

**IBM SPSS Collaboration and
Deployment Services 5 Reporting
Service Developer's Guide**



Note: Before using this information and the product it supports, read the general information under Notices on p. 53.

This edition applies to IBM SPSS Collaboration and Deployment Services 5 and to all subsequent releases and modifications until otherwise indicated in new editions.

Adobe product screenshot(s) reprinted with permission from Adobe Systems Incorporated.

Microsoft product screenshot(s) reprinted with permission from Microsoft Corporation.

Licensed Materials - Property of IBM

© **Copyright IBM Corporation 2000, 2012.**

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Preface

This guide is intended for developers working with the web services available in IBM® SPSS® Collaboration and Deployment Services. Users should have experience writing web service client applications and are assumed to have knowledge of IBM Corp. applications, Java and .NET development, data modeling concepts, and related technologies.

About IBM Business Analytics

IBM Business Analytics software delivers complete, consistent and accurate information that decision-makers trust to improve business performance. A comprehensive portfolio of [business intelligence](#), [predictive analytics](#), [financial performance and strategy management](#), and [analytic applications](#) provides clear, immediate and actionable insights into current performance and the ability to predict future outcomes. Combined with rich industry solutions, proven practices and professional services, organizations of every size can drive the highest productivity, confidently automate decisions and deliver better results.

As part of this portfolio, IBM SPSS Predictive Analytics software helps organizations predict future events and proactively act upon that insight to drive better business outcomes. Commercial, government and academic customers worldwide rely on IBM SPSS technology as a competitive advantage in attracting, retaining and growing customers, while reducing fraud and mitigating risk. By incorporating IBM SPSS software into their daily operations, organizations become predictive enterprises – able to direct and automate decisions to meet business goals and achieve measurable competitive advantage. For further information or to reach a representative visit <http://www.ibm.com/spss>.

Technical support

Technical support is available to maintenance customers. Customers may contact Technical Support for assistance in using IBM Corp. products or for installation help for one of the supported hardware environments. To reach Technical Support, see the IBM Corp. web site at <http://www.ibm.com/support>. Be prepared to identify yourself, your organization, and your support agreement when requesting assistance.

Contents

1	<i>Introduction to web services</i>	1
	What are web services?	1
	Web service system architecture	1
	Web service protocol stack	2
	Simple Object Access Protocol	3
	Web Service Description Language	3
	Proxies	6
2	<i>Reporting Service overview</i>	8
	Accessing the Reporting Service	8
	Calling Reporting Service operations	8
	Workflow	8
3	<i>Reporting concepts</i>	10
	Reports	10
	Data sources	10
	Data sets	10
	Variable groups	11
	Jobs	12
	URI locations	13
	Prompt values	14
4	<i>Operation reference</i>	15
	The cancelJob operation	15
	The getReportMetadata operation	16
	The getSelectedReportMetadata operation	21
	The getVersion operation	24
	The renderJob operation	25
	The renderSync operation	25
	The retrieveCascadingPromptValues operation	31
	The retrieveJobResult operation	34
	The retrievePromptValues operation	36

The runJob operation	39
The setJobResult operation	43
The updateJobStatus operation	43
The validateDataSource operation	44

Appendices

A Microsoft® .NET Framework-based clients **48**

Adding a service reference.	48
Service reference modifications	49
Configuring the web service endpoint.	50
Configuring endpoint behaviors	51
Exercising the service.	51

B Notices **53**

Index **56**

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.

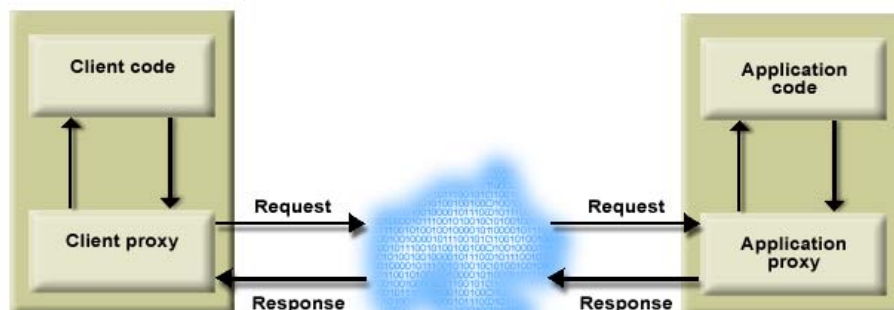
Advantages to implementing application functionality as web services include the following:

- 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 JBoss Application Server, WebSphere®, 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 1-1
Web service architecture



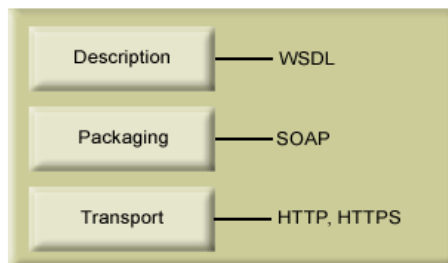
The client code supplies input to an operation offered by a proxy class. The proxy class generates a request containing a standardized representation of the input and sends it across the network to the application. A proxy class on the server receives the request and unmarshals the contents into objects 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. [For more information, see the topic Proxies on p. 6.](#)

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 1-2
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. [For more information, see the topic Simple Object Access Protocol on p. 3.](#)

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. [For more information, see the topic Web Service Description Language on p. 3.](#)

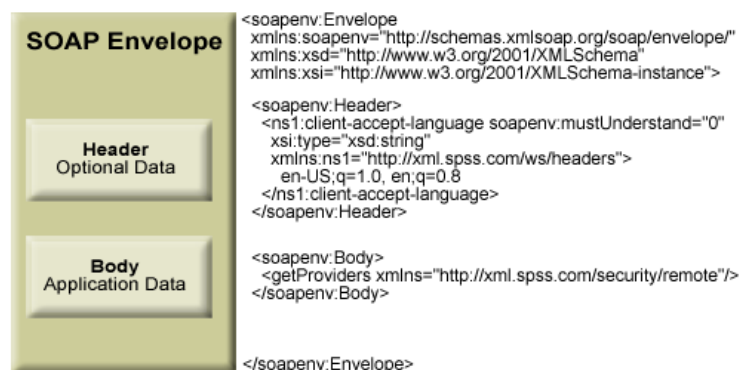
Simple Object Access Protocol

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 1-3
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 desired programming language.

In accordance with the [WSDL specification \(http://www.w3.org/TR/wsdl\)](http://www.w3.org/TR/wsdl) 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. [For more information, see the topic Types on p. 4.](#)
- **Messages.** Input and output definitions for the web service operations. [For more information, see the topic Messages on p. 5.](#)
- **PortTypes.** Groups of operations offered by the web service. [For more information, see the topic Port types on p. 5.](#)
- **Bindings.** Protocols and formats for the web service operations. [For more information, see the topic Bindings on p. 5.](#)
- **Services.** Endpoints at which the web service functionality can be accessed. [For more information, see the topic Services on p. 6.](#)

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 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="tns: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 message definitions:

```
<wsdl:message name="getProvidersResponse">
  <wsdl:part element="tns:getProvidersResponse" name="parameters" />
</wsdl:message>
<wsdl:message name="getProvidersRequest">
  <wsdl:part element="tns: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. [For more information, see the topic Types on p. 4.](#)

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 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. [For more information, see the topic Messages on p. 5.](#)

Bindings

The binding element of a WSDL file binds the interface defined by the port type to transport and messaging protocols. Consider the following 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:operation>
</wsdl:binding>
```

```
</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. Both the `getProviders` and `getProvidersResponse` operations in the interface are 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 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.

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>/reporting-ws/services/Reporting
```

The value of *<host-name>* corresponds to the machine on which IBM® SPSS® Collaboration and Deployment Services Repository is installed, with *<port-number>* indicating the port number on which it is running. 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*, 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 meta-data describing the report. [For more information, see the topic Reports in Chapter 3 on p. 10.](#)

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. [For more information, see the topic Data sources in Chapter 3 on p. 10.](#)
3. Prompt the user for any necessary report parameter values. [For more information, see the topic Prompt values in Chapter 3 on p. 14.](#)
4. Render the report, providing status if desired.
5. Retrieve the result.

Reporting concepts

Reports

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

- Data sources. [For more information, see the topic Data sources on p. 10.](#)
- Data sets. [For more information, see the topic Data sets on p. 10.](#)
- Variable groups. [For more information, see the topic Variable groups on p. 11.](#)
- 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 meta-data for reports and for retrieving a subset of meta-data.

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, on the other hand, 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. [For more information, see the topic Data sources on p. 10.](#)
- 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 3-1
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: <ul style="list-style-type: none"> ■ <i>Unformatted</i> ■ <i>General Date</i> ■ <i>Short Date</i> ■ <i>Medium Date</i> ■ <i>Long Date</i>
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

Property	Type	Description
defaultValueType	string	Default type for the value. Valid values include: <ul style="list-style-type: none"> ■ <i>NONE</i> ■ <i>CURRENT_VALUE</i> ■ <i>USER_SPECIFIED</i>
promptType	string	Determines the type for the prompt. Valid values include: <ul style="list-style-type: none"> ■ <i>SINGLE_PROMPT</i> ■ <i>LIST_PROMPT</i> ■ <i>BETWEEN_PROMPT</i> ■ <i>BOOLEAN_PROMPT</i> ■ <i>RADIO_PROMPT</i>
valueType	string	Identifies whether the parameters values are derived from a specified list (<i>static</i>) or generated (<i>dynamic</i>)
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. [For more information, see the topic URI locations on p. 13.](#)
- 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][#.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][#.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
```

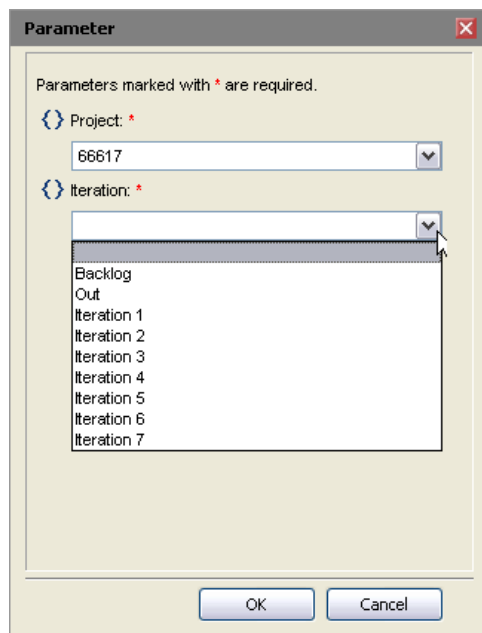
refers to the latest version of the job *campaign2*.

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.

The “[Report prompts](#)” figure illustrates two prompts for a report. The user must specify a project ID and an iteration as input for the report. The list of valid iteration values is limited to a set retrieved from a data source.

Figure 3-1
Report prompts



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 the “[Report prompts](#)” figure, if the list of iteration values depends on the specified project, the project prompt cascades to the iteration prompt. 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.

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 4-1

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 cancelled.

```
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's data sources, variable groups, data sets, and output information.

Input fields

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

Table 4-2
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 4-3
Return Value

Type	Description
reportMetadata	The report metadata which is returned.

Java example

To access the meta-data 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>
        </reportDataSources>
      </reportMetadata>
    </getReportMetadataResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

```

</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 4-4
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 4-5
Return Value

Type	Description
reportMetadata	The specific report metadata which is returned.

Java example

To access specific meta-data for a report:

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

The following sample code returns the *DataSources* meta-data 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#l.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 4-6
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 wishing to run jobs should use the `runJob` operation instead.

Input fields

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

Table 4-7
Fields for `renderJob`

Field	Type/Valid Values	Description
renderSpec	renderSpecification	

Return information

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

Table 4-8
Return Value

Type	Description
string	Job ID returned from <code>renderJob</code> .

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 4-9
Fields for renderSync

Field	Type/Valid Values	Description
renderSpec	renderSpecification	

Return information

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

Table 4-10
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. [For more information, see the topic Jobs in Chapter 3 on p. 12.](#) 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 `setProperty` 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, respectively.
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, respectively.
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:spssec="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="spssec:BinarySecurityToken"
        EncodingType="wsse:Base64Binary"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
        >r00ABXNyAB5jb20uc3Bzcy5zZWV1cm10eS5jb3JILIN1YmplY3QAAAAAAAAAAQIAAIsAC21fc2
        lnbmF0dXJldAACW0JMAA1tX3N1YmplY3REYXRhdAAAtGNvbS9zcHNzL3NIY3VyaXR5L2NvcmlUva
        W50ZXJuYWwvU3ViamVjdERhdGE7eHBwc3IAK2NvbS5zcHNzLnNIY3VyaXR5LmNvcmlUvaW50ZXJu
        YWwvU3ViamVjdERhdGEAAAAAAAAAAQIAB1oACW1faXNBZG1pbkKAEG1fcHJpbmNpcGxlSW5kZXh
        KAA5tX3NlcmlhbE51bWJlckwACW1fyWRkcmVzc3QAEkxqYXZlL2xhbmcvU3RyaW5n00wAE21fYW
        5ub3RhdGlvbklc3NhZ2VxAH4ABUwAdm1fcHJpbmNpcGFsSURzdAAQTGphdmEvdXRpbC9MaXN00
        0wACG1fdXNlcklEcQB+AAV4cAEAAAAAAAAABGT53Xhp0AAkxMjcuMC4wLjFwc3IAJmphaEudXRp
        bC5Db2xsZWV1cm1vZG1maWFibGVmMaXN0/A8IMbXsjaCAAFMAARsaXN0cQB+AAZ4cGAsamF2
        YS51dGlsLkNvbGxlY3Rpb25zJFVubW9kaWZpYWJsZUNvbGxlY3Rpb24ZQgCAy173HglAAU
        wAAWN0ABZMamF2YS91dGlsL0NvbGxlY3Rpb247eHBzcgATamF2YS51dGlsLkFycmF5TGldHiB0
        h2Zx2GdAwABSQAec2l6ZXhwAAAAA3cEAAAAA3QAEc8vdU5hdG12ZS8vYWRtaW50ACMvL2dOYXRp
        dmUvLyQkc2VjdXJpdHkvZXZlcmlvbmVHcm91cHQAKC8vcK5hdG12ZS8vJCRzZWV1cm10eS9yb2x
        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>
          </dataSources>
        </renderSpec>
      </renderSync>
    </soapenv:Body>
  </soapenv:Envelope>
```

```

    <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/2008-04-11.12.45.334-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>spssc:///?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 4-11
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 datasource and credential information is set.

Return information

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

Table 4-12
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 `setProperty` 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.setProperty(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 4-13
Fields for retrieveJobResult

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

Return information

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

Table 4-14
Return Value

Type	Description
<code>jobResult</code>	<code>JobResult</code> contains the <code>RenderedResult</code> and the <code>JobStatus</code> .

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&eventID=0a010a07b5551c63
              000001193e6e8a888039& charset=UTF-8&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&eventID=0a010a07b5551c63
            </ns2:uri>
          </resultLocations>
        </renderedResult>
      </jobResult>
    </retrieveJobResultResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

```

000001193e6e8a888039& charset=UTF-8& 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 4-15
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.
dataSourceLogins	dataSourceLogin[]	Array of DataSourceLogins, where datasource and credential information is set.

Return information

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

Table 4-16
Return Value

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

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 = "spssc://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>

```

```

</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 4-17

Fields for runJob

Field	Type/Valid Values	Description
renderSpec	renderSpecification	

Return information

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

Table 4-18

Return Value

Type	Description
runJobResult	

Java example

Running a job requires the creation of a render specification. [For more information, see the topic The renderJob operation on p. 25.](#)

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("spsssoem.jdbc.sqlserver.SQLServerDriver");
prop[1].setName("REPORT_DATASOURCE_URL");
prop[1].setValue("jdbc:spsssoem: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"
        >r00ABXNyAB5jb20uc3Bzcy5zZWV1cmI0eS5jb3JlLnI1YmplY3QAAAAAAAAAAQIAAIsAC21fc2lnbmF0dXJldAACW0JMAA1tX3N

```

```

</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:spssuem:sqlserver://db1:1433;SelectMethod=cursor;DatabaseName=cq_ecm_data</value>
          </properties>
          <properties>
            <name>REPORT_DATASOURCE_DRIVER_CLASS</name>
            <value>spssuem.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 4-19
Fields for `setJobResult`

Field	Type/Valid Values	Description
<code>eventExecutionID</code>	string	Job ID returned from <code>renderJob</code> .
<code>jobResult</code>	<code>jobResult</code>	<code>JobResult</code> contains the <code>RenderedResult</code> and the <code>JobStatus</code> .

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 4-20
Fields for updateJobStatus

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

The validateDataSource operation

Verifies the report datasource is defined correctly and that valid credentials exist. This is done by attempting a connection to the datasource. 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 4-21
Fields for validateDataSource

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

Return information

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

Table 4-22
Return Value

Type	Description
<code>validateDataSourceResult[]</code>	

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 `setProperty` 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>
```

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. [For more information, see the topic Adding a service reference on p. 48.](#)
2. Configure the web service endpoint. [For more information, see the topic Configuring the web service endpoint on p. 50.](#)
3. Programmatically configure the necessary endpoint behaviors. [For more information, see the topic Configuring endpoint behaviors on p. 51.](#)
4. Exercise the web service. [For more information, see the topic Exercising the service on p. 51.](#)

Note that the IBM SPSS Collaboration and Deployment Services single sign-on implementation is not compatible with Microsoft .NET web services, or the WCF. As a result, single sign-on is not available from these clients.

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 desired 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.

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](http://support.microsoft.com/kb/891386) (<http://support.microsoft.com/kb/891386>) and [326790](http://support.microsoft.com/kb/326790) (<http://support.microsoft.com/kb/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 reference 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;
```

Notices

This information was developed for products and services offered worldwide.

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 character set (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 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, Attention: Licensing, 233 S. Wacker Dr., Chicago, IL 60606, USA.

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.

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

Trademarks

IBM, the IBM logo, [ibm.com](http://www.ibm.com), and SPSS are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at <http://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 of Sun Microsystems, Inc. in the United States, other countries, or both.

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



Index

- app.config files
 - WCF clients, 50
- bindings
 - in WSDL files, 5
- body elements
 - in SOAP messages, 3
- cancelJob operation, 15
- Content Repository service
 - WCF clients, 49
- Content Repository URI service
 - WCF clients, 49
- data sets, 10
- data sources, 10
 - properties, 10
 - validating, 44
- getReportMetadata operation, 16
- getSelectedReportMetadata operation, 21
- getVersion operation, 24
- header elements
 - in SOAP messages, 3
- Holder classes
 - in JAX-WS, 6
- HTTP, 2
- HTTPS, 2
- Java proxies, 6
- JAX-WS, 6
- jobs, 12
 - canceling, 15
 - rendering, 25
 - retrieving results, 34
 - running, 39
 - setting results, 43
 - updating status, 43
- legal notices, 53
- List collections
 - in JAX-WS, 6
- MessageBodyMemberAttribute
 - for WCF clients, 51
- messages
 - in WSDL files, 5
- .NET framework, 48
- .NET proxies, 7
 - parameters, 11
 - PevServices service
 - WCF clients, 49
 - port types
 - in WSDL files, 5
 - Process Management service
 - WCF clients, 49
 - prompt values, 14
 - cascading, 14, 31
 - retrieving, 31, 36
 - protocols
 - in web services, 2
 - proxies, 6
 - Java, 6
 - .NET, 7
 - render specification, 12
 - renderJob operation, 25
 - renderSync operation, 25
 - reporting service
 - stubs, 8
 - Reporting service
 - service endpoint, 8
 - reports, 10
 - retrieving metadata, 16, 21
 - retrieveCascadingPromptValues operation, 31
 - retrieveJobResult operation, 34
 - retrievePromptValues operation, 36
 - runJob operation, 39
 - Scoring service
 - WCF clients, 49
 - service endpoints
 - Reporting service, 8
 - services
 - in WSDL files, 6
 - setJobResult operation, 43
 - single sign-on
 - WCF clients, 48
 - SOAP, 2–3
 - stubs
 - reporting service, 8
 - trademarks, 54
 - types
 - in WSDL files, 4
 - uniform resource identifiers, 13
 - updateJobStatus operation, 43
 - URIs
 - syntax, 13
 - validateDataSource operation, 44
 - variable groups, 11

- versions
 - in URIs, 13
- Visual Studio, 48

- WCF clients, 48, 51
 - endpoint behaviors, 51
 - endpoint configuration, 50
 - limitations, 48
 - service reference, 48–49
 - single sign-on, 48
- web services
 - introduction to web services, 1
 - protocol stack, 2
 - system architecture, 1
 - what are web services?, 1
- web.config files
 - WCF clients, 50
- Windows Communication Foundation, 48
- WSDL files, 2–3
 - accessing, 8
 - bindings, 5
 - messages, 5
 - port types, 5
 - services, 6
 - types, 4
- wsdl.exe, 7
- wsdl2java, 6
- wsimport, 6

- XmlElementAttribute
 - for WCF clients, 51