_XMLA_CONNTCTION = "xmlaConnection";
TYPE_UNKNOW = "unknow";
TYPE_LOV = "lov"; // List of values...
TYPE_QUERY = "query";
// These constants are copied here from DataType for facility
DT_TYPE_TEXT = 1;
DT_TYPE_NUMBER = 2;
DT_TYPE_DATE = 3;
DT_TYPE_DATE_TIME = 4;
// These constants are copied here from InputControl for facility
IC_TYPE_BOOLEAN = 1;
IC_TYPE_SINGLE_VALUE = 2;
IC_TYPE_SINGLE_SELECT_LIST_OF_VALUES = 3;
IC_TYPE_SINGLE_SELECT_QUERY = 4;
IC_TYPE_MULTI_VALUE = 5;
```

```
IC_TYPE_MULTI_SELECT_LIST_OF_VALUES = 6;
IC_TYPE_MULTI_SELECT_QUERY = 7;
IC_TYPE_SINGLE_SELECT_LIST_OF_VALUES_RADIO = 8;
IC_TYPE_SINGLE_SELECT_QUERY_RADIO = 9;
IC_TYPE_MULTI_SELECT_LIST_OF_VALUES_CHECKBOX = 10;
IC_TYPE_MULTI_SELECT_QUERY_CHECKBOX = 11;
PROP_VERSION = "PROP_VERSION";
PROP_PARENT_FOLDER = "PROP_PARENT_FOLDER";
PROP_RESOURCE_TYPE = "PROP_RESOURCE_TYPE";
PROP_CREATION_DATE = "PROP_CREATION_DATE";
// File resource properties
PROP_FILERESOURCE_HAS_DATA = "PROP_HAS_DATA";
PROP_FILERESOURCE_IS_REFERENCE = "PROP_IS_REFERENCE";
PROP_FILERESOURCE_REFERENCE_URI = "PROP_REFERENCE_URI";
PROP_FILERESOURCE_WSTYPE = "PROP_WSTYPE";
// Datasource properties
PROP_DATASOURCE_DRIVER_CLASS = "PROP_DATASOURCE_DRIVER_CLASS";
PROP_DATASOURCE_CONNECTION_URL = "PROP_DATASOURCE_CONNECTION_URL";
PROP_DATASOURCE_USERNAME = "PROP_DATASOURCE_USERNAME";
PROP_DATASOURCE_PASSWORD = "PROP_DATASOURCE_PASSWORD";
PROP_DATASOURCE_JNDI_NAME = "PROP_DATASOURCE_JNDI_NAME";
PROP_DATASOURCE_BEAN_NAME = "PROP_DATASOURCE_BEAN_NAME";
PROP_DATASOURCE_BEAN_METHOD = "PROP_DATASOURCE_BEAN_METHOD";
// ReportUnit resource properties
PROP_RU_DATASOURCE_TYPE = "PROP_RU_DATASOURCE_TYPE";
PROP_RU_IS_MAIN_REPORT = "PROP_RU_IS_MAIN_REPORT";
PROP_RU_INPUTCONTROL_RENDERING_VIEW = "PROP_RU_INPUTCONTROL_RENDERING_VIEW";
PROP_RU_REPORT_RENDERING_VIEW = "PROP_RU_REPORT_RENDERING_VIEW";
// DataType resource properties
PROP_DATATYPE_STRICT_MAX = "PROP_DATATYPE_STRICT_MAX";
PROP_DATATYPE_STRICT_MIN = "PROP_DATATYPE_STRICT_MIN";
PROP_DATATYPE_MIN_VALUE = "PROP_DATATYPE_MIN_VALUE";
PROP_DATATYPE_MAX_VALUE = "PROP_DATATYPE_MAX_VALUE";
PROP_DATATYPE_PATTERN = "PROP_DATATYPE_PATTERN";
PROP_DATATYPE_TYPE = "PROP_DATATYPE_TYPE";
// ListOfValues resource properties
PROP_LOV = "PROP_LOV";
PROP_LOV_LABEL = "PROP_LOV_LABEL";
PROP_LOV_VALUE = "PROP_LOV_VALUE";
// InputControl resource properties
PROP_INPUTCONTROL_TYPE = "PROP_INPUTCONTROL_TYPE";
```

```
PROP_INPUTCONTROL_IS_MANDATORY = "PROP_INPUTCONTROL_IS_MANDATORY";

PROP_INPUTCONTROL_IS_READONLY = "PROP_INPUTCONTROL_IS_READONLY";

// SQL resource properties

PROP_QUERY = "PROP_QUERY";

PROP_QUERY_VISIBLE_COLUMNS = "PROP_QUERY_VISIBLE_COLUMNS";

PROP_QUERY_VISIBLE_COLUMN_NAME = "PROP_QUERY_VISIBLE_COLUMN_NAME";

PROP_QUERY_VALUE_COLUMN = "PROP_QUERY_VALUE_COLUMN";

PROP_QUERY_LANGUAGE = "PROP_QUERY_LANGUAGE";

// SQL resource properties (data)

PROP_QUERY_DATA = "PROP_QUERY_DATA";

PROP_QUERY_DATA_ROW = "PROP_QUERY_DATA_ROW";

PROP_QUERY_DATA_ROW_COLUMN = "PROP_QUERY_DATA_ROW_COLUMN";




// Arguments

MODIFY_REPORTUNIT = "MODIFY_REPORTUNIT_URI";

CREATE_REPORTUNIT = "CREATE_REPORTUNIT_BOOLEAN";

LIST_DATASOURCES = "LIST_DATASOURCES";

IC_GET_QUERY_DATA = "IC_GET_QUERY_DATA";

VALUE_TRUE = "true";

VALUE_FALSE = "false";

RUN_OUTPUT_FORMAT = "RUN_OUTPUT_FORMAT";

RUN_OUTPUT_FORMAT_PDF = "PDF";

RUN_OUTPUT_FORMAT_JRPRINT = "JRPRINT";

RUN_OUTPUT_FORMAT_HTML = "HTML";

RUN_OUTPUT_FORMAT_XLS = "XLS";

RUN_OUTPUT_FORMAT_XML = "XML";

RUN_OUTPUT_FORMAT_CSV = "CSV";

RUN_OUTPUT_FORMAT_RTF = "RTF";

RUN_OUTPUT_IMAGES_URI = "IMAGES_URI";

RUN_OUTPUT_PAGE = "PAGE";
```