

IBM SPSS Collaboration and Deployment Services
Version 8 Release 2

*User Preferences Service Developer's
Guide*

IBM

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 8, release 2, modification 1 of IBM SPSS Collaboration and Deployment Services and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 2000, 2019.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. Introduction to web services 1

What are web services?	1
Web service system architecture	1
Web service protocol stack.	2
Simple Object Access Protocol	2
Web Service Description Language	3
Proxies	5

Chapter 2. User Preferences Service

overview 7

Accessing the User Preferences Service	7
Calling User Preferences Service operations	7

Chapter 3. User preferences concepts. 9

Preference items	9
Preference item values	9

Chapter 4. Operation reference 11

Client port type	11
Operation reference.	11
Manager port type	16
Operation reference.	16

Chapter 5. JAX-WS clients 21

Generating a JAX-WS client	21
Packaging a JAX-WS client	21
Configuring a JAX-WS client	21
SOAPHandler example	22
Exercising web services from JAX-WS clients	24

Chapter 6. Microsoft .NET Framework-based clients 25

Adding a service reference	25
Service reference modifications	25
Configuring the web service endpoint	26
Configuring endpoint behaviors	27
Exercising the service	27
Single sign-on authentication	28

Chapter 7. Message header reference 29

Security headers.	29
---------------------------	----

Security element.	29
UsernameToken element	30
BinarySecurityToken and BinarySecuritySSOToken elements	30
The client-accept-language element	31
HTTP headers	31

Appendix. Deprecated features 33

Notices 35

Privacy policy considerations	36
Trademarks	37

Glossary 39

A.	39
B.	39
C.	39
D.	40
E.	40
F.	41
G.	41
I.	41
J.	41
K.	41
L.	42
M.	42
N.	42
O.	42
P.	43
R.	43
S.	43
T.	44
U.	44
V.	44
W.	44
X.	44

Index 45

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® or JBoss Application 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.

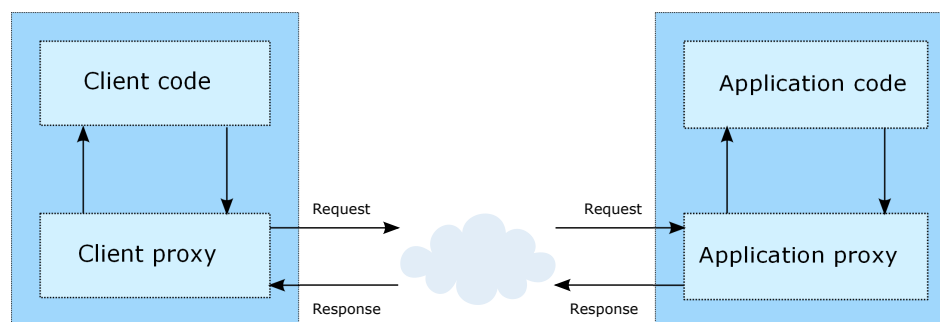


Figure 1. Web service architecture

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.

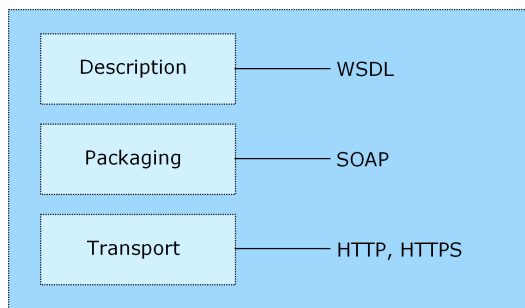


Figure 2. Web service protocol stack

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:

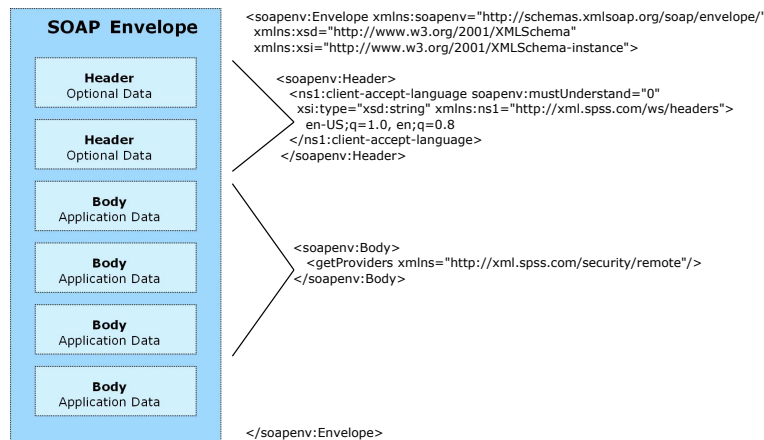


Figure 3. An example SOAP packet

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

```



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

In this case, the transport attribute of the `wsdl:soap: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">
    <wsdl:soap: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. User Preferences Service overview

The User Preferences Service allows users of a client application to store and retrieve individual values for preference items defined in the system, permitting a customized experience for each user. For example, a user can specify his or her e-mail address and have it persist across sessions. In addition, the service includes administrative functionality for managing preference items, such as identifying which users have specified preference values.

Accessing the User Preferences Service

To access the functionality offered by the User Preferences Service, create a client application using the proxy classes generated by your preferred web service tool. The service includes two port types, *Client* and *Manager*, with the following endpoints:

```
http://<host-name>:<port-number>/<context-root>/userpref-ws/services/Client
http://<host-name>:<port-number>/<context-root>/userpref-ws/services/Manager
```

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/userpref-ws/services/Client?wsdl
http://cads_server:80/userpref-ws/services/Manager?wsdl
```

Calling User Preferences 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 = "/userpref-ws/services/Client";
URL url = new URL("http", "cads_server", 80, context);
UserPref service = new UserPrefServiceLocator();
stub = service.getUserPref(url);
```

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

```
stub.getValues(request);
```

Chapter 3. User preferences concepts

Preference items

A preference item corresponds to an internal system setting that is allowed to vary across users. For example, every user has an email address but the individual values of the addresses can differ. Preference items available in the system are defined by the following properties:

- **Definition ID.** An internal identifier used to reference the item, such as `email/default`.
- **Name.** The localized name of the item, such as `Email Address`. The name can be used in client user interfaces that expose the item.
- **Component.** The localized name of the component that defined the item, such as `General Preferences`. The component name may be used to group similar items in lists.
- **Type.** The type of information stored in the item, which can have implications on the interface used to edit the item value. Some types include constraints on the permissible values for the item.

Table 1. Item types.

Type	Description	Constraints
<code>freeForm</code>	A string.	Minimum and maximum number of characters in the string.
<code>email</code>	An email address compliant with RFC-822 .	
<code>password</code>	A security-sensitive string. Values for items of this type are often masked in user interfaces.	
<code>choices</code>	A set of predefined choices.	The number of selections from the set. Valid values include <i>one</i> , <i>many</i> , or at least one (<i>some</i>).
<code>bool</code>	A binary choice, such as Yes/No or True/False.	
<code>int</code>	An integer.	The minimum and maximum values for the integer.
<code>accessControlList</code>	A set of <code>principalID/permissionID</code> pairs.	

An item may also include an HTML string designed to serve as help text in user interfaces that expose the item.

A preference item may be defined to be **backed** by either a configuration item or a user directory item. If the preference item itself has no specified value, the value of the backing item will be used in its place. For example, getting the value for the `email/default` item may return a value from a user directory, such as Active Directory, if the user has not specifically set his or her email address. If neither the preference item nor its backing item have values, no value will be returned.

Preference item values

Each user can have a different value for a preference item. That value has an internal address that consists of the following attributes:

- **Definition ID.** The identifier for the item.
- **Principal ID.** An internal identifier for the user associated with the value, including both the directory containing the user and the user name. This string uniquely identifies the principal in the enterprise as viewed from the IBM SPSS Collaboration and Deployment Services system.

- **Value key.** Some item types, such as *choices*, may have multiple user-named values. For these items, the address includes a value key denoting the value name. If the item allows only one value, the address omits the value key.

References to an item value that omit the principal ID from the address default to the identifier for the current principal. Consequently, the principal ID is only required when one user needs to access another user's item values, such as when an administrator modifies preference item values for a user. However, in that case, the person making the changes must be assigned to a role containing the *userPref/Admin* action.

The User Preferences Service includes operations for assigning, retrieving, and deleting item values based on their addresses.

Chapter 4. Operation reference

Client port type

The Client port type includes operations used for working with preference item values, such as retrieving, setting, and deleting values.

Operation reference

The deleteValues operation

Removes one or more values for user preference items. This would typically be used to clean up obsolete user preferences. For example, a system defined process might determine users or content that no longer exist and remove related user preference values.

Input fields

The following table lists the input fields for the deleteValues operation.

Table 2. Fields for deleteValues.

Field	Type/Valid Values	Description
deleteValues	deleteValues	Identifier for the values to delete, composed of a definition ID, a principal ID, and a values key.

Return information

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

Table 3. Return Value.

Type	Description
string	Total number of values deleted.

Java example

Removing the values assigned to a user preference item involves creating a DeleteValues object. Use the setDefID method to define the definition ID for the item. Supply this object to the deleteValues operation.

```
DeleteValues deleteValues = new DeleteValues();
deleteValues.setDefID("com.spss.security/upDefaultACL");
stub.deleteValues(deleteValues);
```

SOAP request example

Client invocation of the deleteValues 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>
  </soapenv:Header>
  <soapenv:Body>
    <deleteValues>
      <deleteValues>
        <deleteValues>
          <deleteValues>
            <deleteValues>
              <deleteValues>
                <deleteValues>
                  <deleteValues>
                    <deleteValues>
                      <deleteValues>
                        <deleteValues>
                          <deleteValues>
                        </deleteValues>
                      </deleteValues>
                    </deleteValues>
                  </deleteValues>
                </deleteValues>
              </deleteValues>
            </deleteValues>
          </deleteValues>
        </deleteValues>
      </deleteValues>
    </soapenv:Body>
</soapenv:Envelope>
```

```

    </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>
  <deleteValues xmlns="http://xml.spss.com/pes/userPref/remote">
    <deleteValues xmlns="http://www.spss.com/pes/userPref" defID="com.spss.security/upDefaultACL"/>
  </deleteValues>
</soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a deleteValues 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>
    <deleteValuesResponse xmlns="http://xml.spss.com/pes/userPref/remote">
      <ns1:deleteValueResponse xmlns:ns1="http://www.spss.com/pes/userPref">
        <ns1:response>1</ns1:response>
      </ns1:deleteValueResponse>
    </deleteValuesResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

The getValueByDefId operation

Returns current user preference item value for a specified item.

Input fields

The following table lists the input fields for the getValueByDefId operation.

Table 4. Fields for getValueByDefId.

Field	Type/Valid Values	Description
defIdRequest	defIdRequest	

Return information

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

Table 5. Return Value.

Type	Description
defIdResponse	

The getValues operation

Returns current user preference item values for one or more items based on partial or fully qualified value addresses. If the address omits the definition ID for an item, the operation returns values for all items for the user. The request also includes a flag indicating whether or not the values from the hierarchy and whether configuration and user preference macros should be resolved.

Input fields

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

Table 6. Fields for `getValues`.

Field	Type/Valid Values	Description
<code>valueRequest</code>	<code>valueRequest</code>	Identifier for the values to retrieve, composed of a definition ID, a principal ID, and a values key.

Return information

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

Table 7. Return Value.

Type	Description
<code>valuesResponse</code>	Preference values corresponding to the request parameters.

Java example

To retrieve values for a user preference item, create a `ValueRequest` object. Use the `setDefID` method to define the definition ID for the item. Supply this object to the `getValues` operation.

The web service returns the values in a `ValuesResponse` object. Use the `getValueArray` method to return a `ValueArray` object, from which individual values can be returned using the `getValue` method. The following sample retrieves the value for the `email/default` item:

```
String emailAddress = "";
ValueRequest valueRequest = new ValueRequest();
valueRequest.setDefID("email/default");
ValuesResponse valuesResponse = stub.getValues(valueRequest);
```

SOAP request example

Client invocation of the `getValues` 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>
    <getValues xmlns="http://xml.spss.com/pes/userPref/remote">
      <valueRequest xmlns="http://www.spss.com/pes/userPref" defID="email/default"/>
    </getValues>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getValues` 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>
    <getValuesResponse xmlns="http://xml.spss.com/pes/userPref/remote">
      <ns1:valuesResponse xmlns:ns1="http://www.spss.com/pes/userPref">
        <ns1:valueArray>
          <ns1:value ns1:component="$$userPref/grpGeneral" ns1:name="$$userPref/emailName"
            ns1:defID="email/default" ns1:principalID="//uNative//validUser">
            validUser@company.com
          </ns1:value>
        </ns1:valueArray>
      </ns1:valuesResponse>
    </getValuesResponse>
  </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 8. Return Value.

Type	Description
string	The version of the web service.

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

```

The setValue operation

Assigns a value to a user preference item using a fully qualified value address. The User Preferences Service validates the value against any constraints defined for the item. If the value is invalid, the operation returns an exception without saving the value. To report the problem to the user, use the information returned by the exception's getFaultReason method. For proxies generated by Axis, casting to an AxisFault may be needed:

```
String displayMessage = ((AxisFault)e).getFaultReason()
```

Input fields

The following table lists the input fields for the setValue operation.

Table 9. Fields for setValue.

Field	Type/Valid Values	Description
configMacro	configMacro	Refers to a PES configuration item. Value of the item is the configuration key.
prefMacro	prefMacro	Child element within a value.

Return information

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

Table 10. Return Value.

Type	Description
string	General status message. This message should not be interpreted by client applications as the value may change in future releases. It currently serves merely as a placeholder for the protocol.

Java example

To specify a value for a user preference item, create a SetValue object. Use the setDefID method to specify the definition ID for the item being set. The setContent method defines the value to assign to that item. Supply this object to the setValue operation.

```
SetValue value = new SetValue();
value.setDefID("email/default");
value.setContent("jjones@yahoo.com");
SetValueResponse valueResponse = stub.setValue(value);
```

SOAP request example

Client invocation of the setValue 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>
    <setValue xmlns="http://xml.spss.com/pes/userPref/remote">
      <setValue xmlns="http://www.spss.com/pes/userPref"
        xmlns:ns1="http://www.spss.com/pes/userPref" ns1:defID="email/default">
        jjones@yahoo.com
      </setValue>
    </setValue>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a setValue 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>
    <setValueResponse xmlns="http://xml.spss.com/pes/userPref/remote">
      <ns1:setValueResponse xmlns:ns1="http://www.spss.com/pes/userPref">
        <ns1:response>OK</ns1:response>
      </ns1:setValueResponse>
    </setValueResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Manager port type

The Manager port type includes operations used for managing preference items, such as retrieving item definitions and identifying users with values for items.

Operation reference

The getDefinitions operation

Retrieves all preference item definitions available in the system. The operation can also return a specific item based on its definition ID.

Use this operation to get a list of all possible preference items for constructing a data-driven user interface. The item types can be used to determine the optimal interface control to use for editing the value, while the constraints can be used to validate new values. Use the setValue operation to actually update the existing value. See the topic “The setValue operation” on page 14 for more information.

Input fields

The following table lists the input fields for the getDefinitions operation.

Table 11. Fields for definitionRequest.

Field	Type/Valid Values	Description
requestedItemID	string	Identifier for the definition to retrieve. If omitted, all defined preference definitions are returned.

Return information

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

Table 12. Return Value.

Type	Description
preferenceDataItems	Characteristics of the preference item(s) corresponding to the request.

Java example

The getDefinitions operation returns a DefinitionResponse object containing information about the existing preference items. Use the getPreferenceDataItems method to return a PreferenceDataItems

object from which an array of PreferenceData objects containing details about the items can be obtained using the getPreferenceData method. For each entry in the array, accessor methods return specific properties, such as the item name, ID, and type.

```
DefinitionResponse response = stub.getDefinitions();
PreferenceDataItems items = response.getPreferenceDataItems();
PreferenceData[] prefData = items.getPreferenceData();
for (int j = 0; j < prefData.length; j++) {
    System.out.println("Name: " + prefData[j].getItemName());
    System.out.println("ID: " + prefData[j].getDefID());
    System.out.println("Type: " + prefData[j].getItemType());
    System.out.println("Component: " + prefData[j].getComponentName());
    System.out.println("Instance Count: " + prefData[j].getInstanceCount());
}
```

SOAP request example

Client invocation of the getDefinitions 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>
    <definitionRequest xmlns="http://xml.spss.com/pes/userPref/remote">
      <definitionRequest xmlns="http://www.spss.com/pes/userPref"/>
    </definitionRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a getDefinitions 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>
    <definitionRequestResponse xmlns="http://xml.spss.com/pes/userPref/remote">
      <ns1:definitionResponse xmlns:ns1="http://www.spss.com/pes/userPref">
        <ns1:preferenceDataItems>
          <ns1:preferenceData ns1:componentName="security/cfgGroup/"
            ns1:defID="com.spss.security/upDefaultACL" ns1:exposeInServer="false"
            ns1:instanceCount="0" ns1:itemName="security/upnDefaultACL"
            ns1:itemType="accessControlList" ns1:keyed="false">
            <ns1:helpHTML>Default privileges for creating objects.</ns1:helpHTML>
          </ns1:preferenceData>
          <ns1:preferenceData ns1:componentName="User Preference Sample"
            ns1:defID="demo/Bool" ns1:exposeInServer="false" ns1:instanceCount="0"
            ns1:itemName="bool type" ns1:itemType="bool" ns1:keyed="false">
            <ns1:helpHTML>Yes or No</ns1:helpHTML>
          </ns1:preferenceData>
          <ns1:preferenceData ns1:componentName="User Preference Sample"
            ns1:defID="demo/Choice" ns1:exposeInServer="false" ns1:instanceCount="0"
            ns1:itemName="choices type" ns1:itemType="choices" ns1:keyed="false">
            <ns1:constraints>&quot;red|1&quot;;&quot;green|2g&quot;;&quot;blue|3b&quot;</ns1:constraints>
            <ns1:helpHTML>Select a color.</ns1:helpHTML>
          </ns1:preferenceData>
          <ns1:preferenceData ns1:backing="config/demo/FreeForm"
            ns1:componentName="User Preference Sample" ns1:defID="demo/FreeForm"
            ns1:exposeInServer="false" ns1:instanceCount="0" ns1:itemName="freeForm type"
            ns1:itemType="freeForm" ns1:keyed="false">
            <ns1:constraints>0:2147483647</ns1:constraints>
            <ns1:helpHTML>Any string.</ns1:helpHTML>
          </ns1:preferenceData>
          <ns1:preferenceData ns1:componentName="User Preference Sample"
            ns1:defID="demo/Int" ns1:exposeInServer="false" ns1:instanceCount="0">
```

```

        ns1:itemName="int type" ns1:itemType="int" ns1:keyed="false">
        <ns1:constraints>-2147483648:2147483647</ns1:constraints>
        <ns1:helpHTML>A number</ns1:helpHTML>
    </ns1:preferenceData>
    <ns1:preferenceData ns1:componentName="User Preference Sample"
        ns1:defID="demo/Password" ns1:exposeInServer="false" ns1:instanceCount="0"
        ns1:itemName="password type" ns1:itemType="password" ns1:keyed="false">
        <ns1:helpHTML>A password.</ns1:helpHTML>
    </ns1:preferenceData>
    <ns1:preferenceData ns1:backing="userdir:email" ns1:componentName="General"
        ns1:defID="email/default" ns1:exposeInServer="true" ns1:instanceCount="0"
        ns1:itemName="email type" ns1:itemType="email" ns1:keyed="false">
        <ns1:helpHTML>The user's specified default email address.</ns1:helpHTML>
    </ns1:preferenceData>
</ns1:preferenceDataItems>
</ns1:definitionResponse>
</definitionRequestResponse>
</soapenv:Body>
</soapenv:Envelope>

```

The getUsers operation

Returns the set of all users that have user preferences values defined. The principal calling this operation must be assigned to a role with the *userPref/Admin* action.

The list of users returned by the `getUsers` operation can be compared to lists of users for directories in use. Users present in the former list but not in the latter have been removed from the system and their values can be safely deleted. Supply the `deleteValues` operation with value addresses that include the principal IDs for those users to remove their values. See the topic “The `deleteValues` operation” on page 11 for more information.

Input fields

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

Table 13. Fields for *usersRequest*.

Field	Type/Valid Values	Description
usersRequest	usersRequest	

Return information

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

Table 14. Return Value.

Type	Description
userDataItems	Characteristics of users, such as their principal IDs and display names, who have preference values specified.

Java example

The `getUsers` operation returns a `UsersResponse` object containing information about the users. Use the `getUserDataItems` method to return a `UserDataItems` object from which an array of `UserDetails` objects containing details about the returned users can be obtained using the `getUserDetails` method.

For each entry in the array, the `getInstanceCount` method returns the number of preference values for the user. In addition, the `getUser` method returns a `User` object containing the display name and principal ID for the user.

```

UsersResponse users = stub.getUsers();
UserDetails[] details = users.getUserDataItems().getUserDetails();
for (int j = 0; j < details.length; j++) {
    User userProfile = details[j].getUser();
}

```

```

System.out.println("User " + userProfile.getDisplay() +
    " (principal ID = " + userProfile.getPrincipalID() + ") has " +
    details[j].getInstanceCount() + "preference item values defined.");
}

```

SOAP request example

Client invocation of the `getUsers` 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>
    <usersRequest xmlns="http://xml.spss.com/pes/userPref/remote">
      <usersRequest xmlns="http://www.spss.com/pes/userPref"/>
    </usersRequest>
  </soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a `getUsers` 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>
    <usersRequestResponse xmlns="http://xml.spss.com/pes/userPref/remote">
      <ns1:usersResponse xmlns:ns1="http://www.spss.com/pes/userPref">
        <ns1:userDataItems>
          <ns1:userDetails ns1:instanceCount="1">
            <ns1:user ns1:principalID="//uNative//validUser" ns1:display="validUser"/>
          </ns1:userDetails>
          <ns1:userDetails ns1:instanceCount="1">
            <ns1:user ns1:principalID="//uADL/domain/jjones" ns1:display="jjones (domain)"/>
          </ns1:userDetails>
        </ns1:userDataItems>
      </ns1:usersResponse>
    </usersRequestResponse>
  </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 15. Return Value.

Type	Description
string	The version of the web service.

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://xml.spss.com/pes/userPref/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/pes/userPref/remote">
      <version>4.20.000</version>
    </getVersionResponse>
  </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/wsd1/scoring.wsd1"),
        new QName("http://xml.spss.com/scoring/wsd1", "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 26 for more information.
3. Programmatically configure the necessary endpoint behaviors. See the topic “Configuring endpoint behaviors” on page 27 for more information.
4. Exercise the web service. See the topic “Exercising the service” on page 27 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 16. 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/
spssec	http://xml.spss.com/security

Security element

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

Table 17. Attributes of `wsse:Security`

Attribute	Description	Example
<code>soapenv:actor</code>	Targets a given endpoint along the message path. This value is ignored.	http://schemas.xmlsoap.org/soap/actor/next
<code>soapenv:mustUnderstand</code>	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 18. Attributes of `wsse:UsernameToken`

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

Table 19. Child elements of `wsse:UsernameToken`

Attribute	Description	Example
<code>wsse:Username</code>	The xml value represents the identity of the user.	<code>a_user</code>
<code>wsse:Password</code>	The attribute <code>Type</code> specifies the type of password. <code>PasswordText</code> is currently the only supported type. The xml value can handle plain text passwords and encrypted data.	<code>myPassword</code> <code>[{AES}KrY+KLIOYo4O6545tgGsYQ==]</code>
<code>wsse:Nonce</code>	The xml value represents a cryptographically random nonce encoded as base64 data. This is currently ignored.	<code>RUx1ugQo0o3g0Xyl+sUEsA==</code>
<code>wsu:Created</code>	The xml value represents the creation time as a timestamp conforming to <code>wsu:Timestamp</code> . This is currently ignored.	<code>2013-10-08T02:09:20Z</code>

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 20. Attributes of `wsse:BinarySecurityToken`

Attribute	Description	Example
<code>ValueType</code>	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 <code>spssec:BinarySecuritySSOToken</code> .	<code>spssec:BinarySecurityToken</code> <code>spssec:BinarySecuritySSOToken</code>

Table 20. 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 <code>wsse:Base64Binary</code> . IBM SPSS Collaboration and Deployment Services always writes these values when creating its own XML, but this value is currently ignored during processing.	<code>wsse:Base64Binary</code>
<code>wsu:Id</code>	An identifier for the token. This value must be correctly provided. You should always provide <code>spssSSOToken</code> . The only valid case for using <code>spssToken</code> is for internal web service calls, which use an internal token format.	<code>spssToken</code> <code>spssSSOToken</code>
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 21. HTTP headers

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

Appendix. Deprecated features

If you are migrating from an earlier release of IBM SPSS Collaboration and Deployment Services, you should be aware of the various features that have been deprecated since the last version.

If a feature is deprecated, IBM Corp. might remove this capability in a subsequent release of the product. Future investment will be focused on the strategic function listed under the recommended migration action. Typically, a feature is not deprecated unless an equivalent alternative is provided.

No features have been deprecated in this release. For reference purposes, the following table indicates features that were deprecated in recent previous versions of the product. Where possible, the table also indicates the recommended migration action.

Table 22. Features deprecated in previous versions

Deprecation	Recommended migration action
Security Provider: Active Directory with local override, which supports extended groups and allowed users	Use the standard Active Directory security provider with any necessary groups added
IBM SPSS Collaboration and Deployment Services Enterprise View	Use the Analytic Data View feature
IBM SPSS Collaboration and Deployment Services Enterprise View Driver	Use the Analytic Data View feature
Scenario files	Scenario files (.scn) are no longer supported. Enterprise View source nodes cannot be modified in Deployment Manager. Old scenario files can be modified in IBM SPSS Modeler client and resaved as stream files. Also, scoring configurations that used a scenario file must be deleted and recreated based on a stream file.
Web Install for IBM SPSS Deployment Manager	Use the standalone installer
BIRT Report Designer for IBM SPSS	None
BIRT Report Designer for IBM SPSS viewer	None
IBM SPSS Collaboration and Deployment Services Portlet	Use the IBM SPSS Collaboration and Deployment Services Deployment Portal directly, or use the web services APIs
IBM SPSS Collaboration and Deployment Services Web Part	Use the IBM SPSS Collaboration and Deployment Services Deployment Portal directly, or use the web services APIs
Scoring Service V1 API	Scoring Service V2 API
Scheduling Server Service	None
Reporting Service	None
Authentication Service login operation	Authentication Service doLogin operation
Search Service search operation	Search Service search2.5 operation
SPSS AXIS/Castor web services client jar	Use the tools provided with the Java Runtime Environment, Integrated Development Environment, or Eclipse Web Tools Platform (WTP)
clemr1_setLogFile() API function	None

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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, MD-NC119
Armonk, NY 10504-1785
US*

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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide 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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

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.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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.

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 actual people or business enterprises 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.

Privacy policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, See IBM's Privacy Policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details> the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.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

This glossary includes terms and definitions for IBM SPSS Collaboration and Deployment Services.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology (opens in new window).

A

access control list (ACL)

In computer security, a list associated with an object that identifies all the subjects that can access the object and their access rights.

ACL See access control list.

action A permission for an aspect of system functionality. For example, the ability to set up notifications is defined as an action. Actions are grouped and assigned to users through roles. See also role.

Active Directory (AD)

A hierarchical directory service that enables centralized, secure management of an entire network, which is a central component of the Microsoft Windows platform.

AD See Active Directory.

allowed user

A subset of the users defined in a remote directory, such as SiteMinder or Windows Active Directory, that are allowed access to SPSS Predictive Enterprise Services. Allowed users are defined when only a few users in a remote directory need access to the application.

API See application programming interface.

appender

A component that receives logging

requests from a logger and writes log statements to a specified file or console. See also logger.

application programming interface (API)

An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

B

batch file

A file that contains instructions that are processed sequentially, as a unit.

binary large object (BLOB)

A data type whose value is a sequence of bytes that can range in size from 0 bytes to 2 gigabytes less 1 byte. This sequence does not have an associated code page and character set. BLOBs can contain, for example, image, audio, or video data.

BLOB See binary large object.

break group

A set of rows of returned data that are grouped according to a common column value. For example, in a column of states, the rows of data for each state are grouped together.

burst report

A report that generates multiple output files during a single run by using multiple input parameters taken from break groups in the report.

C

cascading permission

A permission of a parent folder in the content repository that has been propagated to its child objects.

character large object (CLOB)

A data type whose value is a sequence of characters (single byte, multibyte, or both) that can range in size from 0 bytes to 2 gigabytes less 1 byte. In general, the CLOB data type is used whenever a

character string might exceed the limits of the VARCHAR data type.

CLOB See character large object.

common warehouse metamodel (CWM)

A metamodel written to be a common standard by the Object Management Group (OMG).

content repository

A centralized location for storing analytical assets, such as models and data. Content repository includes facilities for security and access control, content management, and process automation.

context data

Input data that is passed with a scoring request in real time. For example, when a score is requested for a customer based on credit rating and geocode, the credit score and geocode will be the context data for the request.

credential

Information acquired during authentication that describes a user, group associations, or other security-related identity attributes, and that is used to perform services such as authorization, auditing, or delegation. For example, a user ID and password are credentials that allow access to network and system resources.

CWM See common warehouse metamodel.

D

data warehouse

A subject-oriented collection of data that is used to support strategic decision making. The warehouse is the central point of data integration for business intelligence. It is the source of data for data marts within an enterprise and delivers a common view of enterprise data.

distinguished name (DN)

The name that uniquely identifies an entry in a directory. A distinguished name is made up of attribute:value pairs, separated by commas. For example, CN=person name and C=country or region.

DN See distinguished name.

Document Object Model (DOM)

A system in which a structured document, for example an XML file, is viewed as a tree of objects that can be programmatically accessed and updated. See also Simple API for XML.

document type definition (DTD)

The rules that specify the structure for a particular class of SGML or XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each element, attribute, and notation can be used within the particular class of documents.

DOM See Document Object Model.

dormant schedule

A schedule associated with a deleted or unlabeled version of a job. A dormant schedule cannot be used until it is associated with a valid labeled job version.

DTD See document type definition.

E

EAR See enterprise archive.

enterprise archive (EAR)

A specialized type of JAR file, defined by the Java EE standard, used to deploy Java EE applications to Java EE application servers. An EAR file contains EJB components, a deployment descriptor, and web archive (WAR) files for individual web applications. See also Java archive, web archive.

execution server

A server that enables analytical processing of resources stored in the repository. For example, to execute an IBM SPSS Statistics syntax in an IBM SPSS Collaboration and Deployment Services job, an IBM SPSS Statistics execution server must be designated.

export The process of storing objects and metadata from the content repository to an external file.

extended group

A locally-defined group of remote users. Extended groups are defined when groups in the remote directory are not fine-grained enough.

Extensible Markup Language (XML)

A standard metalanguage for defining markup languages that is based on Standard Generalized Markup Language (SGML).

Extensible Stylesheet Language (XSL)

A language for specifying style sheets for XML documents. Extensible Stylesheet Language Transformation (XSLT) is used with XSL to describe how an XML document is transformed into another document.

F**field content assist**

A feature that provides predefined system and variable values for entry fields.

G**general job step**

A method for running native operating system commands and executable programs on a host or a remote process server. General jobs have access to files stored within the repository and on the file system and can be used to control the input/output of analytical processing.

I**import**

The process of adding objects and metadata defined in an external file generated by export, to the content repository.

iterative consumer reporting job step

A job step that is passed a set of input values generated by a preceding iterative producer reporting job step. The report in iterative consumer job step is executed for each tuple in the received data set.

iterative producer reporting job step

A job step that generates a set of values passed as input parameters to a following iterative consumer job step.

J

JAAS See Java Authentication and Authorization Service.

JAR See Java archive.

Java archive (JAR)

A compressed file format for storing all of the resources that are required to install and run a Java program in a single file. See also enterprise archive, web archive.

Java Authentication and Authorization Service (JAAS)

In Java EE technology, a standard API for performing security-based operations. Through JAAS, services can authenticate and authorize users while enabling the applications to remain independent from underlying technologies.

Java Generic Security Services (JGSS)

A specification that provides Java programs access to the services that include the signing and sealing of messages and a generic authentication mechanism.

Java Naming and Directory Interface (JNDI)

An extension to the Java platform that provides a standard interface for heterogeneous naming and directory services.

JGSS See Java Generic Security Services.

JNDI See Java Naming and Directory Interface.

job A mechanism for automating analytical processing. A job consists of job steps, executed sequentially or conditionally. Input parameters can be defined for a job. A job can be run on demand or triggered by time-based or message-based schedules, with records of job execution stored as job history.

job step

A discrete unit of processing in a job. Depending on the type, job steps can be run on the content repository host or specially defined execution or remote process servers. Objects stored in the repository or the file system can provide input for a job step, and job step output can be stored in the repository or written to the file system.

K

KDC See key distribution center.

Kerberos

A network authentication protocol that is based on symmetric key cryptography. Kerberos assigns a unique key, called a

ticket, to each user who logs on to the network. The ticket is embedded in messages that are sent over the network. The receiver of a message uses the ticket to authenticate the sender.

key distribution center (KDC)

A network service that provides tickets and temporary session keys. The KDC maintains a database of principals (users and services) and their associated secret keys. It is composed of the authentication server and the ticket granting ticket server.

keystore

In security, a file or a hardware cryptographic card where identities and private keys are stored, for authentication and encryption purposes. Some keystores also contain trusted or public keys.

L

LDAP See Lightweight Directory Access Protocol.

Lightweight Directory Access Protocol (LDAP)

An open protocol that uses TCP/IP to provide access to directories that support an X.500 model and that does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). For example, LDAP can be used to locate people, organizations, and other resources in an Internet or intranet directory.

lock The process by which integrity of data is ensured by preventing more than one user from accessing or changing the same data or object at the same time.

logger A component that prepares log statements to be written to console or log file. See also appender.

M

message-based schedule

A schedule that is used to trigger job execution by an event signalled by a Java Messaging Service (JMS) message. For example, when a job relies on the input from a third-party application, the application must send a JMS message when the input file is ready for processing.

metamodel

A model that defines the language for expressing a model.

meta-object

An instance of an XMI class as defined in the metamodel.

meta-object facility (MOF)

A generalized facility and repository for storing abstract information about concrete object systems; dealing mostly with construction, standardized by the Object Management Group (OMG).

MIME See Multipurpose Internet Mail Extensions.

MOF See meta-object facility.

Multipurpose Internet Mail Extensions (MIME)

An Internet standard that allows different forms of data, including video, audio, or binary data, to be attached to email without requiring translation into ASCII text.

N

notification

A mechanism that is used to generate email messages informing users of specific types of system events, such as changes to content repository objects and processing success and failure. Unlike subscriptions, notifications can be set up to send email to multiple users.

O

Object Management Group (OMG)

A non-profit consortium whose purpose is to promote object-oriented technology and the standardization of that technology. The Object Management Group was formed to help reduce the complexity, lower the costs, and hasten the introduction of new software applications.

ODS See Output Delivery System.

OMG See Object Management Group.

Output Delivery System (ODS)

A method of controlling the destination for output within SAS. ODS can route SAS output to a SAS data file, a text listing file, HTML files, and files optimized for high-resolution printing.

P

package

An installable unit of a software product. Software product packages are separately installable units that can operate independently from other packages of that software product.

principal

An entity that can communicate securely with another entity. A principal is identified by its associated security context, which defines its access rights.

R

remote process server

A remote system that is designated for running native operating system commands and executable programs.

repository content adapter

An optional software package that enables storing and processing content from other IBM SPSS applications, such as Statistics, Modeler, and Data Collection, as well as third parties.

repository database

A relational database that is used for storing content repository objects and metadata.

resource

A content repository object.

resource definition

A subset of content repository resources used to enable analytical processing, such as definitions of data sources, credentials, execution servers, and JMS message domains.

role

A set of permissions or access rights. See also action.

S

SAX See Simple API for XML.

schedule

A content repository object that triggers job execution.

scoring configuration

A configuration that defines model-specific settings for generating

real-time scores, such as input data, processing rules, outputs, logging, etc.

security provider

A system that performs user authentication. Users and groups can be defined locally (in which case, IBM SPSS Collaboration and Deployment Services itself is the security provider) or derived from a remote directory, such as Windows Active Directory or OpenLDAP.

service provider interface (SPI)

An API that supports replaceable components and can be implemented or extended by a third party.

SGML

See Standard Generalized Markup Language.

shell script

A program, or script, that is interpreted by the shell of an operating system.

Simple API for XML (SAX)

An event-driven, serial-access protocol for accessing XML documents, used. A Java-only API, SAX is used by most servlets and network programs to transmit and receive XML documents. See also Document Object Model.

single sign-on (SSO)

An authentication process in which a user can access more than one system or application by entering a single user ID and password.

SOAP A lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP can be used to query and return information and invoke services across the Internet.

SPI See service provider interface.

SSO See single sign-on.

Standard Generalized Markup Language (SGML)

A standard metalanguage for defining markup languages that is based on the ISO 8879 standard. SGML focuses on structuring information rather than presenting information; it separates the structure and content from the presentation. It also facilitates the interchange of documents across an electronic medium.

stop word

A word that is commonly used, such as "the," "an," or "and," that is ignored by a search application.

subscription

Email notices and Really Simple Syndication (RSS) feeds that repository users create to receive when the state of an asset changes.

T

TGT See ticket-granting ticket.

ticket-granting ticket (TGT)

A ticket that allows access to the ticket granting service on the key distribution center (KDC). Ticket granting tickets are passed to the principal by the KDC after the principal has completed a successful request. In a Windows 2000 environment, a user logs on to the network and the KDC will verify the principal's name and encrypted password and then send a ticket granting ticket to the user.

time-based schedule

A schedule that triggers job execution at a specified time or date. For example, a time-based schedule may run a job at 5:00 pm every Thursday.

U**Universally Unique Identifier (UUID)**

The 128-bit numeric identifier that is used to ensure that two components do not have the same identifier.

UUID See Universally Unique Identifier.

V**Velocity**

A Java-based template engine that provides a simple and powerful template language to reference objects defined in Java code. Velocity is an open source package directed by the Apache Project.

W

W3C See World Wide Web Consortium.

WAR See web archive.

web archive (WAR)

A compressed file format, defined by the Java EE standard, for storing all the resources required to install and run a web application in a single file. See also enterprise archive, Java archive.

Web Services Description Language (WSDL)

An XML-based specification for describing networked services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

World Wide Web Consortium (W3C)

An international industry consortium set up to develop common protocols to promote evolution and interoperability of the World Wide Web.

WSDL

See Web Services Description Language.

X

XMI See XML Metadata Interchange.

XML See Extensible Markup Language.

XML Metadata Interchange (XMI)

A model-driven XML integration framework for defining, interchanging, manipulating, and integrating XML data and objects. XMI-based standards are in use for integrating tools, repositories, applications, and data warehouses.

XSL See Extensible Stylesheet Language.

Index

Special characters

.NET framework 25
.NET proxies 5

A

accessControlList type
 for preference items 9
addresses
 for item values 9
app.config files
 WCF clients 26
assigning
 values 14

B

backing
 preference items 9
BinarySecuritySSOToken element
 in SOAP headers 30
BinarySecurityToken element
 in SOAP headers 30
bindings
 in WSDL files 4
body elements
 in SOAP messages 2
bool type
 for preference items 9

C

choices type
 for preference items 9
Client port type 11
client-accept-language element
 in SOAP headers 31
components
 for preference items 9
Content Repository service
 WCF clients 25
Content Repository URI service
 WCF clients 25
Created element
 in SOAP headers 30

D

definition IDs
 for preference items 9
deleteValues operation 11
deleting
 values 11

E

email type
 for preference items 9

F

freeForm type
 for preference items 9

G

getDefinitions operation 16
getUsers operation 18
getValueByDefId operation 12
getValues operation 12
getVersion operation 14, 19
glossary 39

H

header elements
 in SOAP messages 2, 29
 SOAP security elements 29
Holder classes
 in JAX-WS 5
HTTP 2
HTTP headers
 for SOAP messages 31
HTTPS 2

I

integer type
 for preference items 9
items
 retrieving 16

J

Java clients 21, 22, 24
Java proxies 5
JAX-WS 5, 21, 22, 24

L

List collections
 in JAX-WS 5

M

Manager port type 16
MessageBodyMemberAttribute
 for WCF clients 27
messages
 in WSDL files 4

N

names
 for preference items 9
namespaces
 for SOAP security elements 29

Nonce element
 in SOAP headers 30

O

operation reference 11

P

Password element
 in SOAP headers 30
password type
 for preference items 9
PevServices service
 WCF clients 25
port types
 in WSDL files 4
preference items 9
 backing 9
 values 9
Process Management service
 WCF clients 25
protocols
 in web services 2
proxies 5
 .NET 5
 Java 5

R

retrieving
 items 16
 users 18
 values 12

S

Scoring service
 WCF clients 25
Security element
 in SOAP headers 29
services
 in WSDL files 5
setValue operation 14
single sign-on
 for WCF clients 28
 WCF clients 25
SOAP 2
SOAPHandler 22
SSO
 See single sign-on
stubs
 User Preferences Service service 7

T

types
 for preference items 9
 in WSDL files 3

U

- User Preferences Service service
 - stubs 7
- Username element
 - in SOAP headers 30
- UsernameToken element
 - in SOAP headers 30
- users
 - retrieving 18

V

- values
 - addresses 9
 - assigning 14
 - deleting 11
 - retrieving 12
- Visual Studio 25

W

- WCF clients 25, 27, 28
 - endpoint behaviors 27
 - endpoint configuration 26
 - limitations 25
 - service reference 25
 - single sign-on 25
- web services
 - introduction to web services 1
 - protocol stack 2
 - system architecture 1
 - what are web services? 1
- web.config files
 - WCF clients 26
- Windows Communication Foundation 25
- WSDL files 2, 3
 - bindings 4
 - messages 4
 - port types 4
 - services 5
 - types 3
- wsdl.exe 5
- wsdl2java 5
- wsimport 5, 21

X

- XmlElementAttribute
 - for WCF clients 27



Printed in USA