IBM SPSS Collaboration and Deployment Services 5 Directory Information Service Developer's Guide

IBM®

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

Appendices

Introduction to web services

What are web services?

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

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

■ Software written in different languages (Java or C#) running on different platforms (UNIX or Windows) can exchange services and data

■ Application functionality can be accessed by a variety of clients. For example, both a thin-client interface and a rich-client interface can take advantage of the web service operations.

■ Updates to the service are immediately available to all service clients

Web service system architecture

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

Figure 1-1
Web service architecture
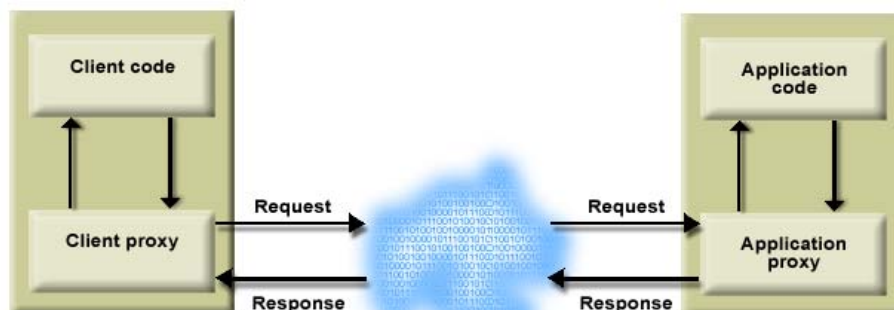
The client code supplies input to an operation offered by a proxy class. The proxy class generates a request containing a standardized representation of the input and sends it across the network to the application. A proxy class on the server receives the request and unmarshals the contents into objects 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
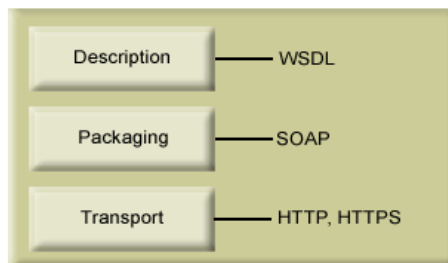
Beginning at the bottom of the stack, the Transport layer defines the technology standards for communication, allowing information to move across the network. HTTP or HTTPS are often used as the standard for the transport layer.

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
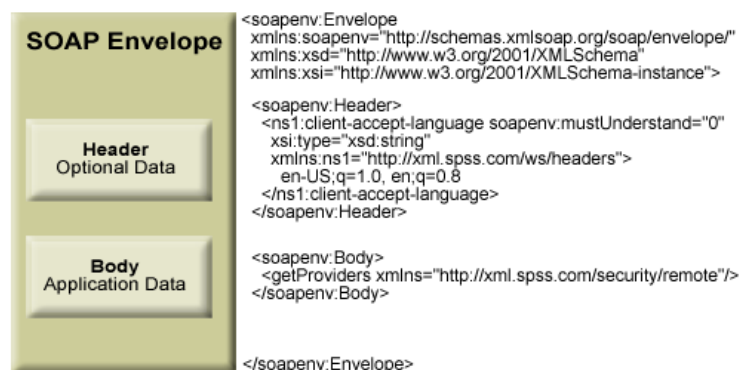
Web Service Description Language

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*) 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 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. 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 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. 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 example binding definition:

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

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

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

Services

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

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

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

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

Proxies

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

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

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

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

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

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

Directory Information Service overview

The Directory Information Service returns information about principals available to IBM® SPSS® Collaboration and Deployment Services.

Accessing the Directory Information Service

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

```
http://<host-name>:<port-number>/security-ws/services/DirectoryInformation
```

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/security-ws/services/DirectoryInformation?wsdl
```

Calling Directory Information 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 = "/security-ws/services/DirectoryInformation";
URL url = new URL("http", "cads_server", 80, context);
DirectoryInformationService service = new DirectoryInformationServiceLocator();
stub = service.getDirectoryInformation(url);
```

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

```
stub.getPrincipals(directoryCriterion);
```

Directory Information Service concepts

Principals

Principals fall into one of three categories, as follows:

- A **user** is an individual who needs access to the system
- A **group** is a set of users who need access to the system
- A **role** is a set of one or more privileges, or actions. Roles are assigned to users or groups to manage system security.

Each principal in the system is characterized by the following attributes:

- **ID.** A unique identifier of the principal. The ID may be useful for debugging purposes, but should generally not be shown to users.
- **Type.** An indicator of whether the principal is a user, group, or role.
- **Display name.** A name for the principal suitable for display by a client application. This name may include the provider name and domain for some system configurations.
- **Type name.** A reference to a localized version of the type indicator.

Principal filters

In the absence of any other information, the Directory Information Service returns all available principals. However, the returned data can be restricted to smaller subsets by specifying filtering (or directory) criteria, such as the following:

- A key identifying the security provider to be searched
- A principal type to restrict the information returned to either users, groups, or roles
- A partial name consisting of the initial letters of the names of principals to be returned. The service ignores the case of the name. Omitting this criterion returns all principals of the specified type.

For example, a request for principals can be limited to all users whose names begin with *j*.

Operation reference

The getPrincipals operation

Returns information about the principals available in IBM® SPSS® Collaboration and Deployment Services. This information can be useful in populating user, group, or role lists.

Input fields

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

Table 4-1
Fields for getPrincipals

Field	Type/Valid Values	Description
directoryCriterion	directoryCriterion	Criterion used to search a user directory.

Return information

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

Table 4-2
Return Value

Type	Description
principalInfo[]	An identity assigned to an authenticated entity, such as an individual or corporation. Examples of Principals include a person's name, a login id, or a corporation name. A principal is generally tied to an authentication mechanism. For example, you might have a Principal related to a certificate, or a Principal related to a Kerberos ticket. This is analogous to the real world where a person carries a name, driver's license, social security card, and passport, each of which uniquely identify the same person. A Principal may either represent a single entity, a group, or even a machine. For example, a person may carry a "frequent diner" card which identifies them as part of the "frequent diners" group.

Java example

The following function returns an array of `PrincipalInfo` objects for a specified security provider. The function accepts a partial name designation, *pattern*, to limit the returned array to principals beginning with a supplied string. In addition, the array can be restricted to users or groups using the boolean arguments *includeUsers* and *includeGroups*.

```
public PrincipalInfo[] getUsersAndGroups(
        ProviderInfo provider,
```

```
        String pattern,
        boolean includeUsers,
        boolean includeGroups)
    throws RemoteException, IOException, ServiceException {
    String providerKey = provider.getProviderKey();
    DirectoryCriterion directoryCriterion = new DirectoryCriterion();
    directoryCriterion.setPartialName(pattern);
    directoryCriterion.setProviderKey(providerKey);
    PrincipalInfo[] principals = getDirectoryInformation().getPrincipals(directoryCriterion);
    if (!includeUsers || !includeGroups) {
      ArrayList filtered = new ArrayList();
      for (int j = 0; j < principals.length; j++) {
        PrincipalInfo principal = principals[j];
        boolean isUser = principal.getIsUser();
        boolean isGroup = principal.getIsGroup();
        if ((includeUsers && isUser) || (includeGroups && isGroup)) {
          filtered.add(principal);
        }
      }
      principals = (PrincipalInfo[]) filtered.toArray(new PrincipalInfo[filtered.size()]);
    }
    return principals;
}
```

The function begins by retrieving the key for the specified provider. This key and the supplied partial name are assigned to a DirectoryCriterion object. The getPrincipals operation uses this object to filter the list of principals returned to the PrincipalInfo array.

If either boolean *includeUsers* or *includeGroups* is false, the principal list needs to be filtered further. In this case, the getIsUser and getIsGroup methods return booleans indicating whether each principal is a user or a group. If users are to be included in the results, the function adds each principal that is a user to an ArrayList. Similarly, if groups are to be included, each principal that is a group is added. Finally, the function converts the ArrayList to an Array, which is returned to the function caller.

SOAP request example

Client invocation of the getPrincipals 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>
    <getPrincipals xmlns="http://xml.spss.com/security/remote">
     <directoryCriterion xmlns="http://xml.spss.com/security">
       <providerKey>p0</providerKey>
       <partialName>%</partialName>
     </directoryCriterion>
    </getPrincipals>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a getPrincipals 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>
  <getPrincipalsResponse xmlns="http://xml.spss.com/security/remote">
   <principalInfo ID="//gNative//$$security/everyoneGroup" isUser="false"
     isGroup="true" isRole="false" principalType="group"
     displayName="–everyone –" typeName="security/principalTypeGroup"
     xmlns="http://xml.spss.com/security"/>
   <principalInfo ID="//uNative//admin" isUser="true"
     isGroup="false" isRole="false" principalType="user"
     displayName="admin" typeName="security/principalTypeUser"
     xmlns="http://xml.spss.com/security"/>
   <principalInfo ID="//rNative//$$security/roleAdministrators"
     isUser="false" isGroup="false" isRole="true"
     principalType="role" displayName="administrators"
     typeName="security/principalTypeRole"
     xmlns="http://xml.spss.com/security"/>
   <principalInfo ID="//uNative//jhealie" isUser="true"
     isGroup="false" isRole="false" principalType="user"
     displayName="jhealie" typeName="security/principalTypeUser"
     xmlns="http://xml.spss.com/security"/>
   <principalInfo ID="//uNative//kkrueter" isUser="true"
     isGroup="false" isRole="false" principalType="user"
     displayName="kkrueter" typeName="security/principalTypeUser"
     xmlns="http://xml.spss.com/security"/>
   <principalInfo ID="//uNative//mnance" isUser="true"
     isGroup="false" isRole="false" principalType="user"
     displayName="mcnance" typeName="security/principalTypeUser"
     xmlns="http://xml.spss.com/security"/>
   <principalInfo ID="//uNative//rcoen" isUser="true"
```

```
      isGroup="false" isRole="false" principalType="user"
      displayName="rcoen" typeName="security/principalTypeUser"
      xmlns="http://xml.spss.com/security"/>
   </getPrincipalsResponse>
 </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 4-3
Return Value

Type	Description
string	The version of the web service.

Java example

The following code uses the WSConnections class to return stubs for the services. The getVersion operation returns the version number of each returned service to the standard output.

```
String host = "localhost";
    int port = 80;
    boolean useSSL = false;
    String username = "admin";
    String password = "spss";
    String acceptLanguage = "en_us";

    // create an instance of the WebServiceConnections, passing in all the
    // relevant connection information.
    WebServiceConnections wsConnections = new WebServiceConnections(host,
        port, useSSL, username, password, acceptLanguage);

CapabilityInformation capabilityInformation = wsConnections.getCapabilityInformation();
    System.out.println("CapabilityInformation version = " + capabilityInformation.getVersion());

DirectoryManagement directoryManagement = wsConnections.getDirectoryManagement();
    System.out.println("Directory Management version = " + directoryManagement.getVersion());

ProviderInformation providerInformation = wsConnections.getProviderInformation();
    System.out.println("ProviderInformation version = " + providerInformation.getVersion());

DirectoryInformation directoryInformation = wsConnections.getDirectoryInformation();
    System.out.println("DirectoryInformation version = " + directoryInformation.getVersion());
```

```
Authentication authentication = wsConnections.getAuthentication();
      System.out.println("Authentication version = " + authentication.getVersion());

SSODirectoryManagement ssoDirectoryManagement = wsConnections.getSSODirectoryManagement();
      System.out.println("SSODirectoryManagement version = " + ssoDirectoryManagement.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/security/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/security/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

2. Configure the web service endpoint. For more information, see the topic

3. Programmatically configure the necessary endpoint behaviors. For more information, see the topic

4. Exercise the web service. For more information, see the topic

   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*) and 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 Servicesweb 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, 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

23

Index