IBM SPSS Collaboration and Deployment Services
Version 6 Release 0

Reporting Service Developer's Guide

Note
Before using this information and the product it supports, read the information in "Notices" on page 49.

Product Information

This edition applies to version 6, release 0, modification 0 of IBM SPSS Collaboration and Deployment Services and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Introduction to web services

What are web services?

At a high level, a web service is a set of functionality distributed across a network (LAN or the Internet) using a common communication protocol. The web service serves as an intermediary between an application and its clients, providing both a standardized information structure and a standardized communication protocol for interaction between the two.

Where other methods of distributed application architecture rely on a single programming language being used on both the application and its clients, a web service allows the use of loosely coupled services between non-homogenous platforms and languages. This provides a non-architecture-specific approach allowing, for example, Java services to communicate with C# clients, or vice versa.

Advantages to implementing application functionality as web services include the following:

- Software written in different languages (Java or C#) running on different platforms (UNIX or Windows) can exchange services and data
- Application functionality can be accessed by a variety of clients. For example, both a thin-client interface and a rich-client interface can take advantage of the web service operations.
- Updates to the service are immediately available to all service clients

Web service system architecture

Web services are deployed and made publicly available using an application server, such as WebSphere, JBoss Application Server, or Oracle WebLogic Server. The published web services are hosted by this application server to handle application requests, access permissions, and process load. A high-level architecture of how web services are implemented is displayed in the following diagram.

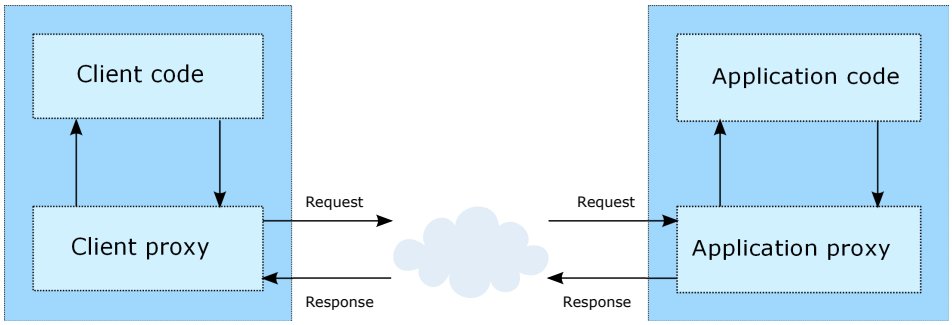

Figure 1. Web service architecture

The client code supplies input to an operation offered by a proxy class. The proxy class generates a request containing a standardized representation of the input and sends it across the network to the application. A proxy class on the server receives the request and unmarshals the contents into objects for processing by the application. Upon completing the operation, the application supplies a proxy with the output. The proxy creates a standardized representation of that output and sends the response back to the client. The client proxy unmarshals the response into native objects for subsequent processing by the client code.

Standardizing the format of the information passing between the client and the application allows a client written in one programming language to communicate with an application written in another. The proxy

classes, which are automatically generated from a web service description by a variety of toolkits, handle the translation between native programming objects and the standardized representation. See the topic "Proxies" on page 5 for more information.

Web service protocol stack

A web service implementation depends on technologies often organized in a layered stack. The implementation itself defines a standard protocol for each technology layer, with each layer depending on the layers appearing below it in the stack.

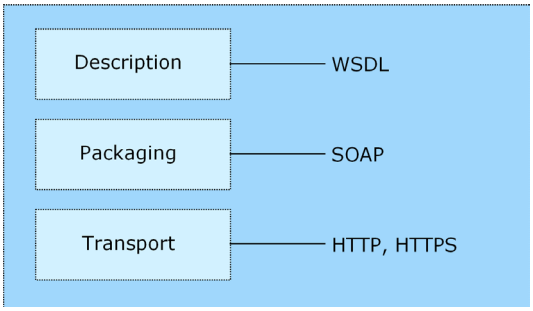

Figure 2. Web service protocol stack

Beginning at the bottom of the stack, the Transport layer defines the technology standards for communication, allowing information to move across the network. HTTP or HTTPS are often used as the standard for the transport layer.

The Packaging layer rests on top of Transport and defines the standard for structuring information for transport across the network. The SOAP format is commonly used, which offers an XML structure for packaging the data. See the topic "Simple Object Access Protocol" for more information.

The topmost layer is Description and identifies the standards used by the layers below it in the stack, as well as providing the definition of the interface available for client use. The most common means of conveying this information is through the use of a WSDL file. See the topic "Web Service Description Language" on page 3 for more information.

Simple Object Access Protocol

The Simple Object Access Protocol (SOAP) is a way to pass information between applications in an XML format.

SOAP messages are transmitted from the sending application to the receiving application, typically over an HTTP session. The actual SOAP message is made up of the Envelope element, which contains a Body element and an optional Header element.

- **Envelope.** This mandatory element is the root of the SOAP message, identifying the transmitted XML as being a SOAP packet. An envelope contains a body section and an optional header section.
- **Header.** This optional element provides an extension mechanism indicating processing information for the message. For example, if the operation using the message requires security credentials, those credentials should be part of the envelope header.
- **Body.** This element contains the message payload, the raw data being transmitted between the sending and receiving applications. The body itself may consist of multiple child elements, with an XML schema typically defining the structure of this data.

A SOAP packet and the corresponding XML is structured in the following way:

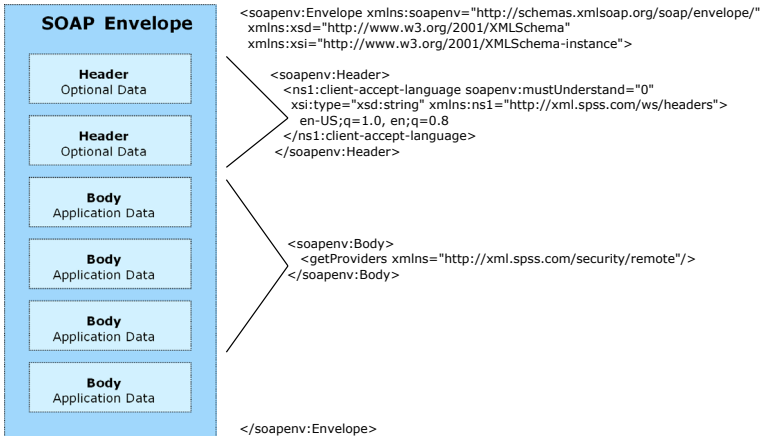

Figure 3. An example SOAP packet

Web Service Description Language

A Web Service Description Language (WSDL) file provides an XML-based map of what functionality the published web service allows, separating the implementation in the service from the interface. The WSDL defines the following:

- The access location of the web service
- Operations the web service exposes
- Parameters the exposed operations accept
- Any request or response messages associated with the operations

The WSDL provides the information necessary to generate a client-side proxy in the target programming language.

In accordance with the WSDL specification adopted by the World Wide Web Consortium, information in the WSDL is organized into the following sections:

- **Types.** Content definitions for web service operation input and output. See the topic "Types" for more information.
- **Messages.** Input and output definitions for the web service operations. See the topic "Messages" on page 4 for more information.
- **PortTypes.** Groups of operations offered by the web service. See the topic "Port types" on page 4 for more information.
- **Bindings.** Protocols and formats for the web service operations. See the topic "Bindings" on page 4 for more information.
- **Services.** Endpoints at which the web service functionality can be accessed. See the topic "Services" on page 5 for more information.

Types

The types element of a WSDL file contains the data type definitions employed by messages processed by the web service. These definitions use XML to organize the information relevant to the type element being defined. Consider the following example type definitions:

```
<wsdl:types>
  <schema targetNamespace="http://xml.spss.com/security/remote"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="getProviders">
      <complexType />
    </element>
    <element name="getProvidersResponse">
      <complexType>
```

```
        <sequence>
          <element name="providerInfo[unbounded]" type="tns1:providerInfo" />
        </sequence>
      </complexType>
    </element>
  </schema>
</wsdl:types>
```

This section defines two elements, *getProviders* and *getProvidersResponse*. The former is an empty element. The latter contains a sequence of *providerInfo* child elements. These children are all of the *providerInfo* type, which is defined elsewhere.

In practice, the WSDL file typically references type element definitions found in an external XML schema. For instance, the following definition uses `security-remote.xsd` to define type elements.

```
<wsdl:types>
  <xs:schema>
    <xs:import namespace="http://xml.spss.com/security/remote"
      schemaLocation="security-remote.xsd"/>
  </xs:schema>
</wsdl:types>
```

Messages

The message elements of a WSDL file defines the input or output parameters for operations available in the web service. Each message can consist of one or more parts, with the parts similar to the parameters of a function call in a traditional programming language. Consider the following two example message definitions:

```
<wsdl:message name="getProvidersResponse">
  <wsdl:part element="tns2:getProvidersResponse" name="parameters" />
</wsdl:message>
<wsdl:message name="getProvidersRequest">
  <wsdl:part element="tns2:getProviders" name="parameters" />
</wsdl:message>
```

The *getProvidersResponse* message contains a single part, corresponding to the *getProvidersResponse* element defined in the types section of the WSDL file. Similarly, the *getProvidersRequest* message also contains a single part, as defined by the *getProviders* element in the types section. See the topic "Types" on page 3 for more information.

Port types

The portType element of a WSDL file defines the actual interface to the web service. A port type is simply a group of related operations and is comparable to a function library, module, or class in a traditional programming language. The definition specifies the parameters for the operations, as well as any values returned. The parameters and return values correspond to messages defined elsewhere in the WSDL file. Consider the following example port type definition:

```
<wsdl:portType name="ProviderInformation">
  <wsdl:operation name="getProviders">
    <wsdl:input message="impl:getProvidersRequest" name="getProvidersRequest" />
    <wsdl:output message="impl:getProvidersResponse" name="getProvidersResponse" />
  </wsdl:operation>
</wsdl:portType>
```

The *ProviderInformation* port type consists of a single operation, *getProviders*. Input to this operation corresponds to the *getProvidersRequest* message. The operation returns information in the structure defined by the *getProvidersResponse* message. See the topic "Messages" for more information.

Bindings

The binding element of a WSDL file binds the interface defined by the port type to transport and messaging protocols. Consider the following example binding definition:

```
<wsdl:binding name="ProviderInformationSoapBinding" type="impl:ProviderInformation">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="getProviders">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="getProvidersRequest">
      <wsdlsoap:body namespace="http://xml.spss.com/security/remote" use="literal" />
    </wsdl:input>
    <wsdl:output name="getProvidersResponse">
```

```
      <wsdlsoap:body namespace="http://xml.spss.com/security" use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

In this case, the transport attribute of the `wsdlsoap:binding` element defines HTTP as the transport protocol. The `getProviders` operation in the interface is bound to the SOAP messaging protocol.

Services

The service element of a WSDL file identifies the network location at which the service interface can be accessed. Consider the following example service definition:

```
<wsdl:service name="ProviderInformationService">
  <wsdl:port binding="impl:ProviderInformationSoapBinding" name="ProviderInformation">
    <wsdlsoap:address location="http://pes_server:8080/security-ws/services/ProviderInformation" />
  </wsdl:port>
</wsdl:service>
```

In this example, the operations comprising the *ProviderInformation* port type can be accessed at:

```
http://pes_server:8080/security-ws/services/ProviderInformation
```

Proxies

Proxies serve as bridges between the client and the web service. A client-side proxy marshals the input objects into a standardized representation which is sent to the web service. A server-side proxy unmarshals the information into input objects for the service operations. The results of the operation are marshalled into standard representations and returned to the client. The client proxy unmarshals the response information into objects for any additional processing by the client.

Creating a proxy is the first step when developing a web service client; the proxy is the translation-unit between your application and the web service the application is using. Fortunately, many development environments include tools for automatically generating the client proxy from the web service WSDL file, allowing the client developer to focus on the client application code instead of transport and packaging protocols.

The proxy classes generated from a WSDL file depend on the tool used. For Java, the `wsdl2java` tool, which is part of the Apache Axis project, can be used. This tool produces a Java class for each type in the WSDL. Each port type results in a Java interface. A binding creates a stub class, and a WSDL service yields a service interface with a locator implementation. These generated classes and interfaces can be called directly from a client application written in Java to access the web service functionality.

An alternative Java proxy tool is `wsimport`, which is part of JAX-WS. The general structure of the generated classes is similar to that created by the Axis tool, but there are some differences. For example, instead of using arrays for input fields and returned items, the code generated from the `wsimport` tool uses `List` collections. In addition, if an input type matches an output type for a method, the `wsimport` tool uses a `Holder` class for the parameter.

In contrast, on the .NET platform, the `wsdl.exe` tool is often used to generate a web service proxy. This tool creates a single source file in a specified language containing the proxy class. This class includes both synchronous and asynchronous methods for each operation defined in the WSDL. For example, the web service operation `getProviders` results in the methods `getProviders`, `getProvidersBegin`, and `getProvidersEnd`. The latter two can be used for asynchronous processing.

A variety of other tools exist for other programming languages. For details, consult the documentation for those tools. In each case, the tool creates native programming constructs that permit leveraging a web service regardless of the service implementation language.

Chapter 2. Reporting Service overview

The Reporting Service allows a client to submit a predefined report for processing, such as a report created with BIRT Report Designer for IBM® SPSS®. Using information contained within the report, the client can validate input sources and create prompts for user input to direct report processing. Report output is available in a variety of formats for optimal display in any type of client.

Accessing the Reporting Service

To access the functionality offered by the Reporting Service, create a client application using the proxy classes generated by your preferred web service tool. The endpoint for the service is:

```
http://<host-name>:<port-number>/<context-root>/reporting-ws/services/Reporting
```

The value of *<host-name>* corresponds to the name or IP address of the machine on which IBM SPSS Collaboration and Deployment Services Repository is installed.

Note: An IPv6 address must be enclosed in square brackets, such as `[3ffe:2a00:100:7031::1]`.
The value of *<port-number>* indicates the port number on which the repository server is running. The *<context-root>* value specifies the custom context root, if any, configured for your repository server. If your system does not use a context root, omit this portion of the endpoint. To access the WSDL file for the service, append *?wsdl* to the service endpoint.

For example, if IBM SPSS Collaboration and Deployment Services Repository is running on port 80 of the machine *cads_server* without a context root, the WSDL file can be accessed using the path:

```
http://cads_server:80/reporting-ws/services/Reporting?wsdl
```

Calling Reporting Service operations

Clients access the operations offered by the web service using a stub for the service. The following is an example of how to acquire a stub in Java through Axis defined methods:

```
String context = "/reporting-ws/services/Reporting";
URL url = new URL("http", "cads_server", 80, context);
ReportingService service = new ReportingServiceLocator();
stub = service.getStatus(url);
```

The service operations can be called directly from the stub, such as:

```
stub.getReportMetadata(reportLocation);
```

Workflow

The process of generating a report from an input data source typically involves the following steps:

1. For a selected report, retrieve the metadata describing the report. See the topic “Reports” on page 9 for more information.
2. Validate that a connection can be made successfully to the data source defined in the report. If not, the connection properties should be modified as needed to make a successful connection. See the topic “Data sources” on page 9 for more information.
3. Prompt the user for any necessary report parameter values. See the topic “Prompt values” on page 12 for more information.
4. Render the report, providing status if wanted.
5. Retrieve the result.

Chapter 3. Reporting concepts

Reports

A report presents information contained within an input data source in a structured fashion that facilitates informed decision making. Metadata describing a report includes the following:

- Data sources. See the topic "Data sources" for more information.
- Data sets. See the topic "Data sets" for more information.
- Variable groups. See the topic "Variable groups" for more information.
- The MIME type and title for the report
- Columns used for generating burst reports
- The MIME types of any outputs generated by the report

The Reporting Service includes operations for retrieving all metadata for reports and for retrieving a subset of metadata.

Data sources

A data source defines the connection parameters for accessing the data on which a report is based. The data source is identified by name and consists of a list of properties and their values. The properties define the connection information and depend on the type of data source.

For example, JDBC sources require two properties: the driver class and the data source URL. The driver class property is `REPORT_DATASOURCE_DRIVER_CLASS`. The actual value for this property depends on the data source. For a MySQL source, the driver class might be `com.mysql.jdbc.Driver`. The URL property is `REPORT_DATASOURCE_URL`. Again, the property value depends on the data source. For the MySQL driver defined previously, it would be `jdbc:mysql://mydataserver:3306/mysource`.

Data sets

Data sets define the structure of the data retrieved from the data source on which reports are based. A data set is identified by its name and is characterized by the following information:

- A list of named columns in the data set with their types. For example, the first variable might be *id* having the *integer* type.
- The data source on which the data set is based. See the topic "Data sources" for more information.
- Optional named tables identifying column groups
- A SQL statement for extracting the data set from the data source
- Uniform resource identifiers for any linked sources

Variable groups

A variable group for a report specifies any parameters used in the report. The group is identified by name and consists of a list of report variables. Each variable is characterized by the following:

- A variable name
- A variable type, such as *INTEGER*, that determines formatting options for the parameter
- Optional selection values identifying valid values for the parameter as pairs of strings indicating the values and their descriptions
- Properties useful when prompting the user for a parameter value.

The following table lists a variety of typical prompt properties. The set of properties used for a particular parameter depends on the variable type.

Table 1. Prompt properties.

Property	Type	Description
acceptAnswersOnlyFromList	boolean	Indicates whether the parameter is open or closed
minLength	integer	Minimum length for the parameter value
maxLength	integer	Maximum length for the parameter value
minValue	string	Minimum value for the parameter
maxValue	string	Maximum value for the parameter
nullCapable	boolean	Indicates whether or not the parameter allows null values
allowBlank	boolean	Indicates whether or not the parameter allows blank values
skipIfHasValue	boolean	Indicates whether or not to prompt for a value if a value already exists
formatCategory	string	Format grouping. Valid values include Unformatted, General Date, Short Date, Medium Date, and Long Date.
formatPattern	string	Pattern for formatting the value
text	string	Label for the prompt
helpText	string	Instructional text often used to assist the user entering the parameter value
lastSavedValues	pair of strings	Most recent saved values for the parameter, if any
defaultValues	pair of strings	Default value(s) with optional description(s) for the parameter
defaultValueType	string	Default type for the value. Valid values include NONE, CURRENT_VALUE, and USER_SPECIFIED.
promptType	string	Determines the type for the prompt. Valid values include SINGLE_PROMPT, LIST_PROMPT, BETWEEN_PROMPT, BOOLEAN_PROMPT, and RADIO_PROMPT.
valueType	string	Identifies whether the parameters values are derived from a specified list (static) or generated (dynamic)
hasSelectionValues	boolean	Indicates whether the parameter has a list of valid values

Jobs

A job defines processing instructions for a report as a **render specification**. The specification defines the following:

- The report location. For reports stored in IBM SPSS Collaboration and Deployment Services Repository, the location is specified using the repository URI. See the topic "URI locations" for more information.
- Data sources on which the report is based with credentials for accessing them
- Variables/parameters for the report. For each parameter, the specification defines a name and value.
- The MIME type, encoding, format, and location for the report output
- The name of the data set used by the report
- Property values for rendering the report
- Linked sources, if any, for reports that respond to user actions such as a mouse click or mouse over event. The definition includes the name of the JavaScript function called when the event occurs.
- The length of time to allow for processing of the report
- The locale to use for generated results. The locale controls the formatting of numbers and dates for the report.

The Reporting Service includes operations for rendering jobs, updating status, retrieving results, and canceling jobs.

URI locations

Resources within the IBM SPSS Collaboration and Deployment Services Repository are often referenced using a uniform resource identifier. A content repository URI consists of the following items:

- The scheme *spsscr:*
- A hierarchical specification consisting of an authority definition and an optional object path
- An optional query specifying an object identifier
- Optional fragments defining version information

The URI has the following format:

```
spsscr://[host][:port]/[path/filename [?hierarchyType=type] | ?id=repositoryID][#l.label | #m.marker]
```

The hierarchical portion begins with two slashes, followed by the authority definition. This information identifies the host name and port number for the repository containing the object, followed by a slash. The authority definition may be omitted, in which case the URI indicates a relative location within the repository processing the service request.

```
spsscr:///[path/filename [?hierarchyType=type] | ?id=repositoryID][#l.label | #m.marker]
```

The URI continues with either the full path to the object, including its name, or a question mark and a query term consisting of the key *id*, an equals sign, and the repository resource identifier for the object. This identifier can be obtained from the information returned by the `getResource` operation of the Content Repository Service.

If the URI specifies an object path, the path may be followed by a query parameter designating the type of hierarchy containing the object. This parameter begins with a question mark, followed by the key *hierarchyType*, an equals sign, and the hierarchy type designator. Valid hierarchy types include *folder*, *topic*, *configuration*, *server*, *credential*, *datasource*, *enterprise*, and *submitted*. If the *hierarchyType* parameter is omitted, the *folder* hierarchy is used by default. The *hierarchyType* parameter is valid only when using the path to identify the object.

Optional version fragments follow the object information. The fragments begin with a hash symbol (#), followed by a single letter indicating whether the fragment is a version label (`l`) or a version timestamp marker (`m`). The fragment ends with a period and the actual label or marker for the version. Replace any spaces in the label or marker with escape characters. For example, the URI:

```
spsscr://myserver:80/marketing/campaign1#m.0:2006-10-08%2012:34:10.223
```

refers to the version of the *campaign1* job in the *marketing* folder saved at 12:34 on October 8, 2006. A URI that does not include a version fragment references the latest version of the object. For instance, the URI:

```
spsscr://localhost/campaign2
```

refers to the latest version of the job *campaign2*.

Prompt values

Report results may be dependent on user input. For example, the user may be able to specify a specific project or date range for processing. Typically this input is obtained using user prompts during the initial stages of report processing. Prompts may be open-ended (allowing any value to be input) or limited to a specific set of values from which the user must select. In the latter case, the set of valid values for a prompt can be retrieved before processing the report.

A request for a values set must specify the following:

- Report defining the prompt variable
- Name of the variable
- Login information for the data source on which the report is based

Some reports may include prompts with value sets that depend on other input prompts. For example, a prompt may limit the list of available projects to those active during a user-specified date range. The prompts **cascade**, with the value of one prompt influencing the values of another. In this case, the information request must include the names of the variables that cascade as well as the values being cascaded to the other prompts.

The Reporting Service includes operations for retrieving both standard and cascading prompt values.

Chapter 4. Operation reference

The cancelJob operation

Cancels the job, using the job's execution identifier. The job's execution ID is returned from the `renderJob` operation.

Input fields

The following table lists the input fields for the `cancelJob` operation.

Table 2. Fields for cancelJob.

Field	Type/Valid Values	Description
eventExecutionID	string	Execution ID of job. Returned from runJob.

Java example

To cancel the execution of a job, supply the `cancelJob` operation with a string corresponding to the identifier for the execution being canceled.

```
String executionID = new String;
executionID = "0a010a07b5551c63000001193e6e8a888059";
stub.cancelJob(executionID);
```

SOAP request example

Client invocation of the `cancelJob` operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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:Header>
    <wsse:Security soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <cancelJob xmlns="http://spss.com/reporting/ws/schema">
      <eventExecutionID>0a010a07b5551c63000001193e6e8a888059</eventExecutionID>
    </cancelJob>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `cancelJob` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
    <cancelJobResponse xmlns="http://spss.com/reporting/ws/schema"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The getReportMetadata operation

Returns properties of the report. Properties include the report data sources, variable groups, data sets, and output information.

Input fields

The following table lists the input fields for the `getReportMetadata` operation.

Table 3. Fields for getReportMetadata.

Field	Type/Valid Values	Description
reportLocation	locationType	Object indicating the report's location. Can be of type AttachmentLocation or URILocation.

Return information

The following table identifies the information returned by the `getReportMetadata` operation.

Table 4. Return Value.

Type	Description
reportMetadata	The report metadata which is returned.

Java example

To access the metadata associated with a report, supply the `getReportMetadata` operation with a `UriLocation` object corresponding to the report URI.

```
String uri = new String();
uri = "spsscr://pes_server:80/reports/xp.rptdesign#m.0:2008-04-08%2013:41:51.089";
UriLocation location = new UriLocation();
location.setUri(uri);
ReportMetadata md = stub.getReportMetadata(location);
```

SOAP request example

Client invocation of the `getReportMetadata` operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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:Header>
    <wsse:Security soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <getReportMetadata xmlns="http://spss.com/reporting/ws/schema">
      <reportLocation xsi:type="ns1:uriLocation" xmlns:ns1="http://spss.com/reporting/ws/schema">
        <ns1:uri xsi:type="xsd:anyURI"
          >spsscr://chikkroeger:8080/reports/xp.rptdesign#m.0:2008-04-08%2013:41:51.089</ns1:uri>
      </reportLocation>
    </getReportMetadata>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getReportMetadata` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
    <getReportMetadataResponse xmlns="http://spss.com/reporting/ws/schema">
      <reportMetadata>
        <reportDataSources>
          <name>XPlanner</name>
          <properties>
            <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
            <value>com.mysql.jdbc.Driver</value>
          </properties>
          <properties>
            <name>REPORT_DATASOURCE_URL</name>
            <value>jdbc:mysql://pubslinux:3306/xplanner</value>
          </properties>
          <isCredentialRequired>false</isCredentialRequired>
        </reportDataSources>
        <variableGroups xsi:type="ns1:defaultVariableGroup"
          xmlns:ns1="http://spss.com/reporting/ws/schema">
          <ns1:name>Default</ns1:name>
          <ns1:reportVariables>
            <ns1:variableName><ns1:name>ProjectID</ns1:name></ns1:variableName>
            <ns1:variableDataType>INTEGER</ns1:variableDataType>
            <ns1:promptProperties>
              <ns1:acceptAnswersOnlyFromList>true</ns1:acceptAnswersOnlyFromList>
              <ns1:nullCapable>false</ns1:nullCapable>
              <ns1:allowBlank>false</ns1:allowBlank>
              <ns1:formatCategory>Unformatted</ns1:formatCategory>
              <ns1:text>Project Name</ns1:text>
              <ns1:helpText>Select the project to analyze.</ns1:helpText>
              <ns1:defaultValues><ns1:value>66617</ns1:value></ns1:defaultValues>
              <ns1:defaultValueType>CURRENT_VALUE</ns1:defaultValueType>
              <ns1:promptType>SINGLE_PROMPT</ns1:promptType>
              <ns1:valueType>dynamic</ns1:valueType>
              <ns1:hasSelectionValues>true</ns1:hasSelectionValues>
            </ns1:promptProperties>
          </ns1:reportVariables>
        </variableGroups>
        <reportResource>
          <mimeType>application/vnd.birt-rptdesign</mimeType>
          <title>xp.rptdesign</title>
        </reportResource>
        <dataSets>
          <name>Project</name>
          <selectedColumns>
            <name>id</name><description>id</description><type>integer</type>
          </selectedColumns>
          <selectedColumns>
            <name>last_update</name><description>last_update</description><type>date-time</type>
          </selectedColumns>
          <selectedColumns>
            <name>name</name><description>name</description><type>string</type>
          </selectedColumns>
          <selectedColumns>
            <name>description</name><description>description</description><type>string</type>
          </selectedColumns>
          <selectedColumns>
            <name>is_hidden</name><description>is_hidden</description><type>integer</type>
          </selectedColumns>
          <dataSource>
            <name>XPlanner</name>
            <properties>
              <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
              <value>com.mysql.jdbc.Driver</value>
            </properties>
            <properties>
              <name>REPORT_DATASOURCE_URL</name>
              <value>jdbc:mysql://pubslinux:3306/xplanner</value>
            </properties>
            <isCredentialRequired>false</isCredentialRequired>
          </dataSource>
          <sqlStmt>select * from project</sqlStmt>
        </dataSets>
        <dataSets>
          <name>Iteration</name>
          <selectedColumns>
```

```
    <name>id</name><description>id</description><type>integer</type>
  </selectedColumns>
  <selectedColumns>
    <name>last_update</name><description>last_update</description><type>date-time</type>
  </selectedColumns>
  <selectedColumns>
    <name>project_id</name><description>project_id</description><type>integer</type>
  </selectedColumns>
  <selectedColumns>
    <name>name</name><description>name</description><type>string</type>
  </selectedColumns>
  <selectedColumns>
    <name>description</name><description>description</description><type>string</type>
  </selectedColumns>
  <selectedColumns>
    <name>start_date</name><description>start_date</description><type>date</type>
  </selectedColumns>
  <selectedColumns>
    <name>end_date</name><description>end_date</description><type>date</type>
  </selectedColumns>
  <selectedColumns>
    <name>status</name><description>status</description><type>integer</type>
  </selectedColumns>
  <selectedColumns>
    <name>days_worked</name><description>days_worked</description><type>float</type>
  </selectedColumns>
  <selectedColumns>
    <name>orig_iteration_id</name><description>orig_iteration_id</description>
    <type>integer</type>
  </selectedColumns>
  <dataSource>
    <name>XPlanner</name>
    <properties>
      <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
      <value>com.mysql.jdbc.Driver</value>
    </properties>
    <properties>
      <name>REPORT_DATASOURCE_URL</name>
      <value>jdbc:mysql://pubslinux:3306/xplanner</value>
    </properties>
    <isCredentialRequired>false</isCredentialRequired>
  </dataSource>
  <sqlStmt>select * from iteration</sqlStmt>
</dataSets>
<dataSets>
  <name>AllStories</name>
  <selectedColumns>
    <name>id</name><description>id</description><type>integer</type>
  </selectedColumns>
  <selectedColumns>
    <name>last_update</name><description>last_update</description><type>date-time</type>
  </selectedColumns>
  <selectedColumns>
    <name>name</name><description>name</description><type>string</type>
  </selectedColumns>
  <selectedColumns>
    <name>description</name><description>description</description><type>string</type>
  </selectedColumns>
  <selectedColumns>
    <name>iteration_id</name><description>iteration_id</description><type>integer</type>
  </selectedColumns>
  <selectedColumns>
    <name>tracker_id</name><description>tracker_id</description><type>integer</type>
  </selectedColumns>
  <selectedColumns>
    <name>estimated_hours</name><description>estimated_hours</description><type>float</type>
  </selectedColumns>
  <selectedColumns>
    <name>priority</name><description>priority</description><type>integer</type>
  </selectedColumns>
  <selectedColumns>
    <name>customer_id</name><description>customer_id</description><type>integer</type>
  </selectedColumns>
  <selectedColumns>
    <name>status</name><description>status</description><type>string</type>
  </selectedColumns>
  <selectedColumns>
    <name>original_estimated_hours</name><description>original_estimated_hours</description>
    <type>float</type>
  </selectedColumns>
  <selectedColumns>
    <name>disposition</name><description>disposition</description><type>string</type>
  </selectedColumns>
  <selectedColumns>
```

```
          <name>postponed_hours</name><description>postponed_hours</description><type>float</type>
        </selectedColumns>
        <selectedColumns>
          <name>it_start_estimated_hours</name><description>it_start_estimated_hours</description>
          <type>float</type>
        </selectedColumns>
        <selectedColumns>
          <name>orderNo</name><description>orderNo</description><type>integer</type>
        </selectedColumns>
        <selectedColumns>
          <name>project</name><description>project</description><type>string</type>
        </selectedColumns>
        <selectedColumns>
          <name>eng_story</name><description>eng_story</description><type>integer</type>
        </selectedColumns>
        <selectedColumns>
          <name>prd_number</name><description>prd_number</description><type>string</type>
        </selectedColumns>
        <selectedColumns>
          <name>tracker_name</name><description>tracker_name</description><type>string</type>
        </selectedColumns>
        <dataSource>
          <name>XPlanner</name>
          <properties>
            <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
            <value>com.mysql.jdbc.Driver</value>
          </properties>
          <properties>
            <name>REPORT_DATASOURCE_URL</name>
            <value>jdbc:mysql://pubslinux:3306/xplanner</value>
          </properties>
          <isCredentialRequired>false</isCredentialRequired>
        </dataSource>
        <sqlStmt>select * from story</sqlStmt>
      </dataSets>
    </reportMetadata>
  </getReportMetadataResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

The getSelectedReportMetadata operation

Returns specific report information. One or more of the following WSDLConstants can be used as a parameter:

- METADATA_DATASOURCE for data sources
- METADATA_DATASET for data sets
- METADATA_DATASET_COL for data set columns
- METADATA_VARIABLE for variables
- METADATA_BURST for burst columns
- METADATA_OUTPUTTYPES for output types

Input fields

The following table lists the input fields for the `getSelectedReportMetadata` operation.

Table 5. Fields for getSelectedReportMetadata.

Field	Type/Valid Values	Description
reportLocation	locationType	Object indicating the report's location. Can be of type AttachmentLocation or URILocation.
selectedMetadata	string[]	The type of report metadata which is requested. See WSDLConstants for types of metadata which can be requested.

Return information

The following table identifies the information returned by the `getSelectedReportMetadata` operation.

Table 6. Return Value.

Type	Description
reportMetadata	The specific report metadata which is returned.

Java example

To access specific metadata for a report:

1. Create a `UriLocation` object corresponding to the report URI.
2. Create a string containing the name of the metadata to return. If multiple fields are needed, use a string array.
3. Supply the `getSelectedReportMetadata` operation with the URI and metadata string.

The following sample code returns the data sources metadata for a report.

```
String uri = new String();
uri = "spsscr://pes_server:80/reports/cascade.rptdesign#1.LATEST";
UriLocation location = new UriLocation();
location.setUri(uri);
String selection = new String();
selection = "DataSources";
ReportMetadata md = stub.getSelectedReportMetadata(location, selection);
```

SOAP request example

Client invocation of the `getSelectedReportMetadata` operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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:Header>
    <wsse:Security soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsse:Username>Native/validUser</wsse:Username>
        <wsse:Password
          wsse:Type="http://docs.oasis-open.org/wss/2004/01/
             oasis-200401-wss-username-token-profile-1.0#PasswordText"
          >password</wsse:Password>
        <wsse:Nonce>bpl0mpZvSxWqphlGozvEGg==</wsse:Nonce>
        <wsu:Created
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          >2008-04-14T18:26:03Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <getSelectedReportMetadata xmlns="http://spss.com/reporting/ws/schema">
      <reportLocation xsi:type="ns1:uriLocation" xmlns:ns1="http://spss.com/reporting/ws/schema">
        <ns1:uri xsi:type="xsd:anyURI"
        >spsscr://pes_server:80/reports/cascade.rptdesign#1.LATEST</ns1:uri>
      </reportLocation>
      <selectedMetadata>DataSources</selectedMetadata>
    </getSelectedReportMetadata>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getSelectedReportMetadata` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
    <getSelectedReportMetadataResponse xmlns="http://spss.com/reporting/ws/schema">
      <reportMetadata>
        <reportDataSources>
          <name>XPlanner</name>
          <properties>
            <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
            <value>com.mysql.jdbc.Driver</value>
          </properties>
          <properties>
            <name>REPORT_DATASOURCE_URL</name>
            <value>jdbc:mysql://pubslinux:3306/xplanner</value>
          </properties>
          <isCredentialRequired>false</isCredentialRequired>
        </reportDataSources>
        <reportResource>
          <mimeType>application/vnd.birt-rptdesign</mimeType>
          <title>cascade.rptdesign</title>
        </reportResource>
      </reportMetadata>
    </getSelectedReportMetadataResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getVersion operation

Returns the version number of the service.

Return information

The following table identifies the information returned by the `getVersion` operation.

Table 7. Return Value.

Type	Description
string	The service version number.

Java example

To access the version number of the service, call the `getVersion` operation from the service stub.

```
System.out.println("Service Version =  " + stub.getVersion());
```

SOAP request example

Client invocation of the `getVersion` operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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>
      <getVersion xmlns="http://spss.com/reporting/ws/schema"/>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getVersion` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
        <getVersionResponse xmlns="http://spss.com/reporting/ws/schema">
```

```
            <version>4.20.000</version>
        </getVersionResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

The renderJob operation

This operation is deprecated. Consumers of the web service wanting to run jobs should use the `runJob` operation instead.

Input fields

The following table lists the input fields for the `renderJob` operation.

Table 8. Fields for renderJob.

Field	Type/Valid Values	Description
renderSpec	renderSpecification	

Return information

The following table identifies the information returned by the `renderJob` operation.

Table 9. Return Value.

Type	Description
string	Job ID returned from renderJob.

The renderSync operation

Runs the report synchronously, according to the properties specified in the render specification. One or more of the following WSDLConstants can be set as properties:

- RENDERPROPERTY_SAVE_COLUMN_HEADINGS_WITH_DATA
- RENDERPROPERTY_APPLY_EDIT_MASK_TO_DATA
- RENDERPROPERTY_OUTPUT_REPORT_DETAILS_ONLY
- RENDERPROPERTY_INCLUDE_SUMMARY_TEXT
- RENDERPROPERTY_INCLUDE_HTML_HEADER
- RENDERPROPERTY_SEND_TO_PRINTER
- RENDERPROPERTY_BURST_COLUMN

The report results are returned.

Input fields

The following table lists the input fields for the `renderSync` operation.

Table 10. Fields for renderSync.

Field	Type/Valid Values	Description
renderSpec	renderSpecification	

Return information

The following table identifies the information returned by the renderSync operation.

Table 11. Return Value.

Type	Description
renderedResult	Contains render details of the report. Includes executionDetails and output location.

Java example

Rendering a job requires the creation of a render specification. See the topic "Jobs" on page 10 for more information. To define the specification and render a job:

1. Create a RenderSpecification object.
2. Create a UriLocation object and use the setUri method to assign a string corresponding to the URI for the report. Use the setReportLocation method to assign the location object to the specification.
3. Create a DataSourceLogin object for the data source and credential information.
4. Create a DataSource object and use the setName method to define the name of the data source.
5. Create an array of Property objects to define the data source connection information. For each property, use the setName method to define the name and the setValue method to define the property value. Use the setProperties method to assign the property array to the data source object.
6. Supply the setIsCredentialRequired method with a boolean indicating whether or not credentials are required.
7. Use the setDataSource method to assign the data source to the login object.
8. Create a Credential object for the data source credentials. Use the setUsername method to assign a string corresponding to the user name for the credential. Use the setPassword method to assign a string corresponding to the password for the credential.
9. Use the setCredential method to assign the credential to the login object.
10. Use the setDataSources method to assign the login object to the specification.
11. Create a VariableValue object to define any variables used in the report. Use an array for multiple variables.
12. Create a VariableName object and use the setName method to specify a string corresponding to the name of the variable. Use the setVariableName method to assign the name object to the values object.
13. Create a ValueDescription object and supply strings corresponding to the value and its description. Use the setValues method to assign the description to the values object.
14. Use the setVariables method to assign the values object to the specification.
15. Create an OutputSpecification object to define the output characteristics. Use the setMimeType and setEncoding methods to define the MIME type and encoding.
16. Create a RepositoryOutput object.
17. Use the setViewKey method to specify a string corresponding to a unique view key for the report.
18. Create a URI object and specify a string corresponding to the URI for the report. Use the setUri method to assign the URI object to the output object.
19. Define the repository hierarchy using the setHierarchy method.
20. Define the author for report using the setAuthor method.
21. To set an expiration date for the report, create a Calendar object and specify the expiration date and time. Use the setExpirationDate method to assign the date to the output object.
22. Use the setOutputLocation method to assign the output object to the output specification.
23. Use the setOutputs method to assign the output object to the render specification.

24. Create a `Property` object to define properties for report rendering. Supply the name and value for the property. Use the `setRenderProperties` method to assign the property to the render specification.
25. Use the `setTimeout` and `setLocale` methods to define the timeout value and locale.
26. Supply the `renderSync` operation with the render specification.

The following sample creates an HTML report for a JDBC data source.

```
RenderSpecification rSpec = new RenderSpecification();

String uri = new String();
uri = "spsscr://pes_server:80/reports/xp.rptdesign#m.0:2008-04-08%2013:41:51.089";
UriLocation location = new UriLocation();
location.setUri(uri);
rSpec.setReportLocation

DataSourceLogin dsLogin = new DataSourceLogin();
DataSource ds = new DataSource();
ds.setName("XPlanner");
Property [] prop = new Property[2];
prop[0].setName("REPORT_DATASOURCE_DRIVER_CLASS");
prop[0].setValue("com.mysql.jdbc.Driver");
prop[1].setName("REPORT_DATASOURCE_URL");
prop[1].setValue("jdbc:mysql://mylinux:3306/xplanner");
ds.setProperties(prop);
ds.setIsCredentialRequired(false);
Credential cred = new Credential();
cred.setUsername("xpuser");
cred.setPassword("xppass");
dsLogin.setDataSource(ds);
dsLogin.setCredential(cred);
rSpec.setDataSources(dsLogin);

VariableValue vValues = new VariableValue();
VariableName vName = new VariableName();
vName.setName("ProjectID");
vValues.setVariableName(vName);
ValueDescription vDesc = new ValueDescription("66617", "66617");
vValues.setValues(vDesc);
rSpec.setVariables(vValues);

OutputSpecification oSpec = new OutputSpecification();
oSpec.setMimeType("text/html");
oSpec.setEncoding("utf8");
RepositoryOutput  rOutput = new RepositoryOutput();
rOutput.setViewKey("0a010a07b5551c63000001193e6e8a888039");
URI myUri = new URI("spsscr://pes_server:80/validUser/2008-04-11.12.12.45.334-xp.rptdesign/xp.rptdesign.html");
rOutput.setUri(myUri);
rOutput.setHierarchy("submitted");
rOutput.setAuthor("validUser");
Calendar expirationDate = new Calendar();
expirationDate.set(2008, 04, 16, 17, 12, 329);
rOutput.setExpirationDate(expirationDate);
oSpec.setOutputLocation(rOutput);
rSpec.setOutputs(oSpec);

Property prop = new Property(RENDERPROPERTY_REPORT_MODE, REPORT_MODE_SUBMITTED);
rSpec.setRenderProperties(prop);

long lg = 240000;
Long tOut = new Long(lg);
rSpec.setTimeout(tOut);
rSpec.setLocale("en_US");

RenderedResult rResult = stub.renderSync(rSpec);
```

SOAP request example

Client invocation of the `renderSync` operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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:Header>
    <wsse:Security
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:spsssec="http://xml.spss.com/security"
      soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
```

```
      <wsse:BinarySecurityToken wsu:Id="spssToken" ValueType="spsssec:BinarySecurityToken"
        EncodingType="wsse:Base64Binary"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
        >rO0ABXNyAB5jb20uc3Bzcy5zZWN1cml0eS5jb3JlLlN1YmplY3QAAAAAAAAAAQIAAlsAC21fc2
        lnbmF0dXJldAACW0JMAA1tX3N1YmplY3REYXRhdAAtTGNvbS9zcHNzL3NlY3VyaXR5L2NvcmUva
        W50ZXJuYWwvU3ViamVjdERhdGE7eHBwc3IAK2NvbS5zcHNzLnNlY3VyaXR5LmNvcmUuaW50ZXJu
        YWwuU3ViamVjdERhdGEAAAAAAAAAAQIAB1oACW1faXNBZG1pbkkAEG1fcHJpbmNpcGxlSW5kZXh
        KAA5tX3NlcmlhbE51bWJlckwACW1fYWRkcmVzc3QAEkxqYXZhL2xhbmcvU3RyaW5nO0wAE21fYW
        5ub3RhdGlvbk1lc3NhZ2VxAH4ABUwADm1fcHJpbmNpcGFsSURzdAAQTGphdmEvdXRpbC9MaXN0O
        0wACG1fdXNlcklEcQB+AAV4cAEAAAAAAAAABGT53Xhp0AAkxMjcuMC4wLjFwc3IAJmphdmEudXRp
        bC5Db2xsZWN0aW9ucyRVbm1vZGlmaWFibGVMaXN0/A8lMbXsjhACAAFMAARsaXN0cQB+AAZ4cgA
        samF2YS51dGlsLkNvbGxlY3Rpb25zJFVubW9kaWZpYWJsZUNvbGxlY3Rpb24ZQgCAy173HgIAAU
        wAAWN0ABZMamF2YS91dGlsL0NvbGxlY3Rpb247eHBzcgATamF2YS51dGlsLkFycmF5TGlzdHiB0
        h2Zx2GdAwABSQAEc2l6ZXhwAAAAA3cEAAAA3QAEC8vdU5hdGl2ZS8vYWRtaW50ACMvL2dOYXRp
        dmUvLyQkc2VjdXJpdHkvZXZlcnlvbmVHcm91cHQAKC8vck5hdGl2ZS8vJCRzZWN1cml0eS9yb2x
        lQWRtaW5pc3RyYXRvcnN4cQB+AA50AAVhZG1pbg==</wsse:BinarySecurityToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <renderSync xmlns="http://spss.com/reporting/ws/schema">
      <renderSpec>
        <reportLocation xsi:type="ns1:uriLocation" xmlns:ns1="http://spss.com/reporting/ws/schema">
          <ns1:uri>spsscr://pes_server:80/reports/xp.rptdesign#m.0:2008-04-08%2013:41:51.089</ns1:uri>
        </reportLocation>
        <dataSources>
          <dataSource>
            <name>XPlanner</name>
            <properties>
              <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
              <value>com.mysql.jdbc.Driver</value>
            </properties>
            <properties>
              <name>REPORT_DATASOURCE_URL</name>
              <value>jdbc:mysql://mylinux:3306/xplanner</value>
            </properties>
            <isCredentialRequired>false</isCredentialRequired>
          </dataSource>
          <credential xsi:type="ns2:credential" xmlns:ns2="http://spss.com/reporting/ws/schema">
            <ns2:username>xpuser</ns2:username>
            <ns2:password>xppass</ns2:password>
          </credential>
        </dataSources>
        <variables>
          <variableName>
            <name>ProjectID</name>
          </variableName>
          <values>
            <value>66617</value>
            <description>66617</description>
          </values>
        </variables>
        <outputs>
          <mimeType>text/html</mimeType>
          <encoding>utf8</encoding>
          <outputLocation xsi:type="ns3:repositoryOutput"
            xmlns:ns3="http://spss.com/reporting/ws/schema">
            <ns3:viewKey>0a010a07b5551c63000001193e6e8a888039</ns3:viewKey>
            <ns3:uri>spsscr://pes_server:80/validUser/2008-04-11.12.12.45.334-xp.rptdesign/xp.rptdesign.html</ns3:uri>
            <ns3:hierarchy>submitted</ns3:hierarchy>
            <ns3:author>validUser</ns3:author>
            <ns3:expirationDate>2008-04-16T17:12:53.329Z</ns3:expirationDate>
          </outputLocation>
        </outputs>
        <renderProperties>
          <name>RENDERPROPERTY_REPORT_MODE</name>
          <value>REPORT_MODE_SUBMITTED</value>
        </renderProperties>
        <timeout>240000</timeout>
        <locale>en_US</locale>
      </renderSpec>
    </renderSync>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a renderSync operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
    <renderSyncResponse xmlns="http://spss.com/reporting/ws/schema">
      <renderedResult>
        <resultLocations xsi:type="ns1:uriLocation" xmlns:ns1="http://spss.com/reporting/ws/schema">
          <ns1:mimeType>multipart/related</ns1:mimeType>
          <ns1:uri>spsscr:///?id=0a010a07b5551c63000001193e6e8a888042#m.0:2008-04-11%2012:13:34.663</ns1:uri>
        </resultLocations>
      </renderedResult>
    </renderSyncResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The retrieveCascadingPromptValues operation

Returns a set of valid prompt values for input parameters that depend on the values of other parameters.

Input fields

The following table lists the input fields for the `retrieveCascadingPromptValues` operation.

Table 12. Fields for retrieveCascadingPromptValues.

Field	Type/Valid Values	Description
reportLocation	locationType	Object indicating the report's location. Can be of type AttachmentLocation or URILocation.
variableGroupName	string	String id for the variable group name.
groupKeyValues	string[]	String array of specific values requested.
dataSourceLogins	dataSourceLogin[]	Array of DataSourceLogins, where data source and credential information is set.

Return information

The following table identifies the information returned by the `retrieveCascadingPromptValues` operation.

Table 13. Return Value.

Type	Description
valueDescription[]	Array returned from retrieveCascadingPromptValues containing the requested prompt values.

Java example

To retrieve cascading prompt values for a report:

1. Create a `UriLocation` object and use the `setUri` method to assign a string corresponding to the URI for the report.
2. Create a `DataSourceLogin` object for the data source and credential information.
3. Create a `DataSource` object and use the `setName` method to define the name of the data source.
4. Create an array of `Property` objects to define the data source connection information. For each property, use the `setName` method to define the name and the `setValue` method to define the property value. Use the `setProperties` method to assign the property array to the data source object.
5. Supply the `setIsCredentialRequired` method with a boolean indicating whether or not credentials are required.
6. Use the `setDataSource` method to assign the data source to the login object.

7. Create a `Credential` object for the data source credentials. Use the `setUsername` method to assign a string corresponding to the user name for the credential. Use the `setPassword` method to assign a string corresponding to the password for the credential.
8. Use the `setCredential` method to assign the credential to the login object.
9. Supply the `retrievePromptValues` operation with the location, a string corresponding to the name of the cascading variable, a string identifying the value being cascaded, and login objects.

```
String uri = new String();
uri = "spsscr://pes_server:80/reports/cascade.rptdesign";
UriLocation location = new UriLocation();
location.setUri(uri);

String grpName=new String("NewCascadingParameterGroup");
String keyValues=new String("66617");

DataSourceLogin dsLogin = new DataSourceLogin();
DataSource ds = new DataSource();
ds.setName("XPlanner");
Property [] prop = new Property[2];
prop[0].setName("REPORT_DATASOURCE_DRIVER_CLASS");
prop[0].setValue("com.mysql.jdbc.Driver");
prop[1].setName("REPORT_DATASOURCE_URL");
prop[1].setValue("jdbc:mysql://mylinux:3306/xplanner");
ds.setProperties(prop);
ds.setIsCredentialRequired(false);
Credential cred = new Credential();
cred.setUsername("xpuser");
cred.setPassword("xppass");
dsLogin.setDataSource(ds);
dsLogin.setCredential(cred);

ValueDescription[] vDesc = stub.retrieveCascadingPromptValues(location, grpName, keyValues, dsLogin);

for (int j = 0; j < vDesc.length; j++) {
   System.out.println(vDesc[j].getValue() + " with a description of " +
      vDesc[j].getDescription());
}
```

SOAP request example

Client invocation of the `retrieveCascadingPromptValues` operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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:Header>
    <wsse:Security soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
    <soapenv:Body>
        <retrieveCascadingPromptValues xmlns="http://spss.com/reporting/ws/schema">
            <reportLocation xsi:type="ns1:uriLocation"
                xmlns:ns1="http://spss.com/reporting/ws/schema">
                <ns1:uri xsi:type="xsd:anyURI"
                >spsscr://pes_server:80/reports/cascade.rptdesign</ns1:uri>
            </reportLocation>
            <variableGroupName>NewCascadingParameterGroup</variableGroupName>
            <groupKeyValues>66617</groupKeyValues>
            <dataSourceLogins>
                <dataSource>
                    <name>XPlanner</name>
                    <properties>
                        <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
                        <value>com.mysql.jdbc.Driver</value>
                    </properties>
                    <properties>
                        <name>REPORT_DATASOURCE_URL</name>
                        <value>jdbc:mysql://mylinux:3306/xplanner</value>
                    </properties>
                    <isCredentialRequired>false</isCredentialRequired>
                </dataSource>
                <credential xsi:type="ns2:credential"
                    xmlns:ns2="http://spss.com/reporting/ws/schema">
```

```
                    <ns2:username>xpuser</ns2:username>
                    <ns2:password>xppass</ns2:password>
                </credential>
            </dataSourceLogins>
        </retrieveCascadingPromptValues>
    </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `retrieveCascadingPromptValues` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
        <retrieveCascadingPromptValuesResponse xmlns="http://spss.com/reporting/ws/schema"/>
    </soapenv:Body>
</soapenv:Envelope>
```

The retrieveJobResult operation

Retrieves the job result, using the job's executionID returned from the `renderJob` operation. The job result contains the job status and the completed report.

Input fields

The following table lists the input fields for the `retrieveJobResult` operation.

Table 14. Fields for retrieveJobResult.

Field	Type/Valid Values	Description
eventExecutionID	string	Job ID returned from runJob.

Return information

The following table identifies the information returned by the `retrieveJobResult` operation.

Table 15. Return Value.

Type	Description
jobResult	JobResult contains the RenderedResult and the JobStatus.

Java example

To return the results of a running a report, supply the `retrieveJobResult` operation with a string corresponding to the report execution identifier.

```
String executionID = new String;
executionID = "0a010a07b5551c63000001193e6e8a888039";
JobResult result = stub.retrieveJobResult(executionID);
System.out.println("Status = " + result.getJobStatus.getStatus());
```

The returned object contains information about the status of the execution as well as the location of the results.

SOAP request example

Client invocation of the `retrieveJobResult` operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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:Header>
    <wsse:Security soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <retrieveJobResult xmlns="http://spss.com/reporting/ws/schema">
      <eventExecutionID>0a010a07b5551c63000001193e6e8a888039</eventExecutionID>
    </retrieveJobResult>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `retrieveJobResult` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
    <retrieveJobResultResponse xmlns="http://spss.com/reporting/ws/schema">
      <jobResult>
        <renderedResult>
          <resultLocation xsi:type="ns1:uriLocation"
             xmlns:ns1="http://spss.com/reporting/ws/schema">
            <ns1:mimeType>multipart/related</ns1:mimeType>
            <ns1:uri>http://pes_server:80/reporting-ws/reporting_viewer?
               mimeType=multipart/related&amp;eventID=0a010a07b5551c63
               000001193e6e8a888039&amp;charset=UTF-8&amp;dummyParm=foo.mht
            </ns1:uri>
          </resultLocation>
          <resultLocations xsi:type="ns2:uriLocation"
            xmlns:ns2="http://spss.com/reporting/ws/schema">
            <ns2:mimeType>multipart/related</ns2:mimeType>
            <ns2:uri>http://pes_server:80/reporting-ws/reporting_viewer?
               mimeType=multipart/related&amp;eventID=0a010a07b5551c63
               000001193e6e8a888039&amp;charset=UTF-8&amp;dummyParm=foo.mht
            </ns2:uri>
          </resultLocations>
        </renderedResult>
        <jobStatus>
          <status>SUCCESS</status>
        </jobStatus>
      </jobResult>
    </retrieveJobResultResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The retrievePromptValues operation

Returns a set of valid prompt values for a specified input parameter.

Input fields

The following table lists the input fields for the `retrievePromptValues` operation.

Table 16. Fields for retrievePromptValues.

Field	Type/Valid Values	Description
reportLocation	locationType	Object indicating the report's location. Can be of type AttachmentLocation or URILocation.
variableNames	variableName[]	VariableName contains the name and scope of the variable.

Table 16. Fields for retrievePromptValues (continued).

Field	Type/Valid Values	Description
dataSourceLogins	dataSourceLogin[]	Array of DataSourceLogins, where data source and credential information is set.

Return information

The following table identifies the information returned by the `retrievePromptValues` operation.

Table 17. Return Value.

Type	Description
variableValue[]	Array of variable values returned from retreivePromptValues.

Java example

To retrieve the prompt values for a report:

1. Create a `UriLocation` object and use the `setUri` method to assign a string corresponding to the URI for the report.
2. Create a `VariableName` object for the prompt variable. Use the `setName` method to assign a string corresponding to the name of the variable.
3. Create a `DataSourceLogin` object for the data source and credential information.
4. Create a `DataSource` object and use the `setName` method to define the name of the data source.
5. Create an array of `Property` objects to define the data source connection information. For each property, use the `setName` method to define the name and the `setValue` method to define the property value. Use the `setProperties` method to assign the property array to the data source object.
6. Supply the `setIsCredentialRequired` method with a boolean indicating whether or not credentials are required.
7. Use the `setDataSource` method to assign the data source to the login object.
8. Create a `Credential` object for the data source credentials. Use the `setUsername` method to assign a string corresponding to the user name for the credential. Use the `setPassword` method to assign a string corresponding to the password for the credential.
9. Use the `setCredential` method to assign the credential to the login object.
10. Supply the `retrievePromptValues` operation with the location, variable, and login objects.

```
String uri = new String();
uri = "spsscr://pes_server:80/reports/xp.rptdesign#m.0:2008-04-08%2013:41:51.089";
UriLocation location = new UriLocation();
location.setUri(uri);

VariableName variable = new VariableName();
variable.setName("ProjectID");

DataSourceLogin dsLogin = new DataSourceLogin();
DataSource ds = new DataSource();
ds.setName("XPlanner");
Property [] prop = new Property[2];
prop[0].setName("REPORT_DATASOURCE_DRIVER_CLASS");
prop[0].setValue("com.mysql.jdbc.Driver");
prop[1].setName("REPORT_DATASOURCE_URL");
prop[1].setValue("jdbc:mysql://mylinux:3306/xplanner");
ds.setProperties(prop);
ds.setIsCredentialRequired(false);
Credential cred = new Credential();
cred.setUsername("xpuser");
cred.setPassword("xppass");
dsLogin.setDataSource(ds);
dsLogin.setCredential(cred);
```

```
VariableValue[] vValues = stub.retrievePromptValues(location, variable, dsLogin);

for (int i = 0; i < vValues.length; i++) {
   System.out.println(vValues[i].getVariableName().getName() +
      " has the following values:");
   ValueDescription[] desc = vValues[i].getValues();
   for (int j = 0; j < desc.length; j++) {
      System.out.println(desc[j].getValue() + " with a description of " +
         desc[j].getDescription());
   }
}
```

SOAP request example

Client invocation of the `retrievePromptValues` operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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:Header>
    <wsse:Security soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <retrievePromptValues xmlns="http://spss.com/reporting/ws/schema">
      <reportLocation xsi:type="ns1:uriLocation" xmlns:ns1="http://spss.com/reporting/ws/schema">
        <ns1:uri xsi:type="xsd:anyURI"
          >spsscr://pes_server:80/reports/xp.rptdesign#m.0:2008-04-08%2013:41:51.089</ns1:uri>
      </reportLocation>
      <variableNames>
        <name>ProjectID</name>
      </variableNames>
      <dataSourceLogins>
        <dataSource>
          <name>XPlanner</name>
          <properties>
            <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
            <value>com.mysql.jdbc.Driver</value>
          </properties>
          <properties>
            <name>REPORT_DATASOURCE_URL</name>
            <value>jdbc:mysql://mylinux:3306/xplanner</value>
          </properties>
          <isCredentialRequired>false</isCredentialRequired>
        </dataSource>
        <credential xsi:type="ns2:credential" xmlns:ns2="http://spss.com/reporting/ws/schema">
          <ns2:username>xpuser</ns2:username>
          <ns2:password>xppass</ns2:password>
        </credential>
      </dataSourceLogins>
    </retrievePromptValues>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `retrievePromptValues` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
    <retrievePromptValuesResponse xmlns="http://spss.com/reporting/ws/schema">
      <variableValues>
        <variableName>
          <name>ProjectID</name>
        </variableName>
        <values>
          <value>221</value><description>Project 1</description>
        </values>
        <values>
          <value>222</value><description>Project 2</description>
```

```
        </values>
        <values>
          <value>66617</value><description>Project 3</description>
        </values>
        <values>
          <value>71974</value><description>Project 4</description>
        </values>
      </variableValues>
    </retrievePromptValuesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The runJob operation

Runs the job asynchronously, according to the properties specified in the render specification. One or more of the following WSDLConstants properties can be set:

- RENDERPROPERTY_SAVE_COLUMN_HEADINGS_WITH_DATA
- RENDERPROPERTY_APPLY_EDIT_MASK_TO_DATA
- RENDERPROPERTY_OUTPUT_REPORT_DETAILS_ONLY
- RENDERPROPERTY_INCLUDE_SUMMARY_TEXT
- RENDERPROPERTY_INCLUDE_HTML_HEADER
- RENDERPROPERTY_SEND_TO_PRINTER
- RENDERPROPERTY_BURST_COLUMN
- RENDERPROPERTY_REPORT_MODE
- RENDERPROPERTY_MIME_HTML_WITH_IMAGES

The job returns an executionID which can be used to retrieve the job result and a URI prefix that should be used when accessing the service to retrieve the result or cancel the job.

Input fields

The following table lists the input fields for the `runJob` operation.

Table 18. Fields for runJob.

Field	Type/Valid Values	Description
renderSpec	renderSpecification	

Return information

The following table identifies the information returned by the `runJob` operation.

Table 19. Return Value.

Type	Description
runJobResult	

Java example

Running a job requires the creation of a render specification. See the topic "The renderJob operation" on page 20 for more information.

Supply the `runJob` operation with the render specification to execute the job.

```
RenderSpecification rSpec = new RenderSpecification();

String uri = new String();
uri = "spsscr:///?id=0a010a074a2f7bcc000001201b05e1d980c9";
UriLocation location = new UriLocation();
```

```
location.setUri(uri);
rSpec.setReportLocation(location);

DataSourceLogin dsLogin = new DataSourceLogin();
DataSource ds = new DataSource();
ds.setName("Data Source");
Property [] prop = new Property[2];
prop[0].setName("REPORT_DATASOURCE_DRIVER_CLASS");
prop[0].setValue("spssoem.jdbc.sqlserver.SQLServerDriver");
prop[1].setName("REPORT_DATASOURCE_URL");
prop[1].setValue("jdbc:spssoem:sqlserver://db1:1433;SelectMethod=cursor;DatabaseName=cq_ecm_data");
ds.setProperties(prop);
ds.setIsCredentialRequired(false);
Credential cred = new Credential();
cred.setUsername("xpuser");
cred.setPassword("xppass");
dsLogin.setDataSource(ds);
dsLogin.setCredential(cred);
rSpec.setDataSources(dsLogin);

OutputSpecification oSpec = new OutputSpecification();
oSpec.setMimeType("text/html");
oSpec.setEncoding("utf8");
RepositoryOutput  rOutput = new RepositoryOutput();
rOutput.setAuthor("admin");
URI myuri = new URI();
myuri.setPath("spsscr:////my_report.html");
rOutput.setUri(myuri);
rOutput.setLabel("LATEST");
AccessControlList myACL = new AccessControlList();
rOutput.setAccessControlList(myACL);
oSpec.setOutputLocation(rOutput);
rSpec.setOutputs(oSpec);

Property rProp = new Property;
prop.setName("RENDERPROPERTY_REPORT_MODE");
prop.setValue("REPORT_MODE_SCHEDULED");
rSpec.setRenderProperties(rProp);

rSpec.setLocale("en_US");

RunJobResult result = stub.runJob(rSpec);

System.out.println("Execution ID = " + result.getEventExecutionID();
System.out.getNodeURIPrefix("Node prefix = " + result.getEventExecutionID();
```

SOAP request example

Client invocation of the runJob operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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:Header>
        <wsse:Security
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
            xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            xmlns:spsssec="http://xml.spss.com/security"
            soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0"
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <wsse:BinarySecurityToken wsu:Id="spssToken" ValueType="spsssec:BinarySecurityToken"
                EncodingType="wsse:Base64Binary"
                xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
                rO0ABXNyAB5jb20uc3Bzcy5zZWN1cml0eS5jb3JlLlN1YmplY3QAAAAAAAAAAQIAAlsAC21fc2lnbmF0dXJldAACW0JMA
                A1tX3N1YmplY3REYXRhdAAtTGNvbS9zcHNzL3N1Y3VyaXR5L2NvcmUvaW50ZXJuYWwvU3ViamVjdERhdGE7eHBwc3IAK2Nv
                bS5zcHNzLnNlY3VyaXR5LmNvcmUuaW50ZXJuYWwuU3ViamVjdERhdGEAAAAAAAAAAQIAB1oACW1faXNBZG1pbkkAEG1fcHJ
                pbmNpcGxlSW5kZXhKAA5tX3NlcmlhbE51bWJlckwACW1fYWRkcmVzc3QAEkxqYXZhL2xhbmcvU3RyaW5nO0wAE21fYW5ub3
                RhdGlvbk1lc3NhZ2VxAH4ABUwADm1fcHJpbmNpcGFsSURzdAAQTGphdmEvdXRpbC9MaXN0O0wACG1fdXNlcklEcQB+AAV4c
                AEAAAAAAAABIBtFLr90AAkxMjcuMC4wLjFwc3IAJmphdmEudXRpbC5Db2xsZWN0aW9ucyRVbm1vZGlmaWFibGVMaXN0/A8l
                MbXsjhACAAFMAARsaXN0cQB+AAZ4cgAsamF2YS51dGlsLkNvbGxlY3Rpb25zJFVubW9kaWZpYWJsZUNvbGxlY3Rpb24ZQgC
                Ay173HgIAAUwAAWN0ABZMamF2YS91dGlsL0NvbGxlY3Rpb247eHBzcgATamF2YS51dGlsLkFycmF5TGlzdHiB0h2Zx2GdAw
                ABSQAEc2l6ZXhwAAAAA3cEAAAAA3QAEC8vdU5hdGl2ZS8vYWRtaW50ACMvL2dOYXRpdmUvLyQkc2VjdXJpdHkvZXZlcnlvb
                mVHcm91cHQAKC8vck5hdGl2ZS8vJCRzZWN1cml0eS9yb2xlQWRtaW5pc3RyYXRvcnN4cQB+AA50AAVhZG1pbg==
            </wsse:BinarySecurityToken>
        </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
        <runJob xmlns="http://spss.com/reporting/ws/schema">
            <renderSpec>
                <reportLocation xsi:type="ns1:uriLocation"
```

```
                    xmlns:ns1="http://spss.com/reporting/ws/schema">
                    <ns1:uri>spsscr:///?id=0a010a074a2f7bcc000001201b05e1d980c9</ns1:uri>
                </reportLocation>
                <dataSources>
                    <dataSource>
                        <name>Data Source</name>
                        <properties>
                            <name>REPORT_DATASOURCE_URL</name>
                            <value>jdbc:spssoem:sqlserver://db1:1433;SelectMethod=cursor;DatabaseName=cq_ecm_data</value>
                        </properties>
                        <properties>
                            <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
                            <value>spssoem.jdbc.sqlserver.SQLServerDriver</value>
                        </properties>
                        <isCredentialRequired>false</isCredentialRequired>
                    </dataSource>
                    <credential xsi:type="ns2:credential"
                        xmlns:ns2="http://spss.com/reporting/ws/schema">
                        <ns2:username>xpuser</ns2:username>
                        <ns2:password>xppass</ns2:password>
                    </credential>
                </dataSources>
                <outputs>
                    <mimeType>text/html</mimeType>
                    <encoding>utf8</encoding>
                    <outputLocation xsi:type="ns3:repositoryOutput"
                        xmlns:ns3="http://spss.com/reporting/ws/schema">
                        <ns3:viewKey>0a010a074a2f7bcc000001201b05e1d98114</ns3:viewKey>
                        <ns3:uri>spsscr:////my_report.html</ns3:uri>
                        <ns3:author>admin</ns3:author>
                        <ns4:AccessControlList xmlns:ns4="http://xml.spss.com/repository"/>
                        <ns3:label>LATEST</ns3:label>
                    </outputLocation>
                </outputs>
                <renderProperties>
                    <name>RENDERPROPERTY_REPORT_MODE</name>
                    <value>REPORT_MODE_SCHEDULED</value>
                </renderProperties>
                <locale>en_US</locale>
            </renderSpec>
        </runJob>
    </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `runJob` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
      <runJobResponse xmlns="http://spss.com/reporting/ws/schema">
          <runJobResult>
              <eventExecutionID>0a010a074a2f7bcc000001201b05e1d98181</eventExecutionID>
              <nodeURIPrefx>http://localhost:8080</nodeURIPrefx>
          </runJobResult>
      </runJobResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The setJobResult operation

Sets the job results, using the eventExecutionID returned from the `renderJob` operation.

Input fields

The following table lists the input fields for the `setJobResult` operation.

Table 20. Fields for setJobResult.

Field	Type/Valid Values	Description
eventExecutionID	string	Job ID returned from renderJob.

Table 20. Fields for setJobResult (continued).

Field	Type/Valid Values	Description
jobResult	jobResult	JobResult contains the RenderedResult and the JobStatus.

The updateJobStatus operation

Uses the job's eventExecutionID to update the job's status. One of the following WSDLConstants can be used to set the status:

- JOB_STATUS_PENDING
- JOB_STATUS_RUNNING
- JOB_STATUS_SUCCESS
- JOB_STATUS_FAILED
- JOB_STATUS_CANCELED
- JOB_STATUS_TIMEOUT

Input fields

The following table lists the input fields for the `updateJobStatus` operation.

Table 21. Fields for updateJobStatus.

Field	Type/Valid Values	Description
eventExecutionID	string	Job ID returned from renderJob.
jobStatus	jobStatus	

The validateDataSource operation

Verifies the report data source is defined correctly and that valid credentials exist. This is done by attempting a connection to the data source. If a data source cannot be accessed, attempts to generate a report based on that source will not provide useful results.

Typically, this operation is used before processing a report. If a data source fails, the user can be prompted for alternative credentials for the source. All data sources should be validated before generating report results.

Input fields

The following table lists the input fields for the `validateDataSource` operation.

Table 22. Fields for validateDataSource.

Field	Type/Valid Values	Description
reportLocation	locationType	Object indicating the report's location. Can be of type AttachmentLocation or URILocation.
dataSourceLogins	dataSourceLogin[]	Array of DataSourceLogins, where data source and credential information is set.

Return information

The following table identifies the information returned by the validateDataSource operation.

Table 23. Return Value.

Type	Description
validateDataSourceResult[]	

Java example

To validate a data source:

1. Create a UriLocation object and use the setUri method to assign a string corresponding to the URI for the report.
2. Create a DataSourceLogin object for the data source and credential information.
3. Create a DataSource object and use the setName method to define the name of the data source.
4. Create an array of Property objects to define the data source connection information. For each property, use the setName method to define the name and the setValue method to define the property value. Use the setProperties method to assign the property array to the data source object.
5. Supply the setIsCredentialRequired method with a boolean indicating whether or not credentials are required.
6. Use the setDataSource method to assign the data source to the login object.
7. Create a Credential object for the data source credentials. Use the setUsername method to assign a string corresponding to the user name for the credential. Use the setPassword method to assign a string corresponding to the password for the credential.
8. Use the setCredential method to assign the credential to the login object.
9. Supply the validateDataSource operation with the location and login objects.

```
String uri = new String();
uri = "spsscr://pes_server:80/reports/xp.rptdesign#m.0:2008-04-08%2013:41:51.089";
UriLocation location = new UriLocation();
location.setUri(uri);
DataSourceLogin dsLogin = new DataSourceLogin();
DataSource ds = new DataSource();
ds.setName("XPlanner");
Property [] prop = new Property[2];
prop[0].setName("REPORT_DATASOURCE_DRIVER_CLASS");
prop[0].setValue("com.mysql.jdbc.Driver");
prop[1].setName("REPORT_DATASOURCE_URL");
prop[1].setValue("jdbc:mysql://mylinux:3306/xplanner");
ds.setProperties(prop);
ds.setIsCredentialRequired(false);
Credential cred = new Credential();
cred.setUsername("validUser");
cred.setPassword("password");
dsLogin.setDataSource(ds);
dsLogin.setCredential(cred);
ValidateDataSourceResult[] result = stub.validateDataSource(location, dsLogin);

for (int j = 0; j < result.length; j++) {
   System.out.println(result[j].getDataSourceLogin().getDataSource().getName() +
      " status: " + result[j].getStatus());
   System.out.println("Message: " + result[j].getMessage());
}
```

SOAP request example

Client invocation of the validateDataSource operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<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:Header>
    <wsse:Security soapenv:mustUnderstand="0"
```

```
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <validateDataSource xmlns="http://spss.com/reporting/ws/schema">
      <reportLocation xsi:type="ns1:uriLocation" xmlns:ns1="http://spss.com/reporting/ws/schema">
        <ns1:uri xsi:type="xsd:anyURI"
          >spsscr://pes_server:80/reports/xp.rptdesign#m.0:2008-04-08%2013:41:51.089</ns1:uri>
      </reportLocation>
      <dataSourceLogins>
        <dataSource>
          <name>XPlanner</name>
          <properties>
            <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
            <value>com.mysql.jdbc.Driver</value>
          </properties>
          <properties>
            <name>REPORT_DATASOURCE_URL</name>
            <value>jdbc:mysql://mylinux:3306/xplanner</value>
          </properties>
          <isCredentialRequired>false</isCredentialRequired>
        </dataSource>
        <credential xsi:type="ns2:credential" xmlns:ns2="http://spss.com/reporting/ws/schema">
          <ns2:username>validUser</ns2:username>
          <ns2:password>password</ns2:password>
        </credential>
      </dataSourceLogins>
    </validateDataSource>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `validateDataSource` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<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>
    <validateDataSourceResponse xmlns="http://spss.com/reporting/ws/schema">
      <validateDataSourceResults>
        <status>FAILED</status>
        <message>Access denied for user 'validUser'@'machname.company.com' (using password: YES)</message>
        <dataSourceLogin>
          <dataSource>
            <name>XPlanner</name>
            <properties>
              <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
              <value>com.mysql.jdbc.Driver</value>
            </properties>
            <properties>
              <name>REPORT_DATASOURCE_URL</name>
              <value>jdbc:mysql://mylinux:3306/xplanner</value>
            </properties>
            <isCredentialRequired>false</isCredentialRequired>
          </dataSource>
          <credential xsi:type="ns1:credential" xmlns:ns1="http://spss.com/reporting/ws/schema">
            <ns1:username>validUser</ns1:username>
            <ns1:password>password</ns1:password>
          </credential>
        </dataSourceLogin>
      </validateDataSourceResults>
    </validateDataSourceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Chapter 5. JAX-WS clients

Java developers can create custom web service clients by using JAX-WS.

The discussion here assumes the use of Java 6. In general, the process for accessing IBM SPSS Collaboration and Deployment Services web services involves the following steps:

1. Generate a web service client using wsimport
2. Package the client
3. Programmatically configure the client
4. Exercise the web service

Generating a JAX-WS client

To generate a JAX-WS client, open a command prompt and execute the Java 6 `wsimport` command.

The `wsimport` command creates JAX-WS service classes and JAXB classes that represent the WSDL schema. For example, the following command executes wsimport for the Scoring.HttpV2 service, storing the output in the current directory:

```
"c:\Program Files\IBM\Java60\bin\wsimport.exe" http://localhost:7001/scoring/services/Scoring.HttpV2?wsdl
```

In this example, the command obtained the WSDL from the server by using the endpoint name followed by `?wsdl`. The `wsimport` command requires access to the WSDL in order to generate the files. JAX-WS also requires access to the WSDL file during runtime, so this example hard codes the value provided to `wsimport` in the Java code. The generated client fetches the WSDL from that same location unless otherwise specified. An alternative is to store the WSDL locally and refer to the local copy rather than downloading the WSDL from the server.

Packaging a JAX-WS client

A JAX-WS client must be packaged as a jar file.

The following example command creates a jar file named `scoring.jar`:

```
"c:\Program Files\IBM\Java60\bin\jar.exe" -cvf scoring.jar *
```

This command assumes the command prompt is in the same location in which the client was generated.

If you store the WSDL locally, include the WSDL and XSD files in the jar file. Place the files in the `\META-INF\wsdl` directory within the file. Refer to that directory programmatically when configuring the client.

Configuring a JAX-WS client

JAX-WS clients can obtain the WSDL file remotely from the server or locally from within the jar file.

The following example demonstrates obtaining the WSDL from the server:

```
com.spss.scoring.ws.jaxws.ScoringServices service =
  new com.spss.scoring.ws.jaxws.ScoringServices(
    new URL("http://localhost:7001/scoring/services/Scoring.HttpV2?wsdl"),
    new QName("http://xml.spss.com/scoring/wsdl", "ScoringServices"));
```

The URL includes the host and port for your server.

The following example demonstrates obtaining the WSDL from the within the jar file:

```
com.spss.scoring.ws.jaxws.ScoringServices service =
  new com.spss.scoring.ws.jaxws.ScoringServices(
    DemoClass.class.getResource("/META-INF/wsdl/scoring.wsdl"),
    new QName("http://xml.spss.com/scoring/wsdl", "ScoringServices"));
```

In order to include the required SOAP security headers, create an object that implements `SOAPHandler<SOAPMessageContext>`. See "SOAPHandler example" for an example handler object. The following example shows how this object is used:

```
service.setHandlerResolver(new HandlerResolver()
{
  @Override
    public List<Handler> getHandlerChain(PortInfo portInfo)
    {
      List<Handler> handlerChain = new ArrayList<Handler>();
      handlerChain.add(new SecurityHandler("user", "password", "en-US;q=1.0, en;q=0.8"));
      return handlerChain;
    }
});
```

Next, access the service endpoint:

```
ScoringV2 serviceEndpoint = service.getHttpV2();
```

After obtaining the service endpoint, set the JAX-WS standard endpoint address property, which specifies the URL at which to access the endpoint.

```
Map<String, Object> requestContext = ((BindingProvider)serviceEndpoint).getRequestContext();
requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
   "http://localhost:7001/scoring/services/Scoring.HttpV2");
```

SOAPHandler example

JAX-WS clients must include an object that implements `SOAPHandler<SOAPMessageContext>`.

The following code provides an example of this object.

```
/*************************************************************
** Licensed Materials - Property of IBM
** IBM SPSS Products: Collaboration and Deployment Services
** © Copyright IBM Corp. 2000, 2013
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
**
*************************************************************/

import java.util.Collections;
import java.util.Set;

import javax.xml.namespace.QName;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPFactory;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

/**
 * This is a SOAP handler that applies a security header and a language header to a SOAP message.
 */
public class SecurityHandler implements SOAPHandler<SOAPMessageContext>
{
    // WS-Security header values
    public static final String SECURITY = "Security";
    public static final String USERNAME_TOKEN = "UsernameToken";
    public static final String USERNAME = "Username";
    public static final String PASSWORD = "Password";
    public static final String WS_SECURITY_NAMESPACE =
       "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd";

    // prefixes
```

```
public static final String WSSE_PREFIX = "wsse"; // ws service security
public static final String SPSS_PREFIX = "spss"; // spss prefix

// SPSS custom language header values
public static final String SPSS_HEADER_NAMESPACE = "http://xml.spss.com/ws/headers";
public static final String CLIENT_ACCEPT_LANGUAGE_HEADER = "client-accept-language";

private String i_username;
private String i_password;
private String i_acceptLanguage;

/**
 * Creates a security and language handler
 * @param username A user name to access the web service. Cannot be null.
 * @param password A password to access the web service. Cannot be null.
 * @param acceptLanguage The language that should be used by the web service.
 * This value should be formatted according to the HTTP specification regarding
 * the Accept-Language HTTP header (e.g. en-US;q=1.0, en;q=0.8)
 * If the value is null, the language header will not be added.
 */
public SecurityHandler(String username, String password, String acceptLanguage)
{
    i_username = username;
    i_password = password;
    i_acceptLanguage = acceptLanguage;
}

@Override
public boolean handleMessage(SOAPMessageContext context)
{
    // Apply this handler to only outbound traffic
    if((Boolean)context.get(SOAPMessageContext.MESSAGE_OUTBOUND_PROPERTY))
    {
        // get the message
        SOAPMessage message = context.getMessage();
        try
        {
            // get the message header
            SOAPEnvelope envelope = message.getSOAPPart().getEnvelope();
            SOAPHeader header = envelope.getHeader();
            if (header == null)
            {
                header = envelope.addHeader();
            }

            // add the UsernameToken header
            header.addChildElement(createUsernameTokenSecurityHeader());
            // assuming the language was provided, apply the custom language header
            if(i_acceptLanguage != null)
            {
                header.addChildElement(createLanguageHeader());
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    // allow any other handler to execute
    return true;
}

/**
 * This method creates a custom language header, which allows the scoring service
 * to use the given language if possible.
 * @return A custom language header
 * @throws Exception
 */
private SOAPElement createLanguageHeader() throws Exception
{
    SOAPFactory factory = SOAPFactory.newInstance();

    // create a custom language header
    SOAPElement languageHeader =
       factory.createElement(CLIENT_ACCEPT_LANGUAGE_HEADER,SPSS_PREFIX,SPSS_HEADER_NAMESPACE);

    // include the language text
    languageHeader.addTextNode(i_acceptLanguage);

    return languageHeader;
}

/**
 * Creates the WS-Security SOAP header for UsernameToken as SOAPElement.
```

```
     *
     * @return the WS-Security SOAP header for UsernameToken
     * @throws Exception as appropriate
     */
    private SOAPElement createUsernameTokenSecurityHeader() throws Exception
    {
        SOAPFactory factory = SOAPFactory.newInstance();

        // create a UsernameToken element
        SOAPElement usernameToken =
          factory.createElement(USERNAME_TOKEN, WSSE_PREFIX, WS_SECURITY_NAMESPACE);

        // add the username element
        SOAPElement usernameElement =
          factory.createElement(USERNAME, WSSE_PREFIX, WS_SECURITY_NAMESPACE);
        usernameElement.addTextNode(i_username);
        usernameToken.addChildElement(usernameElement);

        // add the password element
        SOAPElement passwordElement =
          factory.createElement(PASSWORD, WSSE_PREFIX, WS_SECURITY_NAMESPACE);
        passwordElement.addTextNode(i_password);
        usernameToken.addChildElement(passwordElement);

        // create the Security Header
        SOAPElement securityHeader =
          factory.createElement(SECURITY, WSSE_PREFIX, WS_SECURITY_NAMESPACE);
        securityHeader.addChildElement(usernameToken);

        return securityHeader;
    }

    @Override
    public boolean handleFault(SOAPMessageContext context)
    {
        // allow any other handler to execute
        return true;
    }

    @Override
    public void close(MessageContext context)
    {
        // do nothing
    }

    @Override
    public Set<QName> getHeaders()
    {
        return Collections.emptySet();
    }

}
```

Exercising web services from JAX-WS clients

Once properly configured, a JAX-WS client can make calls to IBM SPSS Collaboration and Deployment Services web services.

For example, the following code calls the `getConfigurations` operation of the Scoring Service:

```
serviceEndpoint.getConfigurations();
```

Chapter 6. Microsoft® .NET Framework-based clients

In order to use the web services from a Microsoft Windows Communication Foundation (WCF) client, you will need Visual Studio 2008 or later. The discussion here assumes the use of Visual Studio 2008. In general, the process for accessing IBM SPSS Collaboration and Deployment Services web services involves the following steps:

1. Add a Service Reference. See the topic "Adding a service reference" for more information.
2. Configure the web service endpoint. See the topic "Configuring the web service endpoint" on page 42 for more information.
3. Programmatically configure the necessary endpoint behaviors. See the topic "Configuring endpoint behaviors" on page 43 for more information.
4. Exercise the web service. See the topic "Exercising the service" on page 43 for more information.

Adding a service reference

The first step in using a WCF client to access IBM SPSS Collaboration and Deployment Services web services is to make the service available to the Visual Studio project by adding it as a Service Reference.

1. In Visual Studio, right-click the folder's *References* folder and select **Add Service Reference**.
2. Type the URL of the service WSDL location in the **Address** field, and click **Go**. The value corresponds to the service endpoint appended with `?wsdl`.
3. Specify the namespace in the **Namespace** field.
4. Click **OK**.

Visual Studio adds a new service reference to the Service Reference directory for the project. The name of the reference corresponds to the specified namespace.

Important: If you have a .NET client created by using a version of IBM SPSS Collaboration and Deployment Services before 6.0, you must regenerate your service references from the current WSDL files to allow successful communication between your application and the current server. If you do not regenerate your service references, you may experience a variety of errors that may include incorrect namespace assignments, NullPointerExceptions in the web services being invoked, and data type assignment errors.

Service reference modifications

Due to known compatibility issues between Microsoft tooling and some WSDL files, you need to manually modify some service references before they can be used successfully. For information about the specific issues, see articles 891386 and 326790 on the Microsoft Support site.

To modify a service reference:

1. In Visual Studio, select the project and click **Show All Files** from the Project menu.
2. Expand the service reference that needs to be modified.
3. Expand the **Reference.svcmap** node.
4. Open the `Reference.cs` file.
5. Make the required modifications.
6. Save the file.

For the Content Repository Service , Content Repository URI Service, and Process Management Service, you need to make the following changes to the `RowType` class:

- `private value[][] cellField` should be changed to `private value[] cellField`
- `public value[][] cell` should be changed to `public value[] cell`

For the Scoring Service, you need to make the following changes:

- in the `returnedDPDOutputTable` class, `private returnedDPDOutputValue[][] returnedDPDOutputrowField` should be changed to `private returnedDPDOutputValue[] returnedDPDOutputrowField`
- in the `returnedDPDOutputTable` class, `private returnedDPDOutputValue[][] returnedDPDOutputRow` should be changed to `private returnedDPDOutputValue[] returnedDPDOutputRow`
- in the `returnedRequestInputTable` class, `private returnedRequestInputValue[][] returnedRequestInputRow` should be changed to `private returnedRequestInputValue[] returnedRequestInputRow`
- in the `returnedRequestInputTable` class, `private returnedRequestInputValue[][] returnedRequestInputRowField` should be changed to `private returnedRequestInputValue[] returnedRequestInputRowField`
- in the `requestInputTable` class, `private input1[][] requestInputRowField` should be changed to `private input1[] requestInputRowField`
- in the `requestInputTable` class, `private input1[][] requestInputRow` should be changed to `private input1[] requestInputRow`

For the PevServices Service, you need to make the following changes:

- in the `avTableConflict` class, `private avColumnMeta[][] avColumnConflictField` should be changed to `private avColumnMeta[] avColumnConflictField`
- in the `avTableConflict` class, `private avColumnMeta[][] avColumnConflict` should be changed to `private avColumnMeta[] avColumnConflict`
- in the `evTableConflict` class, `private evColumnMeta[][] evColumnConflictField` should be changed to `private evColumnMeta[] evColumnConflictField`
- in the `evTableConflict` class, `private evColumnMeta[][] evColumnConflict` should be changed to `private evColumnMeta[] evColumnConflict`

Configuring the web service endpoint

In WCF, you can configure a service endpoint either declaratively using an *app.config* file, or programmatically using the WCF APIs. The following steps describe the creation of a basic configuration within an *app.config* file.

1. In Visual Studio, double-click the *app.config* file for the application (or *web.config* for a web-application).
2. Find the `system.serviceModel` element. Create it if it does not already exist.
3. Find the `client` element. Create it if it does not already exist.
4. Create a new `endpoint` element as a child of the `client` element.
5. Specify the appropriate service endpoint URL as the value of the *address* attribute.
6. Specify *basicHttpBinding* as the value of the *binding* attribute.
7. Specify the appropriate service contract as the value of the *contract* attribute. The service contract is the value of the service reference namespace appended with the service name.
8. Optionally specify a value for the *name* attribute that identifies a name for the endpoint configuration. If the *name* is blank, the configuration is used as the default for the service.

The resulting *app.config* file should be similar to the following example:

```
<system.serviceModel>
   <client>
      <endpoint
         address="http://cads_server:8080/cr-ws/services/ContentRepository"
```

```
        binding="basicHttpBinding"
        bindingConfiguration=""
        contract="IBM.SPSS.ContentRepository"
        name=""/>
   </client>
</system.serviceModel>
```

Configuring endpoint behaviors

The following two issues complicate the use of IBM SPSS Collaboration and Deployment Services web services by WCF clients:

- WCF does not allow the username and password to be transmitted over HTTP
- WCF does not correctly understand the SOAP Fault format returned by the services

To address these problems, a sample Visual Studio project is available that contains classes adding endpoint behaviors that resolve both issues. The IBM SPSS Collaboration and Deployment Services installation media includes this project.

To use these classes, ensure that the *IBM.SPSS.WCF.Utilities* project containing these classes has been compiled and added as a referenced DLL to the Visual Studio project that exercises the web services. When constructing a new service client instance, ensure that the behaviors are added as follows:

```
ContentRepositoryClient serviceClient = new ContentRepositoryClient();
serviceClient.Endpoint.Behaviors.Add(
    new ApplyClientInspectorsBehavior(
    new HeaderInjectionMessageInspector(
        new UsernameTokenSecurityHeader("admin", "Abcdefg1")
    ),
    new SOAPFaultFormatMessageInspector())
);
```

This adds two message inspectors to the behaviors for the endpoint. The first allows message headers to be injected, permitting a `UsernameToken` security header containing the username and password to be transmitted over HTTP. The second message inspector intercepts SOAP Faults, ensuring that they are formatted for proper WCF processing.

Exercising the service

After adding the service reference to the project, configuring the endpoint, and adding the necessary endpoint behaviors, the WCF-based web service client is ready. Add the .NET source code to the project to exercise the web service as needed.

There may be instances in which the .NET client proxies are generated incorrectly, leading to unexpected missing results at runtime. If a web service call returns no results when results are expected, the generated .NET types associated with the request and response should be examined. Specifically, members of the types may have two .NET attributes assigned. The first, `MessageBodyMemberAttribute`, will often include the proper namespace for the member type. The second, `XmlElementAttribute`, should have the same namespace as `MessageBodyMemberAttribute`. If this is not the case, add the namespace to `XmlElementAttribute`. Moreover, the addition of XML serialization attributes, such as `System.XML.Serialization.XmlElementAttribute`, may be necessary to correctly name the expected namespace or element. For example, the following generated client code would need to be modified:

```
public partial class getUsersResponse {
   System.ServiceModel.MessageBodyMemberAttribute(Namespace =
      "http://xml.spss.com/pes/userPref/remote", Order = 0)]
   public IBM.SPSS.ManagerUserPref.usersResponse usersResponse;
```

The corrected code is as follows:

```
public partial class getUsersResponse {
   [System.ServiceModel.MessageBodyMemberAttribute(Namespace =
      "http://xml.spss.com/pes/userPref/remote", Order = 0)]
   [System.Xml.Serialization.XmlElementAttribute(ElementName="usersRequestResponse")]
   public IBM.SPSS.ManagerUserPref.usersResponse usersResponse;
```

Single sign-on authentication

You can use single sign-on authentication for web service calls by obtaining a service ticket that you include in your SOAP requests.

The general process of using single sign-on authentication for WCF clients includes the following steps:

1. Obtain a ticket-grating ticket (TGT) using .NET or WCF code.
2. Send the TGT to the IBM SPSS Collaboration and Deployment Services Repository server using the SSO Authentication Service `getToken` operation to obtain a service ticket. This ensures that single sign-on authentication occurs on the repository server.
3. Send the service ticket in the SOAP header for all subsequent web services calls from your client application.

Chapter 7. Message header reference

The headers for the transport and packaging layers contain vital information for processing a web service call.

For IBM SPSS Collaboration and Deployment Services, the SOAP headers contain the security information under which the web service call is processed. In addition, the HTTP headers contain information about the client that initiated the web service request.

Security headers

Most IBM SPSS Collaboration and Deployment Services web service calls require security information in the request message.

In general, the structure of this content follows the `WS-Security` extension to the SOAP 1.1 standard. This documentation provides details on the XML elements and attributes that are recognized by IBM SPSS Collaboration and Deployment Services. Some of the elements and attributes are required, some are optional, and some are ignored. Refer to the following official specifications for details, but IBM SPSS Collaboration and Deployment Services requires some special values not referenced in the official specifications.

- http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
- http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-UsernameTokenProfile.pdf
- http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-KerberosTokenProfile.pdf

The following table defines the values of namespaces that are used for the SOAP header elements.

Table 24. SOAP header namespaces

Namespace prefix	Namespace value
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
soapenv	http://schemas.xmlsoap.org/soap/envelope/
spsssec	http://xml.spss.com/security

Security element

The `wsse:Security` element is the main security header element included in a `soapenv:Header` element.

Table 25. Attributes of `wsse:Security`

Attribute	Description	Example
`soapenv:actor`	Targets a given endpoint along the message path. This value is ignored.	http://schemas.xmlsoap.org/soap/actor/next
`soapenv:mustUnderstand`	Clients can specify if the server must process this element. This value is ignored.	0

UsernameToken element

Use the wsse:UsernameToken element when a traditional user and password combination is required.

Table 26. Attributes of wsse:UsernameToken

Attribute	Description
wsu:Id	An optional string label for the security token. This value is ignored.

Table 27. Child elements of wsse:UsernameToken

Attribute	Description	Example
wsse:Username	The xml value represents the identity of the user.	a_user
wsse:Password	The attribute Type specifies the type of password. PasswordText is currently the only supported type. The xml value can handle plain text passwords and encrypted data.	myPassword [{AES}KrY+KLlOYo4O6545tgGsYQ==]
wsse:Nonce	The xml value represents a cryptographically random nonce encoded as base64 data. This is currently ignored.	RUx1ugQo0o3g0Xyl+sUEsA==
wsu:Created	The xml value represents the creation time as a timestamp conforming to wsu:Timestamp. This is currently ignored.	2013-10-08T02:09:20Z

BinarySecurityToken and BinarySecuritySSOToken elements

Binary security tokens may be used when IBM SPSS Collaboration and Deployment Services communicates with itself or when single sign-on (SSO) is used. Customer usage of these token types is limited to SSO.

The wsse:BinarySecurityToken and wsse:BinarySecuritySSOToken elements have the same format, but only wsse:BinarySecurityToken is recognized in the official WS-Security standard. The element wsse:BinarySecuritySSOToken was added as a nonstandard element when used in SSO.

Of these two elements, you should use wsse:BinarySecurityToken and you must supply the correct attributes for proper handling. The most critical attribute is the wsu:Id value which is used during web service request processing to handle the security token correctly.

Table 28. Attributes of wsse:BinarySecurityToken

Attribute	Description	Example
ValueType	Indicates the type of the security token. IBM SPSS Collaboration and Deployment Services always writes these values when creating its own XML, but this value is currently ignored during processing. You should use spsssec:BinarySecuritySSOToken.	spsssec:BinarySecurityToken spsssec:BinarySecuritySSOToken

Table 28. Attributes of wsse:BinarySecurityToken (continued)

Attribute	Description	Example
EncodingType	Indicates the encoding type for the token. The only currently supported type is base64, so this value should always be wsse:Base64Binary. IBM SPSS Collaboration and Deployment Services always writes these values when creating its own XML, but this value is currently ignored during processing.	wsse:Base64Binary
wsu:Id	An identifier for the token. This value must be correctly provided. You should always provide spssSSOToken. The only valid case for using spssToken is for internal web service calls, which use an internal token format.	spssToken spssSSOToken
anyAttribute	An extension mechanism to allow any arbitrary attribute in other namespaces. These extensions are ignored.	

The XML value for wsse:BinarySecurityToken and wsse:BinarySecuritySSOToken is string data in base64 format.

The client-accept-language element

This element restricts the set of natural languages that are preferred as a response to the request.

This element is inserted into a soapenv:Header element and is not related to WS-Security in any way. This is the same value found in the HTTP header named Accept-Language as defined in RFC2068. The xml value for this element might look like the following:

```
en-US;q=1.0, en;q=0.8
```

The namespace for this element could be any allowed value, such as ns1, which has an associated value of http://xml.spss.com/ws/headers.

HTTP headers

In addition to SOAP headers, it is possible to apply HTTP headers as well. None of the HTTP headers is required.

Table 29. HTTP headers

HTTP header	Description
Accept-Language	The accept language header value, as defined in RFC2068 (e.g. en-US;q=1.0, en;q=0.8). If not supplied the server language setting is used as a default.
CLIENT_ADDR	The client IP address that ultimately initiated the request.
CLIENT_HOSTNAME	The client host name that ultimately initiated the request.
X-FORWARDED-FOR	The client IP address that ultimately initiated the request. This is standard for determining the originating IP address.

The CLIENT_ADDR, CLIENT_HOSTNAME, and X-FORWARDED-FOR values are useful when a client application makes a call through an HTTP proxy, load balancer, or when IBM SPSS Collaboration and Deployment Services components make internal calls. The CLIENT_ADDR and CLIENT_HOSTNAME entries are specific HTTP headers that can be set by IBM SPSS Collaboration and Deployment Services itself. The X-FORWARDED-FOR header is a standard that some load balancers understand. These headers are used to make a best-effort attempt in determining the originating client for a given call, allowing information to be used for auditing purposes. The headers may not work as intended, but IBM SPSS Collaboration and Deployment Services will fall back to reasonable defaults in those situations.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Software Group
ATTN: Licensing
200 W. Madison St.
Chicago, IL; 60606
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

Glossary

Index

Special characters

A

B

C

D

G

H

J

L

M

N

P

R

S

T

U

V

W

X

Printed in USA