

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



Note: Before using this information and the product it supports, read the general information under Notices on p. 29.

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

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

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

Licensed Materials - Property of IBM

© Copyright IBM Corporation 2000, 2012.

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

Preface

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

About IBM Business Analytics

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

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

Technical support

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

Contents

1	<i>Introduction to web services</i>	1
	What are web services?	1
	Web service system architecture	1
	Web service protocol stack	2
	Simple Object Access Protocol	3
	Web Service Description Language	3
	Proxies	6
2	<i>User Preferences Service overview</i>	8
	Accessing the User Preferences Service	8
	Calling User Preferences Service operations	8
3	<i>User preferences concepts</i>	9
	Preference items	9
	Preference item values	10
4	<i>Operation reference</i>	11
	Client port type	11
	The deleteValues operation	11
	The getValueByDefId operation	12
	The getValues operation	13
	The getVersion operation	15
	The setValue operation	16
	Manager port type	18
	The getDefinitions operation	18
	The getUsers operation	20
	The getVersion operation	22

Appendices

<i>A</i>	<i>Microsoft® .NET Framework-based clients</i>	<i>24</i>
	Adding a service reference.	24
	Service reference modifications	25
	Configuring the web service endpoint.	26
	Configuring endpoint behaviors	27
	Exercising the service.	27
<i>B</i>	<i>Notices</i>	<i>29</i>
	<i>Index</i>	<i>32</i>

Introduction to web services

What are web services?

At a high level, a **web service** is a set of functionality distributed across a network (LAN or the Internet) using a common communication protocol. The web service serves as an intermediary between an application and its clients, providing both a standardized information structure and a standardized communication protocol for interaction between the two. Where other methods of distributed application architecture rely on a single programming language being used on both the application and its clients, a web service allows the use of loosely coupled services between non-homogenous platforms and languages. This provides a non-architecture-specific approach allowing, for example, Java services to communicate with C# clients, or vice-versa.

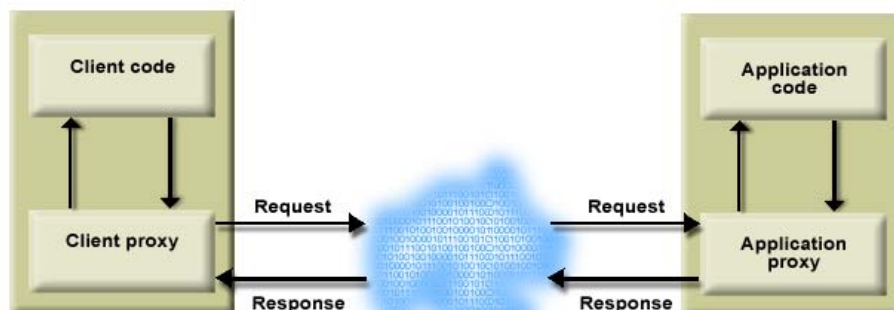
Advantages to implementing application functionality as web services include the following:

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

Web service system architecture

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

Figure 1-1
Web service architecture



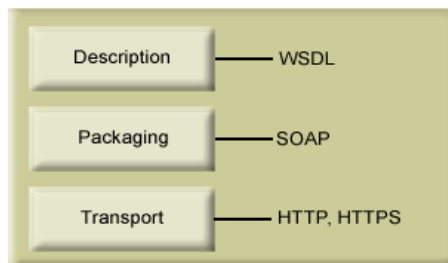
The client code supplies input to an operation offered by a proxy class. The proxy class generates a request containing a standardized representation of the input and sends it across the network to the application. A proxy class on the server receives the request and unmarshals the contents into objects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

Standardizing the format of the information passing between the client and the application allows a client written in one programming language to communicate with an application written in another. The proxy classes, which are automatically generated from a web service description by a variety of toolkits, handle the translation between native programming objects and the standardized representation. For more information, see the topic [Proxies](#) on p. 6.

Web service protocol stack

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

Figure 1-2
Web service protocol stack



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

The Packaging layer rests on top of Transport and defines the standard for structuring information for transport across the network. The SOAP format is commonly used, which offers an XML structure for packaging the data. For more information, see the topic [Simple Object Access Protocol](#) on p. 3.

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

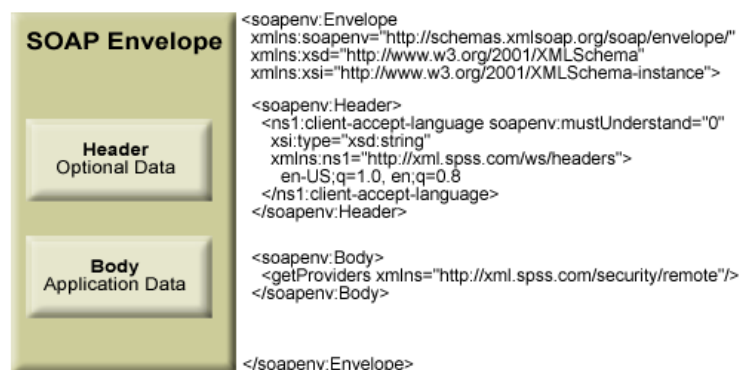
Simple Object Access Protocol

The Simple Object Access Protocol (SOAP) is a way to pass information between applications in an XML format. SOAP messages are transmitted from the sending application to the receiving application, typically over an HTTP session. The actual SOAP message is made up of the Envelope element, which contains a Body element and an optional Header element.

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

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

Figure 1-3
An example SOAP packet



Web Service Description Language

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

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

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

In accordance with the [WSDL specification](http://www.w3.org/TR/wsdl) (<http://www.w3.org/TR/wsdl>) adopted by the World Wide Web Consortium, information in the WSDL is organized into the following sections:

- **Types.** Content definitions for web service operation input and output. For more information, see the topic [Types](#) on p. 4.
- **Messages.** Input and output definitions for the web service operations. For more information, see the topic [Messages](#) on p. 5.
- **PortTypes.** Groups of operations offered by the web service. For more information, see the topic [Port types](#) on p. 5.
- **Bindings.** Protocols and formats for the web service operations. For more information, see the topic [Bindings](#) on p. 5.
- **Services.** Endpoints at which the web service functionality can be accessed. For more information, see the topic [Services](#) on p. 6.

Types

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

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

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

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

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

Messages

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

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

Port types

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

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

The *ProviderInformation* port type consists of a single operation, *getProviders*. Input to this operation corresponds to the *getProvidersRequest* message. The operation returns information in the structure defined by the *getProvidersResponse* message. For more information, see the topic [Messages](#) on p. 5.

Bindings

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

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

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

In this case, the transport attribute of the `wsdlsoap:binding` element defines HTTP as the transport protocol. Both the `getProviders` and `getProvidersResponse` operations in the interface are bound to the SOAP messaging protocol.

Services

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

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

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

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

Proxies

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

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

The proxy classes generated from a WSDL file depend on the tool used. For Java, the `wSDL2java` tool, which is part of the Apache Axis project, can be used. This tool produces a Java class for each type in the WSDL. Each port type results in a Java interface. A binding creates a stub class, and a WSDL service yields a service interface with a locator implementation. These generated classes and interfaces can be called directly from a client application written in Java to access the web service functionality.

An alternative Java proxy tool is `wsimport`, which is part of JAX-WS. The general structure of the generated classes is similar to that created by the Axis tool, but there are some differences. For example, instead of using arrays for input fields and returned items, the code generated from

the *wsimport* tool uses List collections. In addition, if an input type matches an output type for a method, the *wsimport* tool uses a **Holder** class for the parameter.

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

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

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

The value of *<host-name>* corresponds to the machine on which IBM® SPSS® Collaboration and Deployment Services Repository is installed, with *<port-number>* indicating the port number on which it is running. To access the WSDL file for the service, append *?wsdl* to the service endpoint.

For example, if IBM SPSS Collaboration and Deployment Services Repository is running on port 80 of the machine *cads_server*, the WSDL file can be accessed using the path:

```
http://cads_server:80/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);
```

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 e-mail 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 3-1
Item Types

Type	Description	Constraints
<i>freeForm</i>	A string.	Minimum and maximum number of characters in the string.
<i>email</i>	An e-mail address compliant with RFC-822 (http://www.ietf.org/rfc/rfc0822.txt).	
<i>password</i>	A security-sensitive string. Values for items of this type are often masked in user interfaces.	
<i>choices</i>	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>).
<i>bool</i>	A binary choice, such as Yes/No or True/False.	
<i>int</i>	An integer.	The minimum and maximum values for the integer.
<i>accessControlList</i>	A set of principalID/permissionID 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 e-mail 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 comprised 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.

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.

The deleteValues operation

Removes one or more values for user preference items. This would typically be used to cleanup 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 4-1
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/or a values key.

Return information

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

Table 4-2
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>
    <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 4-3
Fields for `getValueByDefId`

Field	Type/Valid Values	Description
<code>defIdRequest</code>	<code>defIdRequest</code>	

Return information

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

Table 4-4
Return Value

Type	Description
<code>defIdResponse</code>	

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 4-5
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/or a values key.

Return information

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

Table 4-6
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 4-7
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 4-8
Fields for `setValue`

Field	Type/Valid Values	Description
setValue	setValue	The address of the value to set, and the actual content/data to assign to the address.

Return information

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

Table 4-9
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.

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. For more information, see the topic [The `setValue` operation](#) on p. 16.

Input fields

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

Table 4-10
Fields for `definitionRequest`

Field	Type/Valid Values	Description
<code>definitionRequest</code>	<code>definitionRequest</code>	

Return information

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

Table 4-11
Return Value

Type	Description
<code>definitionResponse</code>	

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|1r&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. For more information, see the topic [The *deleteValues* operation](#) on p. 11.

Input fields

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

Table 4-12
Fields for `usersRequest`

Field	Type/Valid Values	Description
<code>usersRequest</code>	<code>usersRequest</code>	

Return information

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

Table 4-13
Return Value

Type	Description
<code>usersResponse</code>	

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:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <body>
    <getUsersRequest/>
  </body>
</soapenv:Envelope>
```

```

    <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 4-14
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>
```

Microsoft® .NET Framework-based clients

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

1. Add a Service Reference. For more information, see the topic [Adding a service reference](#) on p. 24.
2. Configure the web service endpoint. For more information, see the topic [Configuring the web service endpoint](#) on p. 26.
3. Programmatically configure the necessary endpoint behaviors. For more information, see the topic [Configuring endpoint behaviors](#) on p. 27.
4. Exercise the web service. For more information, see the topic [Exercising the service](#) on p. 27.

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

Adding a service reference

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

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

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

Service reference modifications

Due to known compatibility issues between Microsoft tooling and some WSDL files, you need to manually modify some service references before they can be used successfully. For information about the specific issues, see articles [891386](http://support.microsoft.com/kb/891386) (<http://support.microsoft.com/kb/891386>) and [326790](http://support.microsoft.com/kb/326790) (<http://support.microsoft.com/kb/326790>) on the Microsoft Support site.

To modify a service reference:

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

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

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

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

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

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

- in the *avTableConflict* class, `private avColumnMeta[][] avColumnConflictField` should be changed to `private avColumnMeta[] avColumnConflictField`

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

Configuring the web service endpoint

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

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

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

```
<system.serviceModel>
  <client>
    <endpoint
      address="http://cads_server:8080/cr-ws/services/ContentRepository"
      binding="basicHttpBinding"
      bindingConfiguration=""
      contract="IBM.SPSS.ContentRepository"
      name=""/>
  </client>
</system.serviceModel>
```


Configuring endpoint behaviors

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

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

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

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

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

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

Exercising the service

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

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

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

```
public IBM.SPSS.ManagerUserPref.usersResponse usersResponse;
```

The corrected code is as follows:

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

Notices

This information was developed for products and services offered worldwide.

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

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

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

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

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

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

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

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

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

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

IBM Software Group, Attention: Licensing, 233 S. Wacker Dr., Chicago, IL 60606, USA.

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

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

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

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

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

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

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

Trademarks

IBM, the IBM logo, [ibm.com](http://www.ibm.com), and SPSS are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>.

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

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

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

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

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

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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



Index

- accessControlList type
 - for preference items, 9
- addresses
 - for item values, 10
- app.config files
 - WCF clients, 26
- assigning
 - values, 16

- backing
 - preference items, 9
- bindings
 - in WSDL files, 5
- body elements
 - in SOAP messages, 3
- bool type
 - for preference items, 9

- choices type
 - for preference items, 9
- Client port type, 11
- components
 - for preference items, 9
- Content Repository service
 - WCF clients, 25
- Content Repository URI service
 - WCF clients, 25

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

- email type
 - for preference items, 9

- freeForm type
 - for preference items, 9

- getDefinitions operation, 18
- getUsers operation, 20
- getValueByDefId operation, 12
- getValues operation, 13
- getVersion operation, 15, 22

- header elements
 - in SOAP messages, 3
- Holder classes
 - in JAX-WS, 6
- HTTP, 2
- HTTPS, 2

- integer type
 - for preference items, 9
- items
 - retrieving, 18

- Java proxies, 6
- JAX-WS, 6

- legal notices, 29
- List collections
 - in JAX-WS, 6

- Manager port type, 18
- MessageBodyMemberAttribute
 - for WCF clients, 27
- messages
 - in WSDL files, 5

- names
 - for preference items, 9
 - .NET framework, 24
 - .NET proxies, 7

- operation reference, 11

- password type
 - for preference items, 9
- PevServices service
 - WCF clients, 25
- port types
 - in WSDL files, 5
- preference items, 9
 - backing, 9
 - values, 10
- Process Management service
 - WCF clients, 25
- protocols
 - in web services, 2
- proxies, 6
 - Java, 6
 - .NET, 7

- retrieving
 - items, 18
 - users, 20
 - values, 12–13

- Scoring service
 - WCF clients, 25
- service endpoints
 - User Preferences Service service, 8

-
- services
 - in WSDL files, 6
 - setValue operation, 16
 - single sign-on
 - WCF clients, 24
 - SOAP, 2–3
 - stubs
 - User Preferences Service service, 8

 - trademarks, 30
 - types
 - for preference items, 9
 - in WSDL files, 4

 - User Preferences Service service
 - service endpoint, 8
 - stubs, 8
 - users
 - retrieving, 20

 - values
 - addresses, 10
 - assigning, 16
 - deleting, 11
 - retrieving, 12–13
 - Visual Studio, 24

 - WCF clients, 24, 27
 - endpoint behaviors, 27
 - endpoint configuration, 26
 - limitations, 24
 - service reference, 24–25
 - single sign-on, 24
 - 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, 24
 - WSDL files, 2–3
 - accessing, 8
 - bindings, 5
 - messages, 5
 - port types, 5
 - services, 6
 - types, 4
 - wsdl.exe, 7
 - wsdl2java, 6
 - wsimport, 6

 - XmlElementAttribute
 - for WCF clients, 27