



JasperReports Server Community Project Web Services Guide Version 4.0

<http://www.jaspersoft.com/>

© 2011 Copyright (C) 2011 Jaspersoft Corporation. All rights reserved.

Printed in the U.S.A. Jaspersoft, the Jaspersoft logo, JasperAnalysis, JasperServer, JasperETL, JasperReports, JasperReports Server, and iReport, are trademarks and/or registered trademarks of Jaspersoft Corporation in the United States and in jurisdictions throughout the world.

All other company and product names are or may be trade names or trademarks of their respective owners.

Table of Contents

1 Introduction.....	4
2 Repository Web Service.....	4
2.1 ResourceDescriptor.....	4
2.2 Request and Operation Result.....	8
2.3 Operations in the Repository Service.....	11
2.3.1 List.....	11
2.3.2 Get.....	13
2.3.3 Put.....	19
2.3.4 Delete.....	21
2.3.5 Move.....	21
2.3.6 Copy.....	21
2.3.7 runReport.....	22
2.3.7.1 Report Output.....	22
2.3.7.2 Report Locales.....	23
2.4 Errors and Implementation Suggestions.....	23
2.4.1 Errors.....	23
2.4.2 Implementation Suggestions.....	25
3 Report Scheduling Web Service.....	26
3.1 Types Defined in the WSDL.....	27
3.2 Operations in the Scheduling Service.....	28
3.2.1 Operation Descriptions.....	28
3.2.2 Example Request and Operation Result.....	28
3.2.3 Java Client Classes.....	30
4 Web Services for Administration.....	31
4.1 Types Defined in the WSDL.....	31
4.2 Users and Roles.....	33
4.2.1 findUsers.....	33
4.2.2 putUser.....	34
4.2.3 deleteUser.....	34
4.2.4 findRoles.....	35
4.2.5 putRole.....	36
4.2.6 updateRoleName.....	37
4.2.7 deleteUser.....	38
4.3 Permissions.....	38
4.3.1 getPermissionsForObject.....	39
4.3.2 putPermission.....	40

4.3.3 deletePermission.....	41
4.4 Related Files.....	41
Appendix A Repository Service API Constants.....	42

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.

JasperReports Server offers these web services:

- The repository web service lets you explore the repository, modify report units and several other resource types, and run reports with or without input parameters.
- The scheduling web service lets you schedule reports, retrieve information about jobs, and to delete them.
- The administrative web service lets you manage users and roles and define object permissions.

The web services 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:

Web Service	URL
Repository	http://localhost:8080/jasperserver/services/repository
Scheduling	http://localhost:8080/jasperserver/services/ReportScheduler
Users and Roles	http://localhost:8080/jasperserver/services/UserAndRoleManagementService
Permissions	http://localhost:8080/jasperserver/services/PermissionsManagementService

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

Note: The context name (jasperserver) may also depend on the specific installation of JasperReports Server.

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

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 JasperReports Server, 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.

2 Repository Web Service

The repository web service is comprised of seven methods: list, get, put, delete, move, copy, and runReport.

You can retrieve the WSDL document that describes the repository service by invoking the URL of the service and appending the string `?wsdl` to it. For example:

<http://localhost:8080/jasperserver/services/repository?wsdl>

The complete WSDL is detailed in Appendix A “Error: Reference source not found” on page Error: Reference source not found.

2.1 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 repository web service operates on all resources.

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	Spring bean data source
contentResource	The output of a report.
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

Value	Description
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
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 JasperReports Server
xmlaConnection	XMLA Connection

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

The `isNew` attribute is used with the `put` operation 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">

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

<resourceProperty name="PROP_DATASOURCE_USERNAME">
<value>username</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 can be omitted. In many cases, the only information required to identify a resource in the repository is 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.

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

2.2 Request and Operation Result

The repository web services operation takes 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, move, copy, 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.

The operation name is redundant, since the operation to execute is intrinsic in the invoked service. However, including the name can clarify 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="1.2.0">
<returnCode>0</returnCode>

....
several resource descriptors....
....
</operationResult>
```

2.3 Operations in the Repository Service

2.3.1 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="1.2.0">
  <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 repository web 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 repository web 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.

The list operation also provides request arguments to get the list of all resources of a given type in the repository, for example all the report units. This use of the list operation has the following syntax:

```
<request operationName="list">
<argument name="LIST_RESOURCES"/>
<argument name="RESOURCE_TYPE">reportUnit</argument>
<argument name="PARENT_DIRECTORY">/reports</argument>
</request>
or
<request operationName="list">
<argument name="LIST_RESOURCES"/>
<argument name="RESOURCE_TYPE">reportUnit</argument>
<argument name="START_FROM_DIRECTORY">/reports</argument>
</request>
```

No value is needed for the `LIST_RESOURCES` argument. The value of the `RESOURCE_TYPE` argument can be any value of `wsType` except `folder`. The `PARENT_DIRECTORY` argument is the URI of the folder in which you want to look for resources. If you want to look for the resources in a branch of the repository, use the `START_FROM_DIRECTORY` argument.

Several Java methods in `com.jaspersoft.jasperserver.irplugin.wsclient.WSClient` use `LIST_RESOURCES`:

- `list(String xmlRequest)` - Sends any custom request, including one using `LIST_RESOURCES` as shown above.
- `listResources(String type)` - Lists all resources of the given type in the repository visible to the logged in user.
- `listResourcesInFolder(String type, String parentFolder)` - Lists resources of the given type in the folder.
- `listResourcesUnderFolder(String type, String ancestorFolder)` - Lists resources of the given type in the folder and the entire tree beneath that folder.

There is a shortcut for the list operation to get the list of data sources available in the repository. To do so, set the request's `LIST_DATASOURCES` argument to `true`.

```
<request operationName="list" locale="en">
<argument name="LIST_DATASOURCES">true</argument>
</request>
```

Java sample code:

```
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')
{
$error_message = $result->getFault()->faultstring;
}
else
{
$folders = getResourceDescriptors($result);
}
```

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

2.3.2 Get

The `get` operation 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.

The following sample request gets 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`. By default, the attachments format is MIME. You can use DIME attachments by specifying the `USE_DIME_ATTACHMENTS` argument in the request.

A get call 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 & L Powers Engineering, Inc</value>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">

```

```

<value>A & 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 & U Jaramillo Telecommunications, Inc</value>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
<value>A & 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 & U Stalker Telecommunications, Inc</value>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
<value>A & 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 & X Caravello Engineering, Inc</value>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">
<value>A & X Caravello Engineering, Inc</value>
</resourceProperty>
<resourceProperty name="PROP_QUERY_DATA_ROW_COLUMN">

```



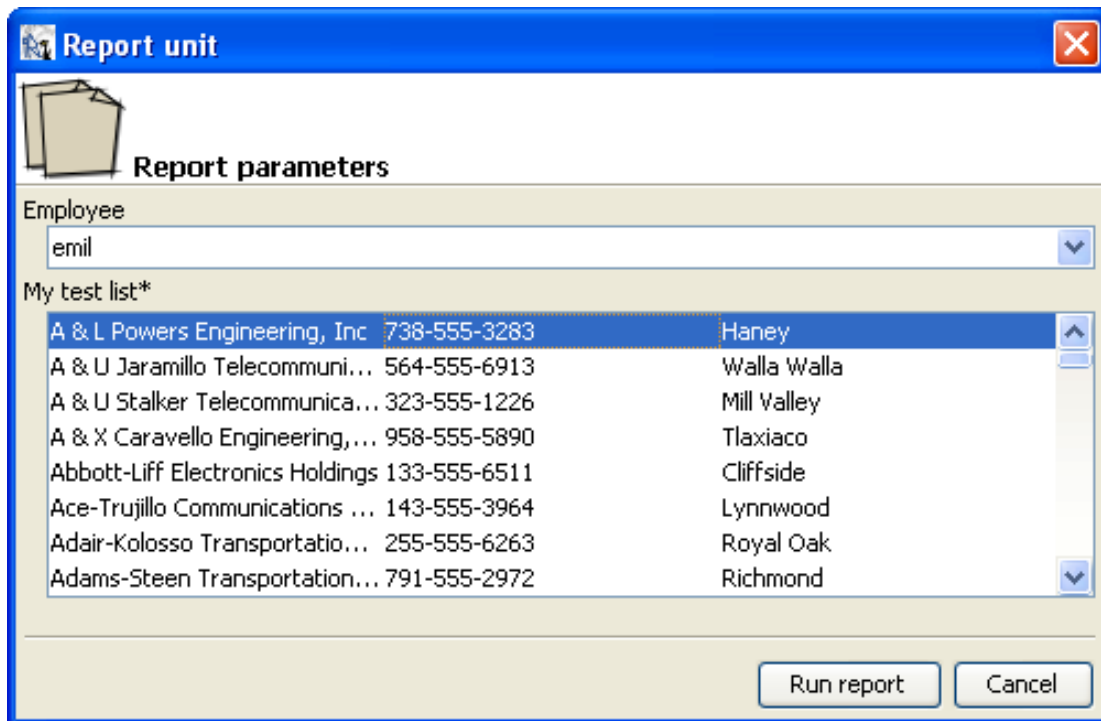
```

<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 repository web 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.



The screenshot above shows single and multiple select input controls based on queries, as they are rendered by the iReport plugin for JasperReports Server. 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 user-selected values (refer to section 2.3.7 “runReport” on page 22 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 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 response processing, the `ResourceDescriptor` class provides the `getQueryData()` method that returns a list of `InputControlQueryDataRow`, which is a convenient class containing all the row information (row and column values).

2.3.3 Put

The `put` operation 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 impact 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 an 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 with JasperReports Server:

```
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 complicated. 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`).

If 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 a 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` operation 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 repository web service doesn'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.

2.3.4 Delete

This operation deletes 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 its 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">
    <label>test image</label>
    <description>test image</description>
  </resourceDescriptor>
</request>
```

2.3.5 Move

This operation moves a repository folder or resource to a different location/folder in the repository. The operation exposes the API repository service `moveResource` and `moveFolder` methods.

The operation expects (as part of the request) a resource descriptor that identifies the resource or folder to be moved. The new location of the resource/folder must be provided as the value of the `DESTINATION_URI` request argument. The destination URI must resolve to an existing repository folder.

The following request moves the report unit located at `/Reports/NewReport` to `/MyReports`:

```
<request operationName="move" locale="en">
  <argument name="DESTINATION_URI">/MyReports</argument>
  <resourceDescriptor name="NewReport" wsType="reportUnit" uriString="/Reports/NewReport">
  </resourceDescriptor>
</request>
```

2.3.6 Copy

This operation creates in the repository a copy of an existing resource or folder. The operation exposes the repository service `copyResource` and `copyFolder` API methods.

The resource or folder to be copied is sent as the resource descriptor of the request; the caller does not need to provide the full resource information but only the information required to locate the resource.

The full location of the copy must be provided as the value of the `DESTINATION_URI` request argument. If this location already exists in the repository at the moment the operation is called, the server automatically changes the name part of the destination URI and saves the resource or folder copy at the new URI.

The copy operation response includes a descriptor for the saved resource or folder copy. The response descriptor is particularly useful in determining whether the copy has been created at the specified destination URI or at a different/generated URI.

When a folder is being copied, all its subfolders and contained resources are recursively copied.

The following request copies the report unit located at /Reports/NewReport to /MyReports/NewReportCopy:

```
<request operationName="copy" locale="en">
  <argument name="DESTINATION_URI">/MyReports/NewReportCopy</argument>
  <resourceDescriptor name="NewReport" wsType="reportUnit" uriString="/Reports/NewReport">
  </resourceDescriptor>
</request>
```

2.3.7 runReport

This operation 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 & L Powers Engineering, Inc</parameter>
    <parameter name="TEST_LIST" isListItem="true">A & U Jaramillo Telecom, Inc</parameter>
    <parameter name="TEST_LIST" isListItem="true">A & 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.

2.3.7.1 Report Output

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 index of page to fill.

2.3.7.2 Report Locales

Reports that have resource bundles for localization can be generated in a specific language when the locale is passed using the `REPORT_LOCALE` built-in report parameter. If this parameter is not specified in the web service request, the report locale defaults to the request's locale. If no locale was specified for the request, the report is generated in the server's default locale.

The following XML fragment shows a request to run a report in the Italian locale, which is 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 from this request, the report is generated in French, based on the locale attribute of the request.

2.4 Errors and Implementation Suggestions

2.4.1 Errors

When the repository web service returns a 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 returned.

The following shows an Axis connection refused error:

```
AxisFault
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException
```

```
AxisFault
faultCode: {http://xml.apache.org/axis/}HTTP
faultSubcode:
faultString: (401)Bad credentials
faultActor:
faultNode:
faultDetail:
{}:return code: 401

<html><head><title>Apache Tomcat/5.5.16 - Error re
port</title><style>!--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;}--></style>
</head><body><h1>HTTP Status 401 - Bad credentials</h1>
<hr size="1" noshade="noshade"><p><b>ty
pe</b> Status report</p><p><b>message</b>
<u>Bad credentials</u><p><p><b>description</b>
<u>This request requires HTTP authentication (Bad credentials)</u>
<hr size="1" noshade="noshade"><h3>Apache T
omcat/5.5.16</h3></body></html>

{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)

```

2.4.2 Implementation Suggestions

The iReport plugin for JasperReports Server relies on the repository web service described in this document.

If you use Java, JasperSoft recommends that you familiarize yourself with the plugin's source code, as it can help you understand how best to implement your own web services client. It is included in JasperReports Server 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:

```

1  import net.sf.jasperreports.engine.JasperPrint;
2  import com.jaspersoft.jasperserver.irplugin.JServer;
3  import com.jaspersoft.jasperserver.api.metadata.xml.domain.impl.*;
4
5  public class MyJIClient {
6
7      private JServer server = null;
8
9      public MyJIClient(String webServiceUrl, String username, String password)
10     {
11         server = new JServer();
12         server.setUsername(username);
13         server.setPassword(password);
14         server.setUrl(webServiceUrl);
15     }
16     public java.util.List list(String uri) throws Exception
17     {
18         ResourceDescriptor rd = new ResourceDescriptor();
19         rd.setWsType(ResourceDescriptor.TYPE_FOLDER);
20         rd.setUriString(uri);
21         return server.getWSClient().list(rd);
22     }
23     public ResourceDescriptor get(String uri) throws Exception
24     {
25         return get(uri, null);
26     }
27     public ResourceDescriptor get(String uri, java.util.List args) throws Exception
28     {
29         ResourceDescriptor rd = new ResourceDescriptor();

```

```
30 rd.setWsType( ResourceDescriptor.TYPE_REPORTUNIT);
31 rd.setUriString(uri);
32 return server.getWSClient().get(rd, null, args);
33 }
34 public JasperPrint runReport(String reportUri, java.util.Map parameters) throws
Exception
35 {
36 ResourceDescriptor rd = new ResourceDescriptor();
37 rd.setWsType( ResourceDescriptor.TYPE_REPORTUNIT);
38 rd.setUriString(reportUri);
39 return server.getWSClient().runReport(rd, parameters);
40 }
41 }
```

This example class allows you to execute the list, get, and runReport operations of the repository web service. 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.jar
- jasperserver-ireport-plugin-1.2.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 Report Scheduling Web Service

The scheduling web service exposes JasperReports Server's report scheduling functionality to integrating applications by the means of a dedicated web service. The web service is the equivalent of the API report scheduling service (com.jaspersoft.jasperserver.api.engine.scheduling.service.ReportSchedulingService) and exposes the same operations as this API service.

The service works via XML-RPC calls that use the SOAP encoding. It uses the HTTP protocol to send and receive requests and responses. By default, it is deployed at /services/ReportScheduler. You can retrieve the service WSDL (Web Service Description Language) document by appending ?wsdl to the service URL. For example:

<http://localhost:8080/services/ReportScheduler/?wsdl>

3.1 Types Defined in the WSDL

The WSDL defines several types that are used by the parameters and operation result of the service. The types belong to the <http://www.jasperforge.org/jasperserver/ws> namespace.

This section provides a partial list of the types used for report scheduling; for the complete reference, refer to the the WSDL document. The report scheduling types include:

- **Job** – encapsulates all the attributes of a report job. This type is used when full report job details are passed to or returned by an operation. Its type elements include:
 - **id** and **version** – required when updating a report job.
 - **reportUnitURI** – the URI of the report.
 - **username** – set automatically to the name of the calling user when a report job is created.
 - **label** – the job label.
 - **simpleTrigger** or **calendarTrigger** – the job trigger, which can be either a simple (fixed interval) trigger or a calendar trigger.
 - **parameters** – the list of report parameter/input control values.
 - **baseOutputFilename** – the base name for the job output.
 - **outputFormats** – the list of job output formats (as strings). JasperReports Server has built-in support for the following formats: PDF, HTML, XLS, RTF and CSV.
 - **outputLocale** – the String representation of a *java.util.Locale* to be used as report locale.
 - **repositoryDestination** – the location in the repository where the report output is saved.
 - **mailNotification** – information regarding the email notification that is sent when the job executes. Set this value to null to suppress notifications.
- **JobSimpleTrigger** – a job trigger that fires at fixed intervals. It has the following attributes:
 - **startDate** and **endDate** – the start and end dates of the job.
 - **timezone** – the timezone of the start and end dates.
 - **occurrenceCount** – how many times to run the job. If a single run job is wanted, 1 should be used as occurrence count; if the job is to be fired indefinitely (or until the end date) -1 should be used.
 - **recurrenceInterval** and **recurrenceIntervalUnit** – the interval at which the job should recur: MINUTE, HOUR, DAY and WEEK.
- **JobCalendarTrigger** – a job trigger that fires at time specified by a CRON-like expression. It has the following attributes:
 - **startDate** and **endDate** – the start and end dates of the job.
 - **timezone** – the timezone of the start and end dates.
 - **minutes** – the minute or minutes of the day when the job is to run.
 - **hours** – the hour or hours of the day when the job is to run.
 - **daysType** – the way that days are specified. Possible values include ALL, WEEK or MONTH.
 - **weekDays** – used when **daysType** is WEEK, this indicates the week days from Saturday (1) to Sunday (7).
 - **monthDays** – used when **daysType** is MONTH, this indicates the month days.
 - **months** – the months (from 0 to 11) on which the job should be triggered.
- **JobRepositoryDestination** – contains information about where to save the report job output in the repository, including:
 - **folderURI** – the URI of the folder where the report output will be saved.
 - **sequentialFileNames** – a flag indicating whether to append timestamps to the base output name.
- **JobMailNotification** – encapsulates the attributes of the mail notification to send regarding the report job. It has the following attributes:

- `toAddresses` – the list of email addresses to which the notification will be sent.
- `subject` – the subject of the email notification.
- `resultSendType` – indicates whether to attach the report output to the email; the value is either `SEND` (only the messages is sent) or `SEND_ATTACHMENT` (the report output is sent along as a message attachment).
- `JobSummary` – This type is used when a list of report jobs is retrieved via the service. The full report job information can be retrieved individually for required jobs. The job summary includes:
 - `id` – the unique identifier of the job.
 - `label` – the display label of the job.
 - `state` – the state of the job. For example, `NORMAL` indicates that the job is waiting for next execution; `EXECUTING` means the job is running.
 - `previousFireTime` – the last/previous run time.
 - `nextFireTime` – the next run time.

3.2 Operations in the Scheduling Service

3.2.1 Operation Descriptions

The report scheduling web service consists of the following operations:

- `getAllJobs` – returns the list of all accessible report jobs.
- `getReportJobs` – returns the list of all accessible report jobs for a specific report (whose URL is sent as a parameter).
- `scheduleJob` – schedules a new job. The job details must be sent as parameters; the operation returns the saved job details as its result.
- `updateJob` – updates an existing job. The full job details (as retrieved via `getJob`) must be sent as a parameter; the operation returns the updated job details as saved by the JasperReports Server scheduling service.
- `getJob` – returns the full job details of a report job whose ID is sent as a parameter.
- `deleteJob` and `deleteJobs` – delete a single or several report job specified by their IDs. These operations do not return any information.

If an exception occurs while processing an operation request, the exception is converted to a SOAP fault which is sent as its response. In this case, the exception stacktrace is included in the response, and can be used for debugging.

Exceptions thrown by the JasperReports Server code have localizable messages. The operation caller can specify the locale in which the messages of such exceptions are returned by setting a SOAP envelope header. The header should be named *locale* and should use <http://www.jasperforge.org/jasperserver/ws> as namespace; the header value is a string representation of the desired message locale. For more information, refer to 2.2 “Request and Operation Result” on page 8.

3.2.2 Example Request and Operation Result

This is the full SOAP request for a `scheduleJob` operation that creates a job with four report parameters:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:scheduleJob soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://www.jasperforge.org/jasperserver/ws">
      <job xsi:type="ns1:Job">
        <reportUnitURI xsi:type="xsd:string">/reports/samples/SalesByMonth</reportUnitURI>
        <username xsi:type="xsd:string" xsi:nil="true"/>
        <label xsi:type="xsd:string">Label 3</label>
        <description xsi:type="xsd:string">Description 3</description>
      </job>
    </ns1:scheduleJob>
  </soapenv:Body>
</soapenv:Envelope>
```

```

    <simpleTrigger xsi:type="ns1:JobSimpleTrigger">
      <timezone xsi:type="xsd:string" xsi:nil="true"/>
      <startDate xsi:type="xsd:dateTime">2008-10-
09T09:25:00.000Z</startDate>
      <endDate xsi:type="xsd:dateTime" xsi:nil="true"/>
      <occurrenceCount xsi:type="xsd:int">1</occurrenceCount>
      <recurrenceInterval xsi:type="xsd:int" xsi:nil="true"/>
      <recurrenceIntervalUnit xsi:type="ns1:IntervalUnit" xsi:nil="true"/>
    </simpleTrigger>
    <calendarTrigger xsi:type="ns1:JobCalendarTrigger" xsi:nil="true"/>
    <parameters soapenc:arrayType="ns1:JobParameter[4]"
xsi:type="soapenc:Array" xmlns:soapenc="http://schemas.xmlsoap.org/soap/
encoding/">
      <parameters xsi:type="ns1:JobParameter">
        <name xsi:type="xsd:string">TextInput</name>
        <value xsi:type="soapenc:int">22</value>
      </parameters>
      <parameters xsi:type="ns1:JobParameter">
        <name xsi:type="xsd:string">CheckboxInput</name>
        <value xsi:type="soapenc:boolean">true</value>
      </parameters>
      <parameters xsi:type="ns1:JobParameter">
        <name xsi:type="xsd:string">ListInput</name>
        <value xsi:type="soapenc:string">2</value>
      </parameters>
      <parameters xsi:type="ns1:JobParameter">
        <name xsi:type="xsd:string">DateInput</name>
        <value xsi:type="xsd:dateTime">2007-10-09T09:00:00.000Z</value>
      </parameters>
    </parameters>
    <baseOutputFilename xsi:type="xsd:string">Sales3</baseOutputFilename>
    <outputFormats soapenc:arrayType="xsd:string[1]" xsi:type="soapenc:Ar-
ray" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
      <outputFormats xsi:type="xsd:string">PDF</outputFormats>
    </outputFormats>
    <outputLocale xsi:type="xsd:string" xsi:nil="true"/>
    <repositoryDestination xsi:type="ns1:JobRepositoryDestination">
      <folderURI xsi:type="xsd:string">/ContentFiles</folderURI>
      <sequentialFileNames xsi:type="xsd:boolean">false</sequentialFile-
names>
      <overwriteFiles xsi:type="xsd:boolean">false</overwriteFiles>
    </repositoryDestination>
    <mailNotification xsi:type="ns1:JobMailNotification" xsi:nil="true"/>
  </job>
</ns1:scheduleJob>
</soapenv:Body>
</soapenv:Envelope>

```

The response of the request contains the job details as saved by the server:

```

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xm-
lns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:scheduleJobResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://www.jasperforge.org/jasperserver/ws">
      <scheduleJobReturn xsi:type="ns1:job">
        <id xsi:type="xsd:long">7</id>
        <version xsi:type="xsd:int">0</version>
        <reportUnitURI xsi:type="xsd:string">/reports/samples/SalesByMonth</re-
portUnitURI>
        <username xsi:type="xsd:string">tomcat</username>
        <label xsi:type="xsd:string">Label 3</label>
        <description xsi:type="xsd:string">Description 3</description>
        <simpleTrigger xsi:type="ns1:jobSimpleTrigger">
          <id xsi:type="xsd:long">7</id>
          <version xsi:type="xsd:int">0</version>
          <timezone xsi:type="xsd:string">Europe/Minsk</timezone>

```

```

    <startDate xsi:type="xsd:dateTime">2008-10-
09T09:25:00.000Z</startDate>
    <endDate xsi:type="xsd:dateTime" xsi:nil="true"/>
    <occurrenceCount xsi:type="xsd:int">1</occurrenceCount>
    <recurrenceInterval xsi:type="xsd:int" xsi:nil="true"/>
    <recurrenceIntervalUnit xsi:type="ns1:IntervalUnit" xsi:nil="true"/>
  </simpleTrigger>
  <calendarTrigger xsi:type="ns1:JobCalendarTrigger" xsi:nil="true"/>
  <parameters soapenc:arrayType="ns1:JobParameter[4]"
xsi:type="soapenc:Array" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encod-
ing/">
    <parameters xsi:type="ns1:jobParameter">
      <name xsi:type="xsd:string">CheckboxInput</name>
      <value xsi:type="soapenc:boolean">true</value>
    </parameters>
    <parameters xsi:type="ns1:jobParameter">
      <name xsi:type="xsd:string">TextInput</name>
      <value xsi:type="soapenc:int">22</value>
    </parameters>
    <parameters xsi:type="ns1:jobParameter">
      <name xsi:type="xsd:string">DateInput</name>
      <value xsi:type="xsd:dateTime">2007-10-09T09:00:00.000Z</value>
    </parameters>
    <parameters xsi:type="ns1:jobParameter">
      <name xsi:type="xsd:string">ListInput</name>
      <value xsi:type="soapenc:string">2</value>
    </parameters>
  </parameters>
  <baseOutputFilename xsi:type="xsd:string">Sales3</baseOutputFilename>
  <outputFormats soapenc:arrayType="xsd:string[1]" xsi:type="soapenc:Ar-
ray" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <outputFormats xsi:type="xsd:string">PDF</outputFormats>
  </outputFormats>
  <outputLocale xsi:type="xsd:string" xsi:nil="true"/>
  <repositoryDestination xsi:type="ns1:jobRepositoryDestination">
    <id xsi:type="xsd:long">7</id>
    <version xsi:type="xsd:int">0</version>
    <folderURI xsi:type="xsd:string">/ContentFiles</folderURI>
    <sequentialFileNames xsi:type="xsd:boolean">false</sequentialFile-
names>
    <overwriteFiles xsi:type="xsd:boolean">false</overwriteFiles>
  </repositoryDestination>
  <mailNotification xsi:type="ns1:JobMailNotification" xsi:nil="true"/>
</scheduleJobReturn>
</ns1:scheduleJobResponse>
</soapenv:Body>
</soapenv:Envelope>

```

3.2.3 Java Client Classes

The JasperReports Server web service client jars contain classes that can be used by Java clients to easily communicate with the report scheduling web service.

XML types used by the report scheduling web service are mapped to Java bean classes found in the `com.jaspersoft.jasperserver.ws.scheduling` package (for example, `Job`, `JobSimpleTrigger`, and `CalendarDaysType`). Instances of these classes can be used as report job objects that are sent to or returned by the web service.

The service itself is represented by Apache Axis-generated client stub classes. A facade (`com.jaspersoft.jasperserver.ws.scheduling.ReportSchedulerFacade`) has been developed on top of these classes. The facade can be instantiated by providing the information required to locate and connect to a web service (the endpoint URL and the username/password for authentication).

Jaspersoft recommends using the facade because it handles items such as the Axis client configuration and the messages locale header.

4 Web Services for Administration

Web services for administration expose a limited set of JasperReports Server's system administration functionality. There are three services:

- Users
- Roles
- Permissions

The services work via XML-RPC calls that use the SOAP encoding. They use the HTTP protocol to send and receive requests and responses. By default, they are deployed at `/services/UserAndRoleManagementService`, and `/services/PermissionsManagementService`. You can retrieve the WSDL documents by appending `?wsdl` to the service URL. For example:

<http://localhost:8080/jasperserver/services/UserAndRoleManagementService?wsdl>

and:

<http://localhost:8080/jasperserver/services/PermissionsManagementService?wsdl>

If an exception occurs while processing an operation request, the exception is converted to a SOAP fault that is sent as its response. In this case, the exception stacktrace is included in the response, which can be useful for debugging.

Exceptions thrown by JasperReports Server have localizable messages. The operation caller can specify the locale in which the messages of such exceptions are returned by setting a SOAP envelope header. The header should be named `locale` and should use `http://www.jasperforge.org/jasperserver/ws` as its namespace; the header value is a string representation of the desired message locale. For more information, refer to section 2.1 "ResourceDescriptor" on page 4.

Note: Some of the operations in these web services include elements that describe tenants and sub-organizations. In the community project of JasperReports Server, these elements are defined in the WSDL but must be null. They are reserved for use in our commercial products.

4.1 Types Defined in the WSDL

The WSDL (Web Services Description Language) document defines the types that are returned by operations of the services. The types belong to the <http://www.jasperforge.org/jasperserver/ws> namespace. For the complete reference, refer to the WSDL document in `jasperserver-ws-server-4.0.jar`.

The following table summarizes the user and role services' operations.

Operation	Parameter	Parameter Type	Description
<code>findUsers</code>	<code>criteria</code>	<code>WSUserSearchCriteria</code>	Returns a list of one or more users. criteria has username mask, list of required roles, and maxRecords; NULLs in parameters means "any." Note: the WSDL include <code>tenantID</code> and <code>includeSubOrgs</code> ; if included they must be null. They are reserved for use in our commercial products.
<code>putUser</code>	<code>user</code>	<code>WSUser</code>	Adds or updates a user. Returns a new or updated <code>WSUser</code> .
<code>deleteUser</code>	<code>user</code>	<code>WSUser</code>	Deletes the named user.
<code>findRoles</code>	<code>criteria</code>	<code>WSRoleSearchCriteria</code>	Returns <code>WSRole[]</code> , a list of roles. criteria has rolename mask, maxRecords;

JasperReports Server Web Services Guide

			<p>NULLs in parameters means "any."</p> <p>Note: The WSDL includes tenantID and includeSubOrgs; if included they must be null. They are reserved for use in our commercial products.</p>
putRoles	role	WSRole	Adds or updates a role.
updateRoleName	oldRole	WSRole	Returns WSRole. Updates a role name.
	newRole	String	New name of role.
deleteRole	role	WSRole	Deletes the named role.

The following table summarizes the permissions service operations.

Operation	Parameter	Parameter Type	Description
getPermissions forObject	targetURI	String	Repository object URI. Returns WSOBJECTPermission[], a list of permissions for the specified object.
putPermissions	objPerm	WSObjectPermission	Object permission. Returns WSOBJECTPermission, a new or updated object permission.
deletePermissions	objPerm	WSObjectPermission	Deletes the named permission.

The following summarizes the getter and setter classes for WSUser, WSRole, WSUserSearchCriteria, WSRoleSearchCriteria, WSOBJECTPermission.

Field name	Parameter/Return Type	Description
getUri	String	URI of object for this permission is set
setUri		
getPermissionRecipient	WSUser or WSRole	Represent user or role which is recipient of the permission
setPermissionRecipient		
getPermissionMask	Integer	Permission mask
setPermissionMask		

4.2 Users and Roles

The web service for administration of users and roles has these operations:

- `findUsers` and `findRoles`. Return a list of users or roles that meet specified criteria.
- `putUser` and `putRole`. Return the named user or role. If the object is not already in the database, the call creates a new one.
- `deleteUser` and `deleteRole`. Delete the named user or role.

Other resources such as input controls and subreports, must be added separately using the `put` operation to modify the report unit.

4.2.1 findUsers

In `findUsers`, the parameter `criteria` has the type `WSUserSearchCriteria` and returns type `WSUser`. `criteria` can be a username mask, an organization/tenant ID, `includeSubOrgs`, a list of required users, and `maxRecords`. Null values indicate "any."

- The mask has a SQL-like notation. For instance, `Ad%`, `U2_`.
- To limit the number of objects to return, set `maxRecords` to the desired number. To allow an infinite number of objects, set `maxRecords` to 0.

To call `findUsers`:

```
WSUserSearchCriteria searchCriteria = new WSUserSearchCriteria();
searchCriteria.setName("demo");
searchCriteria.setMaxRecords(5);
searchCriteria.setIncludeSubOrgs(false);
searchCriteria.setRequiredRoles(requiredRoles);

WSRole role = new WSRole();
role.setRoleName("ROLE_USER");
searchCriteria.setRequiredRoles(new WSRole[] {role});

WSUser[] list = binding.findUsers(searchCriteria);
```

The return is:

```
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
String getRoleName()
String getTenantId()
WSUser[] getUsers()
```

4.2.2 putUser

In putUser, the parameter user has the type WSUser and returns type WSUser.

putUser updates an existing object; if the specified user does not exist, a new one is created.

Note: Before adding users and roles, note that there is a server-side configuration which specifies the default roles that a new user can receive.

To call putUser:

```
WSUser user = new WSUser();
user.setUsername("john");
user.setEnabled(true);
user.setFullName("John Doe");
WSRole role = new WSRole();
role.setRoleName("ROLE_ANONYMOUS");
user.setRoles(new WSRole[] {role});

WSUser value = binding.putUser(user);
```

The return is:

```
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
String getRoleName()
String getTenantId()
WSUser[] getUsers()
```

4.2.3 deleteUser

In deleteUser, the parameter user has the type WSUser.

To call deleteUser:

```
WSUser user = new WSUser();
user.setUsername("john");

binding.deleteUser(user);
```

There is no return.

4.2.4 findRoles

In `findRoles`, the parameter `criteria` has the type `WSRoleSearchCriteria` and returns type `WSRole`.

- `criteria` can be a role name mask, a list of required users or roles, and `maxRecords`. Null values indicate "any."
- The mask has a SQL-like notation. For instance, `Ad%, U2_.`
- To limit the number of objects to return, set `maxRecords` to the desired number. To allow an infinite number of objects, set `maxRecords` to 0.

To call `findRoles`:

```
WSRoleSearchCriteria searchCriteria = new WSRoleSearchCriteria();

searchCriteria.setRoleName("ROLE_USER");

searchCriteria.setMaxRecords(5);

searchCriteria.setIncludeSubOrgs(false);

WSRole[] list = binding.findRoles(searchCriteria);
```

The return is:

```
String getUsername()

String getFullName()

String getPassword()

String getEmailAddress()

Boolean getExternallyDefined()

Boolean getEnabled()

Date getPreviousPasswordChangeTime()

String getTenantId()

WSRole[] getRoles()

String getRoleName()

String getTenantId()

WSUser[] getUsers()
```

4.2.5 putRole

In `putRole`, the parameter `role` has the type `WSRole` and returns type `WSRole`.

`putRole` updates an existing object; if the specified role does not exist, a new one is created.

Note: Before adding users and roles, note that there is a server-side configuration which specifies the default roles that a new user can receive.

To call `putRole`:

```
WSRole role = new WSRole();  
  
role.setRoleName("ROLE_ANONYMOUS");  
  
WSRole value = binding.putRole(role);
```

The return is:

```
String getRoleName()  
  
String getTenantId()  
  
WSUser[] getUsers()  
  
String getUsername()  
  
String getFullName()  
  
String getPassword()  
  
String getEmailAddress()  
  
Boolean getExternallyDefined()  
  
Boolean getEnabled()  
  
Date getPreviousPasswordChangeTime()  
  
String getTenantId()  
  
WSRole[] getRoles()
```

4.2.6 updateRoleName

In `updateRoleName`, the parameter `oldRole` has the type `WSRole`, and the parameter `newName` has the type `String`. The method returns an object of type `WSRole` with an updated name.

To update a role with a call to `oldRole`:

```
WSRole oldRole= new WSRole();  
  
role.setRoleName("ROLE_WS");  
  
WSRole value = binding.updateRoleName(oldRole, "ROLE_WEB_SERVICE");
```

To rename the role with a call to `newName`:

```
"ROLE_WEB_SERVICE"
```

The return for an updated role:

```
String getRoleName()  
  
String getTenantId()  
  
WSUser[] getUsers()  
  
String getUsername()  
  
String getFullName()  
  
String getPassword()  
  
String getEmailAddress()  
  
Boolean getExternallyDefined()  
  
Boolean getEnabled()  
  
Date getPreviousPasswordChangeTime()  
  
String getTenantId()  
  
WSRole[] getRoles()
```

4.2.7 deleteUser

In `deleteUser`, the parameter `user` has the type `WSUser`. In `deleteRole`, the parameter `role` has the type `WSRole`.

Here are examples of calls to `deleteUser` and `deleteRole`:

```
WSRole role = new WSRole();  
  
role.setRoleName("ROLE_WS");  
  
binding.deleteRole(role);
```

There is no return.

4.3 Permissions

The web service for administration of permissions has these operations:

- `getPermissionsForObject`. Returns a list of permissions for the specified object.
- `putPermission`. Returns the named permission. If the object is not already in the database, the call creates a new one.
- `deletePermission`. Deletes the named permission.

4.3.1 getPermissionsForObject

In `getPermissionsForObject`, the parameter `targetURI` has the type `String` and returns type `WSObjectPermission[]`.

To call `getPermissionsForObject`:

```
WSObjectPermission[] objectPermissions = binding.getPermissionsForObject("repo:/");
```

In the return, the permissioned object can be a user or role:

```
String getUri()
Object getPermissionRecipient()
int getPermissionMask()
String getRoleName()
String getTenantId()
WSUser[] getUsers()
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()
```

4.3.2 putPermission

In `putPermission`, the parameter `objPerm` has the type `WSObjectPermission` and returns type `WSObjectPermission`.

To call `putPermission`:

```
WSObjectPermission objectPermission = new WSObjectPermission();

objectPermission.setUri(resourceUri);

objectPermission.setPermissionMask(2);


WSUser wsUser = new WSUser();

wsUser.setUsername("joeuser");

wsUser.setTenantId("organization_1");


objectPermission.setPermissionRecipient(wsUser);


WSObjectPermission value = binding.putPermission(objectPermission);WSObjectPermission
objectPermission = new WSObjectPermission();

objectPermission.setUri(resourceUri);

objectPermission.setPermissionMask(2);


WSUser wsUser = new WSUser();

wsUser.setUsername("joeuser");

wsUser.setTenantId("organization_1");


objectPermission.setPermissionRecipient(wsUser);


WSObjectPermission value = binding.putPermission(objectPermission);
```

The return is:


```

String getUri()
Object getPermissionRecipient()
int getPermissionMask()
String getRoleName()
String getTenantId()
WSUser[] getUsers()
String getUsername()
String getFullName()
String getPassword()
String getEmailAddress()
Boolean getExternallyDefined()
Boolean getEnabled()
Date getPreviousPasswordChangeTime()
String getTenantId()
WSRole[] getRoles()

```

4.3.3 deletePermission

In `deletePermission`, the parameter `objPerm` has the type `WSObjectPermission`.

To call `deletePermission`:

```

WSObjectPermission objectPermission = new WSObjectPermission();

objectPermission.setUri(resourceUri);

objectPermission.setPermissionMask(2);


WSUser wsUser = new WSUser();

wsUser.setUsername("joeuser");


objectPermission.setPermissionRecipient(wsUser);


binding.deletePermission(objectPermission);

```

There is no return.

4.4 Related Files

The web services distribution files include WSDL files as well as client stub classes. The WSDL file for the administration service is in `jasperserver-ws-server-4.0.jar`. Client stub files contain return types; they are in `jasperserver-common-ws-4.0.jar`. The JasperReports Server Professional and Enterprise implementation of the services is in `ji-ws-server-4.0.jar`.

Appendix A Repository Service API 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_CONNECTION = "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";
```