IBM SPSS Collaboration and Deployment Services
Version 6 Release 0

Search Service Developer's Guide

Note
Before using this information and the product it supports, read the information in "Notices" on page 35.

Product Information

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

Contents

Chapter 1. Introduction to web services

What are web services?

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

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

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

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

Web service system architecture

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

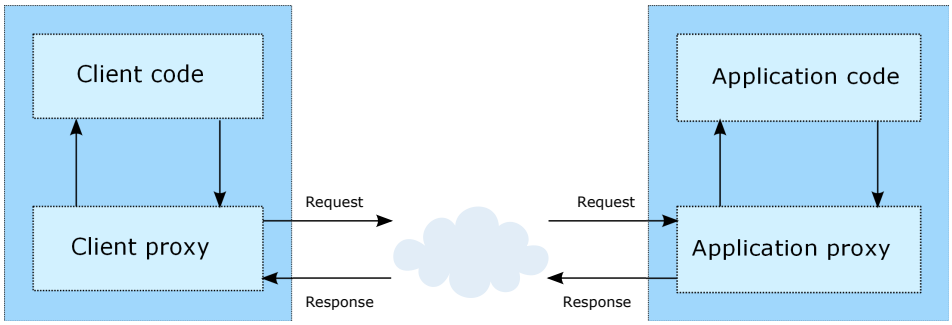

Figure 1. Web service architecture

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

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

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

Web service protocol stack

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

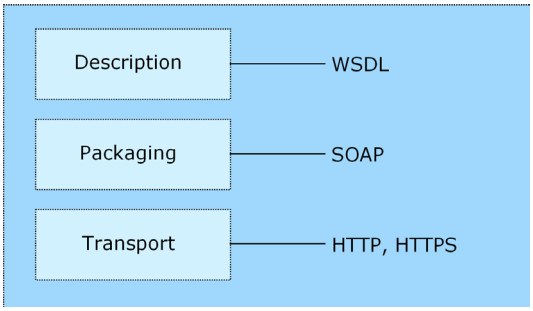

Figure 2. Web service protocol stack

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

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

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

Simple Object Access Protocol

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

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

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

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

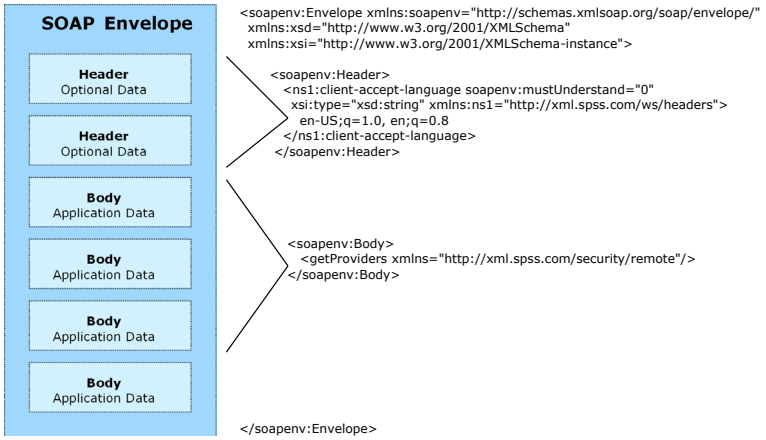

Figure 3. An example SOAP packet

Web Service Description Language

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

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

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

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

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

Types

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

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

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

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

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

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

Messages

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

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

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

Port types

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

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

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

Bindings

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

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

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

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

Services

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

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

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

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

Proxies

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

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

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

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

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

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

Chapter 2. Search Service overview

The Search Service provides a query mechanism for locating content in the repository that meets specified criteria. The query may be a global search for a specified string or a more structured search for information in specific fields. The information contained in the search result set can be customized to be as broad or focused as desired. In addition, large result sets can be returned as individual pages containing a specified number of hits to optimize client performance.

Accessing the Search Service

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

```
http://<host-name>:<port-number>/<context-root>/search-ws/services/Search
```

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

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

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

```
http://cads_server:80/search-ws/services/Search?wsdl
```

Calling Search 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 = "/search-ws/services/Search";
URL url = new URL("http", "cads_server", 80, context);
SearchService service = new SearchServiceLocator();
stub = service.getSearch(url);
```

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

```
stub.search(request);
```

Chapter 3. Search Service concepts

Search requests

The search request provides all input information necessary to perform a search, including the criteria on which to search and the structure used for any returned results. For example, a sample request might be "show me the title and description associated with any file created by the user `bmcgee`." A search request consists of the following parts:

- **Search criteria**. The criteria for which to search and the fields to return for any object meeting the criteria.
- **Page request**. The service returns any found objects as pages, with a specified number of objects on each page. The optional page request indicates a specific page to return, using an internal key to identify the search results to access.

The request may indicate that the returned results be sorted by a designated field and returned in pages of a specified size.

The search request may also define highlighting to include in any returned results. Highlighting involves adding a prefix and a suffix to the search term in the returned value. For example, the results can display asterisks before and after the search term. Alternatively, if the results will be displayed in HTML, the results can use a prefix of ** and a suffix of ** to italicize the search term.

Search criteria

The search criterion defines the conditions that must be met for an object to be returned by a search query. The criterion consists of a search type and an optional object type. The search type is either **parsed string** or **structured**.

A parsed string search consists of a list of terms for which to search. The service parses the string into individual search terms. If any term matches the content of any field for an object, the object is considered a hit and is returned by the service.

In contrast to a string search, a structured search looks for matches using specific object fields. The search request specifies one or more conditions that the object fields must match to be considered a hit. The request defines whether objects classified as hits must match *all* the conditions, or if matching *any* suffices. See the topic “Structured searches” on page 10 for more information.

For both types of search, the object (or component) type serves as a filtering criterion, limiting the search to specific categories of objects. Available types include the following:

- The *ProcessManagement.EventExecution* type, which limits the search to event executions from the Scheduling Server Service
- The *ProcessManagement.Schedule* type, which limits the search to schedules used in Scheduling Server Service
- The *HierarchicalContent.File* type, which limits the search to files and folders in the repository. This is the default used in the absence of a specified type

Searches can also be limited to specific file types using a MIME type filter. For example, to limit a search to IBM SPSS Statistics syntax files, include a MIME type filter with a value of *application/x-vnd.spss-spss-syntax* in the search request.

The search criterion can also define a language to search within. Fields being searched include an optional language characteristic, associating the field value with a language designation. If the criterion specifies a language, the search includes only fields where the optional language characteristic is the specified value or null.

Structured searches

Structured searches include conditions that must be met for an object to be returned by the search query. For example, one type of condition involves the specification of a date range for returned objects. In this case, the search request specifies the name of a date/time field for objects, such as *$$prms/idx_next_sched_time*. The request can define the following:

- A starting date or time to return objects with a field value after that criterion
- An ending date or time to return objects with a field value before that criterion
- Both starting and ending dates or times to return objects with a field value between the criteria

Another type of condition limits the search to descendant objects of a specified path. The service ignores objects that are not descendants, resulting in a faster search than one without this condition.

The final type of condition that can be included in a structured search involves the definition of query items. A query item specifies a specific field name and the value for which to search in that field. The request can specify that the value be parsed into individual terms with a hit occurring for objects that match either any or all of the terms. Alternatively, hits can be restricted to those objects whose field value matches the specified criterion exactly or whose field value begins with the criterion. A structured search request can include any number of query items, allowing searches to be based on multiple object fields. The list of valid field names to include in a query item corresponds to the list of field names used when specifying return fields. See the topic "Return fields" on page 11 for more information.

For example, to find Data Provider Definitions that use or are dependent on a specific Application View, create a query item for the field:

```
$$pev/application_view_uri_field_name
```

Specify a value of:

```
spsscr:///?id=<OBJECTID>
```

Replace <*OBJECTID*> with the ID of the Application View.

Alternatively, to find Data Provider Definition items that support the *Analytic* environment, create a query item for the field:

```
$$pev/environment_field_name
```

Specify a value of *Analytic*.

For more complex searches involving combinations of fields, the query item can specify a free-form string similar to a `WHERE` clause in SQL. The string may contain the following:

- Field names (in single quotation marks). See the table for a list of available fields.
- Field values (in single quotation marks)
- The operators `=`, `BETWEEN`, `AND`, and `OR`
- Parentheses controlling the order of operations

The basic format of the search string is as follows:

```
('field-name' = 'value') OR ('field-name' = 'value') AND ('field-name' = 'value')
```

For date and time values, the format is `yyyy-mm-dd hh:mm:ss.fffffffff`. The complete time specification may be omitted if it is not necessary. If the time specification is needed, milliseconds can be excluded. For example, the search string could be:

```
('$$repository/title_field_name' = 'MyObject' OR
'$$repository/title_field_name' = 'YourObject') AND
'$$repository/version_created_date' BETWEEN '2007-10-01 14:00:00' AND '2007-10-31'
```

Table 1. Valid fields for free-form queries.

Field name	Description
$$repository/author_field_name	Author
$$repository/title_field_name	Title
$$repository/description_field_name	Description
$$repository/version_created_by_field_name	Version Created By
$$repository/version_created_date_field_name	Version Created On
$$repository/version_label_field_name	Label
$$repository/keyword_field_name	Keyword
$$search/topics	Topic
$$search/mimetype	MIME type
$$search/expiration	Expiration Date
$$search/sortStamp	Last Modified

Return fields

A search request can return a variety of descriptive properties about found objects. However, in many cases, only a subset of the available properties is needed. The search request defines this subset by specifying the names of the properties to return, or the **return fields**. A return field corresponds to a specific property of a found object, such as its title. These fields can be classified into the following categories: Search, Content Repository, Event Execution, Schedule, and Enterprise View.

Search fields report information about the result set, such as how many versions of a file match the search criteria and how well a returned object matches the criteria. In general, search fields should only be used as return fields. They should not be used within query items for structured searches.

Table 2. Search fields

Field name	Description
$$search/parentURI	URI of the containing folder
$$search/parentFolder	Name of the parent folder
$$search/parentFolderPath	Full path of the parent folder. It is often best to display the *$$search/parentFolder* value and use the *$$search/parentFolderPath* value in a tooltip or a properties dialog due to the value's length.
$$search/sortVersionCount	Number of versions of the file in the returned result set. This may not be the number of versions of the file that exist in the repository, as some file versions may not satisfy the search criterion.
$$search/sortStamp	Last changed *dateTime*
$$search/expiration	Expiration *dateTime*. This field will be empty if no expiration has been set.
$$search/objectID	Identifier of the object
$$search/mimetype	MIME type of the object
$$search/mimetypeName	Name of the MIME type
$$search/topics	Topic names associated with the object
$$search/topicsPath	Fully qualified topic paths associated with the object
$$search/relevance	Number of matches to the search criterion. This value is useful only when the search criterion includes multiple fields and the OR operator.

Content Repository fields report information about files and folders stored in the repository. These properties include the file or folder title, description, and author. Return fields also offer access to specific information for IBM SPSS Modeler files, such as field names or node labels. To work with IBM SPSS Modeler files, the appropriate adapters must be installed. For more information, see the IBM SPSS Modeler documentation.

Table 3. Content Repository fields

Field name	Description
$$repository/author_field_name	Author of the object
$$repository/title_field_name	Title of the object
$$repository/description_field_name	Description of the object
$$repository/mimetype_field_name	MIME type of the object
$$repository/object_last_modified_by_field_name	User who last modified the object
$$repository/version_created_by_field_name	User who created the version of the object
$$repository/version_created_date_field_name	Date the object version was created
$$repository/version_label_field_name	Label for the object version
$$repository/clementine_fieldname.fieldname	Field name
$$repository/clementine_fieldname.modelalgorithm	Model algorithm
$$repository/clementine_fieldname.modelcategory	Model category. Valid values include the following: *AnomalyDetection*, *Approximation*, *Association*, *AttributeImportance*, *Categorize*, *Classification*, *Clustering*, *ConceptExtraction*, *Reduction*, *Sequence*, *TimeSeries*, and *Unknown*.
$$repository/clementine_fieldname.nodelabel	Node label
$$repository/clementine_fieldname.nodename	Node name

Event Execution fields report information about job executions. These properties include the execution time and state.

Table 4. Event Execution fields

Field name	Description
$$prms/idx_event_cluster_id	Identifier of the event cluster
$$prms/idx_exec_run_time	Total execution time
$$prms/idx_exec_start_time	Start time of the execution
$$prms/idx_exec_state	Execution state
$$prms/idx_exec_success	Indicator of whether or not the execution resulted in a success
$$prms/idx_job_name	Job name
$$prms/idx_job_version_label	Label for the job version
$$prms/idx_mime	MIME type of the execution
$$prms/idx_next_sched_time	Next scheduled execution time for the job

Schedule fields report information about job schedules. These properties include the schedule frequency and start date.

Table 5. Schedule fields

Field name	Description
`$$prms/idx_sched_enabled`	Indicator of whether or not the schedule is enabled
`$$prms/idx_sched_frequency`	Frequency defined by the schedule
`$$prms/idx_sched_start_date`	Start date defined by the schedule
`$$prms/idx_sched_version_label`	Version label associated with the schedule

Enterprise View fields report information about Application View and Data Provider Definition resources stored in the repository. These properties include the uniform resource identifier for the Application View and the view environment.

Table 6. Enterprise View fields

Field name	Description
`$$pev/application_view_uri_field_name`	URI of the Application View
`$$pev/environment_field_name`	A named environment, either *Analytic* or *Operational*

Using return fields involves including the names of the fields to return in a search request. For example, to return file titles and descriptions, create return fields for *$$repository/title_field_name* and *$$repository/description_field_name* in the search request. In the absence of return field specifications, a search query returns all fields related to the matching objects.

Expired objects

The expiration date for an object version specifies the date after which the version of the object is no longer actively in use. When a version of an object expires, it is only visible to its owner and administrators. Other users cannot view the version. Note that expiration does not imply deletion from the repository. An expiration date merely indicates the point in time when an object version will be hidden from general users.

The Search Service can include expired object versions in result sets. The search criteria defines which expired versions are returned. The result set can include the following:

- All expired object versions owned by the user making the search request
- All versions having an expiration date older than a specified date
- All versions having an expiration date newer than a specified date

Search results

Search results consist of three primary pieces, as follows:

- **Object fields.** The names of the object-level fields returned by the search.
- **Version fields.** The names of the version-level fields returned by the search.
- **Hits.** Information about objects returned by the search.

The object and version fields listed in the results correspond to the list of return fields defined in the search request.

Information for each hit includes the object identifier, the path to the object in the repository, and the timestamp associated with the last modification for the object. In addition, the results include the value for each object field and version field for the returned object. The order of the returned field values corresponds to the order of the object and version fields lists. For example, if the list of object fields begins with *$$repository/title_field_name* followed by *$$repository/author_field_name*, the first returned field

value for a hit corresponds to the object title and the second is the object author. If the search request includes a highlighting definition, each returned hit includes both the raw return value and a highlighted return value.

Page results

To organize object and version information, the Search Service can return results in a page consisting of a two-level tabular structure in which rows can contain other rows. The values in a main row of the table correspond to object-level information for a hit, with contained rows reporting version-level information. The table illustrates this structure for search results containing two hits.

Table 7. Example two-level table

Hit	Object Title	Object Author	Version Label	Version creation date
1	*E01_FindDuplicates.str*	Brian McGee	Test	2006-07-24 11:13:25.109
			Production	2006-07-30 12:03:45.111
2	*M06_Matching.str*	Brian McGee	Test	2006-07-24 11:14:27.453
			Production	2006-08-10 10:14:56.154

The Object Title column of the table identifies the name of the returned file for each hit. The Object Author column identifies the file author. The next two columns report version-level information, in this case the version labels and creation dates, for each object hit. These columns can contain multiple rows, referred to as **child rows**, for each hit. The version-level columns are referred to as **child columns** because the values for these columns appear in child rows for the hits.

Page results information is returned by the `search2.5` operation and can be classified into four categories: general metadata, column, row, and navigator. General metadata includes the following:

- The total number of hits in the result set
- The maximum number of hits for any page
- The page number for the current page
- The column used for sorting the hits
- The sort order as *ascending* or *descending*
- A client key value, which is an internal identifier used to synchronize requests for specific pages

Information for columns and child columns consists of the following:

- The display name for the column, such as *Title* or *Author*
- The internal field name for the column. See the topic "Return fields" on page 11 for more information.
- An indicator of the type of information reported in the column. The type is *string*, *stamp*, or *number*.

Row information returned by the Search Service includes the following:

- The row number
- The URI for the object represented by the row
- The MIME type of the object
- The values for the individual cells in an order corresponding to the order of the returned columns. For example, the second value for each hit in the table would be the value for the second column, *Author*. For stamp data, the values also include the XML dateTime format.
- Child rows

Information for child rows is similar to that for the main row. However, the URI for a child row refers to a specific version of the object.

Navigators serve to facilitate the creation of user interfaces to display the results for multiple pages. This information consists of characteristics of each page in the results, as well as data for the preceding and following pages.

Chapter 4. Operation reference

The getServerTimeZone operation

For internal use only. Returns the time zone for the repository server, allowing clients in other time zones to adjust times as needed. For searches to return meaningful results, time-based search requests should account for any time zone differences.

Return information

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

Table 8. Return Value.

Type	Description
string	Defines an output message for the web service implementation. For internal use only.

Java example

The following example retrieves the time zone for the repository server.

```
System.out.println("Server Time Zone =  " + stub.getServerTimeZone());
```

SOAP request example

Client invocation of the `getServerTimeZone` 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>
    <getServerTimeZone xmlns="http://xml.spss.com/search/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getServerTimeZone` 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>
    <getServerTimeZoneResponse xmlns="http://xml.spss.com/search/remote">
      <serverTimeZone>CDT</serverTimeZone>
    </getServerTimeZoneResponse>
  </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 9. Return Value.

Type	Description
string	Defines an output message for the web service implementation. For internal use only.

Java example

The following example creates a `WebServiceConnections` object containing general connection information for accessing the web services available in IBM SPSS Collaboration and Deployment Services. The `getSearch` method of this object returns a stub for the Search Service, from which the `getVersion` operation is called to return the service version.

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://xml.spss.com/search/remote"/>
  </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://xml.spss.com/search/remote">
      <version>4.20.000</version>
    </getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The search operation

The `search` operation is deprecated. Use the `search2.5` operation to retrieve information about objects in the repository.

The search2.5 operation

Retrieves information about objects in the repository that match specified criteria, returning hits in a two-level table containing both object and version information.

Input fields

The following table lists the input fields for the `search2.5` operation.

Table 10. Fields for search2.5.

Field	Type/Valid Values	Description
SearchRequest	SearchRequest	Input for performing a search.

Return information

The following table identifies the information returned by the search2.5 operation.

Table 11. Return Value.

Type	Description
pageResult	

Java example

Performing a search using the search2.5 operation requires the specification of a search request. In general, the steps involved in creating a request are the following:

1. Create a SearchRequest object.
2. Define the return fields.
3. Specify the search criteria. The definition of the criteria depend on whether the search is a simple string search or a structured search.
4. Specify any additional metadata for the search.

This example defines a structured search for all IBM SPSS Modeler streams in the repository.

A Structured object defines the specific structure for which to search. In this case, a QueryItem object defines the value to search for as *application/x-vnd.spss-clementine-stream* and the field to be searched as *$$repository/mimetype_field_name*. The setQueryItem operation assigns this query to the Structured object.

An InitialSearch object specifies the criteria for which to search. The setStructured method assigns the Structured object as the criteria and the setInitialSearch method assigns this criteria to the request object. The setSortField method specifies title as the field on which to sort and the setSortOrder method defines the order as ascending. Finally, the search25 operation processes the request and returns the results as a PageResult object.

```
SearchRequest request = new SearchRequest();
//set return fields
String[] retFields = {
  "$$repository/title_field_name",
  "$$repository/author_field_name",
  "$$repository/mimetype_field_name"
};
request.setReturnField(retFields);

//set initial search
Structured structCriteria = new Structured();
QueryItem qItem = new QueryItem();
qItem.setField("$$repository/mimetype_field_name");
qItem.setValue("application/x-vnd.spss-clementine-stream");
structCriteria.setQueryItem(qItem);

InitialSearch initialSearch = new InitialSearch();
initialSearch.setComponentType("HierarchicalContent.File");
initialSearch.setStructured(structCriteria);
request.setInitialSearch(initialSearch);

//set sort field and order
request.setSortField("$$repository/title_field_name");
SortOrderType orderType = SortOrderType("ascending");
request.setSortOrder(orderType);

PageResult results = stub.search25(request);
```

To specify optional handling of expired object versions, use an ExpirationControl object. Supply the setIncludeIfOwner method with a boolean indicating whether or not to include expired versions owned by the user making the search request. Provide the setIncludeNullExpired method with a boolean

indicating whether or not to include versions with no defined expiration date. Supply the `setStarting` and `setEnding` methods with a `Calendar` object defining a cutoff date to include expired versions newer or older than the specified date.

Use the `setExpirationControl` operation to assign the `ExpirationControl` object to the `InitialSearch` object. Note that the service currently also supports assigning the `ExpirationControl` object to the `Structured` object, but this behavior is deprecated in favor of using the `InitialSearch` object.

For a free-form query, create a `FreeFormQueryItem` object and assign the query string to it using the `setValue` method. Add the query item to the `Structured` object using the `setFreeFormQueryItem` method. The following sample searches for resources created within a specified range.

```
Structured structured = new Structured();
FreeFormQueryItem queryItem = new FreeFormQueryItem();
queryItem.setValue("'$$repository/version_created_date_field_name'
   BETWEEN '2007-10-16' AND '2007-11-01'");
structured.setFreeFormQueryItem(queryItem);

InitialSearch initialSearch = new InitialSearch();
initialSearch.setStructured(structured);
SearchRequest request = new SearchRequest();
request.setInitialSearch(initialSearch);
PageResult results = stub.search25(request);
```

SOAP request example

Client invocation of the `search2.5` 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 xsi:type="xsd:string">validUser</wsse:Username>
        <wsse:Password xsi:type="xsd:string">password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
    <ns1:client-accept-language soapenv:mustUnderstand="0" xsi:type="xsd:string"
    xmlns:ns1="http://xml.spss.com/ws/headers">en-US;q=1.0, en;q=0.8</ns1:client-accept-language>
  </soapenv:Header>
  <soapenv:Body>
    <search2.5 xmlns="http://xml.spss.com/search/remote">
      <ns2:SearchRequest sortField="$$repository/title_field_name"
        sortOrder="ascending" xmlns:ns2="http://xml.spss.com/search">
      <ns2:returnField>$$repository/title_field_name</ns2:returnField>
      <ns2:returnField>$$repository/author_field_name</ns2:returnField>
      <ns2:returnField>$$repository/mimetype_field_name</ns2:returnField>
      <ns2:initialSearch>
        <ns2:parsableQueryString>Brian</ns2:parsableQueryString>
      </ns2:initialSearch>
      </ns2:SearchRequest>
    </search2.5>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `search2.5` operation call by sending a SOAP response message containing the results. An example of such a message follows.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <search2.5Response xmlns="http://www.spss.com/pes/pagerr">
      <ns1:pageResult hitCount="2" pageSize="25" pageNumber="1" clientKey="5eeec5dbga"
        sortColumn="$$repository/author_field_name" sortOrder="descending"
        xmlns:ns1="http://www.spss.com/pes/pager">
        <ns1:column display="Author" fieldName="$$repository/author_field_name" colType="string"/>
        <ns1:column display="Keyword" fieldName="$$repository/keyword_field_name" colType="string"/>
        <ns1:column display="Object Last Modified By"
```

```
  fieldName="$$repository/object_last_modified_by_field_name" colType="string"/>
<ns1:column display="Title" fieldName="$$repository/title_field_name" colType="string"/>
<ns1:childColumn display="Label" fieldName="$$repository/version_label_field_name"
  colType="string"/>
<ns1:childColumn display="repository/version_created_by_field_name"
  fieldName="$$repository/version_created_by_field_name" colType="string"/>
<ns1:childColumn display="Version Creation Date"
  fieldName="$$repository/version_created_date_field_name" colType="string"/>
<ns1:row rowNumber="1"
  uri="spsscr://pes_server:80/?id=0a0a4aac00072ffb0000010ca13be58780f4">
  <ns1:cell>
    <ns1:value><display xmlns="">Brian Magee</display></ns1:value>
  </ns1:cell>
  <ns1:cell>
    <ns1:value><display xmlns="">data understanding</display></ns1:value>
  </ns1:cell>
  <ns1:cell>
    <ns1:value><display xmlns="">admin</display></ns1:value>
  </ns1:cell>
  <ns1:cell>
    <ns1:value><display xmlns="">E01_FindDuplicates.str</display></ns1:value>
  </ns1:cell>
  <ns1:childRow rowNumber="1"
    uri="spsscr://pes_server:80/?id=0a0a4aac00072ffb0000010ca13be58780f4#m.0:2006-07-24%2011:13:25.109">
    <ns1:cell>
      <ns1:value><display xmlns="">Production</display></ns1:value>
      <ns1:value><display xmlns="">Production</display></ns1:value>
    </ns1:cell>
    <ns1:cell>
      <ns1:value><display xmlns="">admin</display></ns1:value>
      <ns1:value><display xmlns="">admin</display></ns1:value>
    </ns1:cell>
    <ns1:cell>
      <ns1:value><display xmlns="">2006-07-24 11:13:25.109</display></ns1:value>
      <ns1:value><display xmlns="">2006-07-24 11:13:25.109</display></ns1:value>
    </ns1:cell>
    <ns1:cell>
      <ns1:value><display xmlns="">Production</display></ns1:value>
      <ns1:value><display xmlns="">Production</display></ns1:value>
    </ns1:cell>
    <ns1:cell>
      <ns1:value><display xmlns="">admin</display></ns1:value>
      <ns1:value><display xmlns="">admin</display></ns1:value>
    </ns1:cell>
    <ns1:cell>
      <ns1:value><display xmlns="">2006-07-24 11:13:25.109</display></ns1:value>
      <ns1:value><display xmlns="">2006-07-24 11:13:25.109</display></ns1:value>
    </ns1:cell>
  </ns1:childRow>
</ns1:row>
<ns1:row rowNumber="2"
  uri="spsscr://pes_server:80/?id=0a0a4aac00072ffb0000010ca13be58781d3">
  <ns1:cell>
    <ns1:value><display xmlns="">Brian Magee</display></ns1:value>
  </ns1:cell>
  <ns1:cell>
    <ns1:value><display xmlns="">modeling</display></ns1:value>
    <ns1:value><display xmlns="">evaluation</display></ns1:value>
  </ns1:cell>
  <ns1:cell>
    <ns1:value><display xmlns="">admin</display></ns1:value>
  </ns1:cell>
  <ns1:cell>
    <ns1:value><display xmlns="">M06_Matching.str</display></ns1:value>
  </ns1:cell>
  <ns1:childRow rowNumber="1"
    uri="spsscr://pes_server:80/?id=0a0a4aac00072ffb0000010ca13be58781d3#m.0:2006-07-24%2011:14:27.453">
    <ns1:cell>
      <ns1:value><display xmlns="">Production</display></ns1:value>
      <ns1:value><display xmlns="">Production</display></ns1:value>
    </ns1:cell>
    <ns1:cell>
      <ns1:value><display xmlns="">admin</display></ns1:value>
      <ns1:value><display xmlns="">admin</display></ns1:value>
    </ns1:cell>
    <ns1:cell>
      <ns1:value><display xmlns="">2006-07-24 11:14:27.453</display></ns1:value>
      <ns1:value><display xmlns="">2006-07-24 11:14:27.453</display></ns1:value>
    </ns1:cell>
    <ns1:cell>
      <ns1:value><display xmlns="">Production</display></ns1:value>
      <ns1:value><display xmlns="">Production</display></ns1:value>
    </ns1:cell>
    <ns1:cell>
```

```
              <ns1:value><display xmlns="">admin</display></ns1:value>
              <ns1:value><display xmlns="">admin</display></ns1:value>
            </ns1:cell>
            <ns1:cell>
              <ns1:value><display xmlns="">2006-07-24 11:14:27.453</display></ns1:value>
              <ns1:value><display xmlns="">2006-07-24 11:14:27.453</display></ns1:value>
            </ns1:cell>
          </ns1:childRow>
        </ns1:row>
        <ns1:navigator>
          <page display="1" selector="1" current="true" xmlns=""/>
        </ns1:navigator>
      </ns1:pageResult>
    </search2.5Response>
  </soapenv:Body>
</soapenv:Envelope>
```

Chapter 5. JAX-WS clients

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

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

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

Generating a JAX-WS client

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

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

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

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

Packaging a JAX-WS client

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

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

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

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

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

Configuring a JAX-WS client

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

The following example demonstrates obtaining the WSDL from the server:

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

The URL includes the host and port for your server.

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

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

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

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

Next, access the service endpoint:

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

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

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

SOAPHandler example

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

The following code provides an example of this object.

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

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

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

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

    // prefixes
```

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

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

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

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

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

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

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

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

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

    return languageHeader;
}

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

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

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

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

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

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

        return securityHeader;
    }

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

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

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

}
```

Exercising web services from JAX-WS clients

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

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

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

Chapter 6. Microsoft® .NET Framework-based clients

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

1. Add a Service Reference. See the topic "Adding a service reference" for more information.
2. Configure the web service endpoint. See the topic "Configuring the web service endpoint" on page 28 for more information.
3. Programmatically configure the necessary endpoint behaviors. See the topic "Configuring endpoint behaviors" on page 29 for more information.
4. Exercise the web service. See the topic "Exercising the service" on page 29 for more information.

Adding a service reference

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

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

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

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

Service reference modifications

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

To modify a service reference:

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

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

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

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

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

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

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

Configuring the web service endpoint

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

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

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

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

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

Configuring endpoint behaviors

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

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

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

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

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

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

Exercising the service

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

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

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

The corrected code is as follows:

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

Single sign-on authentication

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

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

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

Chapter 7. Message header reference

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

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

Security headers

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

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

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

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

Table 12. SOAP header namespaces

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

Security element

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

Table 13. Attributes of `wsse:Security`

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

UsernameToken element

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

Table 14. Attributes of wsse:UsernameToken

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

Table 15. Child elements of wsse:UsernameToken

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

BinarySecurityToken and BinarySecuritySSOToken elements

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

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

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

Table 16. Attributes of wsse:BinarySecurityToken

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

Table 16. Attributes of wsse:BinarySecurityToken (continued)

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

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

The client-accept-language element

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

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

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

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

HTTP headers

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

Table 17. HTTP headers

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

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

Notices

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

Trademarks

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

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

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

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

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

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

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

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

Glossary

Index

Special characters

A

B

C

E

G

H

J

L

M

N

O

P

R

S

T

U

V

W

X

Printed in USA