

IBM SPSS Collaboration and Deployment Services
Version 7 Release 0

*Directory Management Service
Developer's Guide*

IBM

Note

Before using this information and the product it supports, read the information in "Notices" on page 53.

Product Information

This edition applies to version 7, release 0, modification 0 of IBM SPSS Collaboration and Deployment Services and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 2000, 2015.

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

Contents

Chapter 1. Introduction to web services	1
What are web services?	1
Web service system architecture	1
Web service protocol stack.	2
Simple Object Access Protocol	2
Web Service Description Language	3
Proxies	5
Chapter 2. Directory Management	
Service overview	7
Accessing the Directory Management Service	7
Calling Directory Management Service operations	7
Chapter 3. Directory Management	
Service concepts	9
Directories	9
Configurable directories	9
Manageable directories	10
Principals	10
Roles	10
Chapter 4. Operation reference	13
The addConfiguration operation	13
The createPrincipal operation	15
The createRoleDefinition operation	17
The deletePrincipals operation	18
The deleteRoleDefinition operation	19
The getActionList operation	21
The getConfigurableDirectories operation	22
The getConfiguration operation.	24
The getManageableDirectories operation.	25
The getManageablePrincipals operation	26
The getPrincipalData operation	28
The getPrincipalDataEx operation	30
The getVersion operation	30
The importPrincipals operation	31
The putConfiguration operation	33
The removeConfiguration operation	35
The updateConfiguration operation	36
The updatePrincipal operation	37
The updateRoleDefinition operation	39
Chapter 5. JAX-WS clients	41
Generating a JAX-WS client	41
Packaging a JAX-WS client	41
Configuring a JAX-WS client	41
SOAPHandler example	42
Exercising web services from JAX-WS clients	44
Chapter 6. Microsoft .NET	
Framework-based clients	45
Adding a service reference	45
Service reference modifications	45
Configuring the web service endpoint	46
Configuring endpoint behaviors	47
Exercising the service	47
Single sign-on authentication	48
Chapter 7. Message header reference	49
Security headers.	49
Security element.	49
UsernameToken element	50
BinarySecurityToken and	
BinarySecuritySSOToken elements	50
The client-accept-language element	51
HTTP headers	51
Notices	53
Privacy policy considerations	55
Trademarks	55
Glossary	57
Index	59

Chapter 1. Introduction to web services

What are web services?

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

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

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

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

Web service system architecture

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

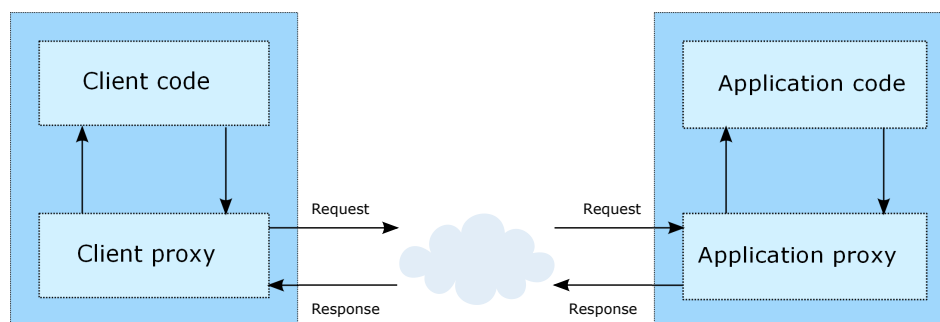


Figure 1. Web service architecture

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

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

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

Web service protocol stack

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

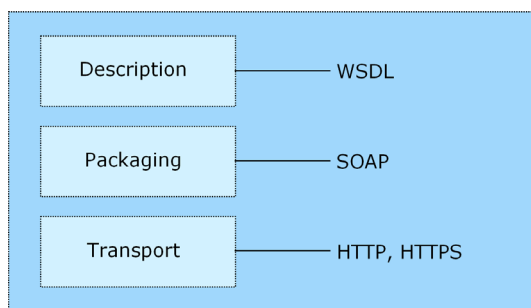


Figure 2. Web service protocol stack

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

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

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

Simple Object Access Protocol

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

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

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

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

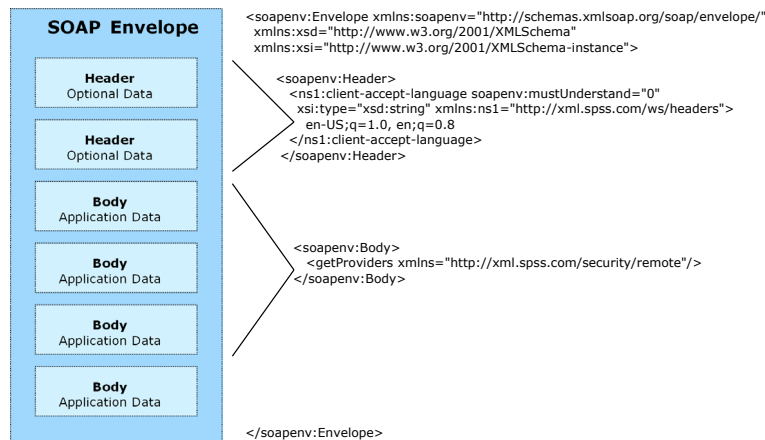


Figure 3. An example SOAP packet

Web Service Description Language

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

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

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

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

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

Types

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

```

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

```

```

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

```

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

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

```

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

```

Messages

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

```

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

```

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

Port types

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

```

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

```

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

Bindings

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

```

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

```



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

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

Services

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

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

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

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

Proxies

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

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

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

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

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

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

Chapter 2. Directory Management Service overview

The Directory Management Service allows control over user and group access to IBM® SPSS® Collaboration and Deployment Services. Specifically, the service offers the ability to perform the following tasks:

- Configure security providers for user authentication
- Create users and groups for the system, and edit their properties
- Assign users and groups to security roles that control access to system functionality

Accessing the Directory Management Service

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

```
http://<host-name>:<port-number>/<context-root>/security-ws/services/DirectoryManagement
```

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

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

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

```
http://cads_server:80/security-ws/services/DirectoryManagement?wsdl
```

Calling Directory Management 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/DirectoryManagement";
URL url = new URL("http", "cads_server", 80, context);
DirectoryManagementService service = new DirectoryManagementServiceLocator();
stub = service.getDirectoryManagement(url);
```

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

```
stub.getVersion();
```

Chapter 3. Directory Management Service concepts

Directories

Available providers include:

- **Native (or local user repository).** The internal security provider for IBM SPSS Collaboration and Deployment Services, in which users, groups, and roles can all be defined. The native provider is always active and cannot be disabled.
- **OpenLDAP®.** An open-source LDAP implementation for authentication, authorization, and security policies. Users and groups for this provider must be defined directly using LDAP tools. After configuring OpenLDAP for use with IBM SPSS Collaboration and Deployment Services, the system can authenticate a user against the OpenLDAP server while maintaining the permissions and access rights associated with that user. In contrast to the native provider, this provider can be enabled or disabled.

Note: OpenLDAP is an open source, reference implementation of LDAP. You can use the OpenLDAP provider to configure and access other directory servers that conform with this protocol, including IBM Security Directory Server.

- **Active Directory®.** The Microsoft version of Lightweight Directory Access Protocol (LDAP) for authentication, authorization, and security policies. Users and groups for this provider must be defined directly in the Active Directory framework. After configuring Active Directory for use with IBM SPSS Collaboration and Deployment Services, the system can authenticate a user against the Active Directory server while maintaining the permissions and access rights associated with that user. This provider can be enabled or disabled. For additional information about Active Directory, see the original vendor's documentation.
- **Active Directory with local override.** A provider that leverages Active Directory but allows the creation of extended groups and allowed-users filters. An extended group contains a list of users from Active Directory but exists outside of the Active Directory framework. An allowed-users filter restricts the list of Active Directory users that can authenticate against the system to a defined set. This provider can be enabled or disabled.

Providers in the system can be characterized by the following properties:

- **ID.** A unique identifier for the provider.
- **Name.** The name of the provider, such as *Native* or *Active Directory with Local Override*. The name typically appears in a Login dialog if multiple providers are enabled to allow a user to select the provider against which to authenticate.
- **Key.** An internal value assigned to the ID/Name combination. Keys can be used in lists of available providers, using the name corresponding to the key in client interfaces.
- **Domain.** The domain within the provider under which users are classified. Some security providers, such as *Native*, do not use domains. For others, the domain returned by the service may be null if the directory only offers one domain.

Configurable directories

Providers that require the specification of configuration parameters before the directory can be accessed are referred to as **configurable directories**. Parameters that must be set vary by provider, but typically include the following:

- **Host URL.** URL for the *Active Directory* server. The default port for LDAP is 389.
- **User Base DN.** Base *distinguished name* for user searches.
- **Group Base DN.** Base distinguished name for group searches.
- **Domain.** The DNS namespace to which the user is logging in.

- **Domain user.** A user ID to perform searches, specified in the format *domain\username*. The name specified must have the proper permissions to look up and authenticate users.
- **Domain user password.** For security, the domain user password specified appears in a hashed asterisk (*) format.

Examples of configurable directories include Active Directory and OpenLDAP.

The Directory Management Service includes operations for retrieving configurable directories and for specifying configuration parameters.

Manageable directories

Users and groups are typically defined within the directories themselves. For example, a *sales* group would be defined directly within Active Directory using its native tools. You cannot define a group within Active Directory using IBM SPSS Collaboration and Deployment Services.

Directories that allow IBM SPSS Collaboration and Deployment Services to define users and groups are referred to as **manageable directories**. These directories provide a list of allowed principal types which can be added. For example, a manageable directory may only allow the specification of users and groups, but not roles. The principals in these directories are also referred to as manageable, denoting that they can be modified and deleted using IBM SPSS Collaboration and Deployment Services. Examples of manageable directories include Active Directory with Local Override and the Local User Repository.

The Directory Management Service includes an operation for retrieving manageable directories. Principals can then be added to any of the returned directories. In addition, you can retrieve the list of manageable principals for any manageable directory.

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.

Principals of the *user* type also include a password used for authentication against a security provider.

The Directory Management Service includes operations for creating, importing, updating, and deleting principals. In addition, you can retrieve information for a specified principal.

Roles

Roles provide a way to manage user and group access to system functionality. Roles are assigned to users and groups and work in conjunction with a security provider.

Each role created has associated actions that represent the permissions and level of control that the user or group assigned to the role has. For example, a basic user role can be created. The basic user role is assigned a limited set of actions for access to the system and the ability to view the contents of the repository. The basic user role does not have the associated actions to define servers, add other users, or define system configurations that would impact other users and groups.

However, an advanced user role is needed to perform administrative tasks, such as deleting users, creating groups, and defining additional roles. In this case, a less restricted role can be created with more control over the application domain and assigned to a very small set of users.

The list of available actions are defined within the system and cannot be edited by the user assigning them.

If the user belongs to several groups, the roles assigned to that user—an action set—consist of all roles explicitly assigned to the user as well as all roles indirectly assigned through group membership. If the user or group is assigned to several roles, the user or group's action set consists of all roles explicitly assigned as well as all roles indirectly assigned through group membership. Users and groups must be managed per security provider, whereas roles are managed across security providers.

The Directory Management Service includes operations for creating, updating, and deleting roles. In addition, you can retrieve the list of actions associated with a role.

Chapter 4. Operation reference

The addConfiguration operation

Creates a configurable security provider having specified configuration parameters. Use the `getConfiguration` operation to access the configuration information for a specific configurable directory. To modify the configuration parameters after creating the provider, use the `updateConfiguration` operation.

Input fields

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

Table 1. Fields for `addProviderConfiguration`.

Field	Type/Valid Values	Description
<code>providerConfiguration</code>	<code>providerConfiguration</code>	Configuration parameters. For each parameter, the information returned includes the name, ID, description, and value.

Return information

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

Table 2. Return Value.

Type	Description
string	General purpose response. Just a string.

SOAP request example

Client invocation of the `addConfiguration` operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsse:Username>Native/validUser</wsse:Username>
        <wsse:Password wsse:Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">
          password</wsse:Password>
        <wsse:Nonce>qwjJN5dMn99rsqCso0qfQw=</wsse:Nonce>
        <wsu:Created xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
          2011-08-31T14:39:13Z</wsu:Created>
        </wsse:UsernameToken>
      </wsse:Security>
    <ns1:client-accept-language soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      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>
    <addProviderConfiguration xmlns="http://xml.spss.com/security/remote">
      <providerConfiguration enabled="false" canDisable="true"
        name="MyAD" id="AD_MyAD" xmlns="http://xml.spss.com/security">
        <providerConfigItem type="text" name="Host URL" id="host">
          <description>LDAP server, for example:
```

```

    ldap://ldap.mycompany.com</description>
<value>ldap://ldap.myfirm.com</value>
</providerConfigItem>
<providerConfigItem type="checkbox" name="Use SSL" id="useSSL">
  <description>Whether or not this connection should use
  SSL</description>
  <value>>false</value>
</providerConfigItem>
<providerConfigItem type="checkbox" name="Page Search Result" id="usePaging">
  <description>Whether or not this connection should page
  search result</description>
  <value>>true</value>
</providerConfigItem>
<providerConfigItem type="checkbox" name="Use anonymous bind during search"
id="useAnonymousBind">
  <description>Whether or not this connection uses
  anonymous bind during search operation. When set to
  "true" Search User ID and it's password is ignored if
  specified.</description>
  <value>>false</value>
</providerConfigItem>
<providerConfigItem type="text" name="Search User ID" id="domainUser">
  <description>UserID of a valid user for directory
  searches, for example:
  cn=manager,dc=myorg,dc=myfirm,dc=com</description>
  <value>mydomain\userID</value>
</providerConfigItem>
<providerConfigItem type="password" name="Search User Password" id="domainUserPassword">
  <description>Password for that user.</description>
  <value>password</value>
</providerConfigItem>
<providerConfigItem type="checkbox" name="Use Context Bind" id="UContextBindg">
  <description>Login by Context Bind. When false use
  password compare method</description>
  <value>>true</value>
</providerConfigItem>
<providerConfigItem type="text" name="User Password Attribute" id="UPasswordAttribute">
  <description>When using password compare method, this
  field must be returned by the LDAP server</description>
  <value>userPassword</value>
</providerConfigItem>
<providerConfigItem type="text" name="User Password Digest" id="UPasswordDigest">
  <description>When using password compare method, this is
  the hash mechanism to encode password by LDAP
  server</description>
  <value>SHA</value>
</providerConfigItem>
<providerConfigItem type="text" name="User Base DN" id="UbaseDN">
  <description>Base user distinguished name, for example:
  ou=users,ou=myorg,dc=mycompany,dc=com</description>
  <value>ou=users,ou=myorg,dc=mycompany,dc=com</value>
</providerConfigItem>
<providerConfigItem type="text" name="User Object Filter Expression"
id="UObjectFilterExpr">
  <description>User Object Filter Expression</description>
  <value>(objectCategory=user)</value>
</providerConfigItem>
<providerConfigItem type="text" name="User Search Filter Expression"
id="USearchFilterExpr">
  <description>User Search Filter Expression</description>
  <value>(sAMAccountName=$ID)</value>
</providerConfigItem>
<providerConfigItem type="text" name="User Attribute" id="UUniqueNameAttribute">
  <description>User Attribute, for example: if search by
  "uid" attribute uid</description>
  <value>sAMAccountName</value>
</providerConfigItem>
<providerConfigItem type="text" name="Search Group Using Group Membership"
id="UsearchGroupMembershipAttribute">
  <description>Search Group Using Group
  Membership</description>
  <value>memberOf</value>
</providerConfigItem>
<providerConfigItem type="text" name="Group Base DN" id="GbaseDN">
  <description>Base group distinguished name, for example:
  ou=groups,ou=myorg,dc=mycompany,dc=com</description>
  <value>ou=groups,ou=myorg,dc=mycompany,dc=com</value>
</providerConfigItem>
<providerConfigItem type="text" name="Group Object Filter Expression"
id="GObjectFilterExpr">
  <description>Group Object Filter Expression</description>
  <value>(objectCategory=group)</value>
</providerConfigItem>
<providerConfigItem type="text" name="Group Search Filter Expression"

```

```

id="GObjectSearchExpr">
  <description>Group Search Filter Expression</description>
  <value>(sAMAccountName=$ID)</value>
</providerConfigItem>
<providerConfigItem type="text" name="Group Attribute" id="GUniqueNameAttribute">
  <description>Group Attribute, for example: if search by
common name "cn" attribute cn</description>
  <value>sAMAccountName</value>
</providerConfigItem>
<providerConfigItem type="text" name="Group Membership Attribute"
id="GMembershipAttribute">
  <description>Group Membership Attribute, for example:
member</description>
  <value>member</value>
</providerConfigItem>
<providerConfigItem type="text" name="Group Refresh Period" id="GRefreshPeriod">
  <description></description>
  <value>300000</value>
</providerConfigItem>
<providerConfigItem type="text" name="Domain" id="domain">
  <description>Active Directory Domain Name</description>
  <value>mydomain</value>
</providerConfigItem>
<providerConfigItem type="checkbox" name="Allowed Users" id="allowedUsers">
  <description>Only allow users in a locally defined list
to be authenticated by Active Directory, and enable
definition of this list in Users and
Groups.</description>
  <value>true</value>
</providerConfigItem>
<providerConfigItem type="checkbox" name="Extended Groups" id="extendedGroups">
  <description>Allow groups of Active Directory users to be
defined. Active Directory users may be assigned to these
locally defined groups.</description>
  <value>true</value>
</providerConfigItem>
</providerConfiguration>
</addProviderConfiguration>
</soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a `addConfiguration` 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>
    <addProviderConfigurationResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:status xmlns:ns1="http://xml.spss.com/security"></ns1:status>
    </addProviderConfigurationResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

The createPrincipal operation

Creates a new user, group, or role in the system. The principal being added can be associated with other principals already existing within the system. For example:

- New users can be associated with existing groups and roles.
- New groups can be associated with existing users and roles.
- New roles can be associated with existing users and groups.

Input fields

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

Table 3. Fields for createPrincipal.

Field	Type/Valid Values	Description
newPrincipal	newPrincipal	Information about a principal being created. This information includes: a provider ID, a user ID and password, the principal type, and a list of principals to associate with the new principal.

Return information

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

Table 4. Return Value.

Type	Description
string	An internal identifier for a principal.

Java example

Creating a principal involves the specification of the following pieces of information:

- An identifier for the security provider that will contain the principal
- The principal type
- A user ID for the principal
- A password for the principal
- An optional list of associated principals

This information is contained within a NewPrincipal object.

The following sample creates a new user principal with the ID *bbrever* in the native security provider.

```
NewPrincipal np = new NewPrincipal();
np.setProviderID("Native");
np.setType(PrincipalType.USER);
np.setUserID("bbrever");
np.setUserPassword("qw12as3z");
stub.createPrincipal(np);
```

SOAP request example

Client invocation of the createPrincipal 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>
```

```

<createPrincipal xmlns="http://xml.spss.com/security/remote">
  <newPrincipal providerID="Native" userID="bbrever" userPassword="qw12as3z"
    type="user" xmlns="http://xml.spss.com/security"/>
</createPrincipal>
</soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a createPrincipal 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>
    <createPrincipalResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:principalID xmlns:ns1="http://xml.spss.com/security"/>//uNative//bbrever</ns1:principalID>
    </createPrincipalResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

The createRoleDefinition operation

Creates a new named role in the system. See the topic “Roles” on page 10 for more information.

Input fields

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

Table 5. Fields for createRoleDefinition.

Field	Type/Valid Values	Description
newRoleDefinition	newRoleDefinition	Definition of the role, including the role name and a list of associated actions. Use internal action identifiers to specify the actions to associate with the role.

Return information

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

Table 6. Return Value.

Type	Description
string	General purpose response. Just a string.

Java example

Creating a role requires two pieces of information:

- A name for the role
- The actions to associate with the role

This information is contained within a NewRoleDefinition object.

The following sample creates an array of strings corresponding to action identifiers. The setActionID method assigns these identifiers to the NewRoleDefinition object. The setName method assigns the name *newRole* to the role. Use the createRoleDefinition operation to apply the changes to the system.

```

NewRoleDefinition nrd = new NewRoleDefinition();
nrd.setName("newRole");
String[] actionArray = {

```

```

"security/roleDefinition",
"security/manage",
});
nrd.setActionID(actionArray); stub.createRoleDefinition(nrd);

```

SOAP request example

Client invocation of the createRoleDefinition 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>
    <createRoleDefinition xmlns="http://xml.spss.com/security/remote">
      <newRoleDefinition xmlns="http://xml.spss.com/security">
        <actionID>security/roleDefinition</actionID>
        <actionID>security/manage</actionID>
        <name>newRole</name>
      </newRoleDefinition>
    </createRoleDefinition>
  </soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a createRoleDefinition 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>
    <createRoleDefinitionResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:status xmlns:ns1="http://xml.spss.com/security"></ns1:status>
    </createRoleDefinitionResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

The deletePrincipals operation

Removes one or more principals from the system. This operation may fail if a principal to be removed is in a remote directory, or if removing a specified principal would result in no remaining administrative users.

Input fields

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

Table 7. Fields for deletePrincipals.

Field	Type/Valid Values	Description
principalIDList	principalIDList	List of IDs for principals to be deleted.

Return information

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

Table 8. Return Value.

Type	Description
string	General purpose response. Just a string.

Java example

Deleting a principal involves the creation of a `PrincipalIDList` object to contain the identifiers for the principals to be deleted. Supply this object to the `deletePrincipals` operation. The following sample deletes a single principal having an identifier of `principalID`.

```
PrincipalIDList principalIDList = new PrincipalIDList();
principalIDList.addPrincipalID(principalID);
directoryManagement.deletePrincipals(principalIDList);
```

SOAP request example

Client invocation of the `deletePrincipals` 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>
    <deletePrincipals xmlns="http://xml.spss.com/security/remote">
      <principalIDList xmlns="http://xml.spss.com/security">
        <principalID>//uNative//bbrevert</principalID>
      </principalIDList>
    </deletePrincipals>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deletePrincipals` 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>
    <deletePrincipalsResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:status xmlns:ns1="http://xml.spss.com/security"/>
    </deletePrincipalsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The `deleteRoleDefinition` operation

Deletes an existing role from the system.

Input fields

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

Table 9. Fields for deleteRoleDefinition.

Field	Type/Valid Values	Description
roleID	roleID	An internal identifier for a role.
name	string	

Return information

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

Table 10. Return Value.

Type	Description
string	General purpose response. Just a string.

Java example

Deleting a role involves the creation of a RoleID object to contain the identifier for the role to be deleted. Use setID to assign the identifier for the role. Supply this object to the deleteRoleDefinition operation. The following sample deletes the role with an identifier of *roleDefinitionID*.

```
RoleID roleID = new RoleID();
    String id = principalInfo.getID();
    roleID.setID(id);
    directoryManagement.deleteRoleDefinition(roleID);
```

SOAP request example

Client invocation of the deleteRoleDefinition 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>
    <deleteRoleDefinition xmlns="http://xml.spss.com/security/remote">
      <roleID ID="//rNative/newRole"/>
    </deleteRoleDefinition>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a deleteRoleDefinition 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>
    <deleteRoleDefinitionResponse xmlns="http://xml.spss.com/security/remote">
```



```

    <ns1:status xmlns:ns1="http://xml.spss.com/security"/>
  </deleteRoleDefinitionResponse>
</soapenv:Body>
</soapenv:Envelope>

```

The getActionList operation

Retrieves the list of actions available in the system. The actions returned are typically used for defining roles.

Input fields

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

Table 11. Fields for getActionList.

Field	Type/Valid Values	Description
actionListRequest	string	

Return information

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

Table 12. Return Value.

Type	Description
actionList	A list of all current (licensed) actions. This is very static and can be safely be cached in a client.

Java example

The following example function calls getActionList from the stub for the service, returning an array of actions for the specified principal.

```

...
ActionList list = null;

try {
    list = stub.getActionList(null);

    testActionList(list.getActionDetail());
} catch (RemoteException e) {
    fail(e);
}

return list;
}

```

SOAP request example

Client invocation of the getActionList 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>
  <getActionList xmlns="http://xml.spss.com/security/remote">
    <ns2:actionListRequest xsi:nil="true" xmlns:ns2="http://xml.spss.com/security"/>
  </getActionList>
</soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a `getActionList` 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>
    <getActionListResponse xmlns="http://xml.spss.com/security/remote">
      <actionList xmlns="http://xml.spss.com/security">
        <actionDetail name="Custom Properties" id="contentRepository/customProperties">
          <description>Custom Properties</description>
        </actionDetail>
        <actionDetail name="Manage Users, Groups and Roles" id="security/manage">
          <description>Manage Users, Groups and Roles</description>
        </actionDetail>
        <actionDetail name="crUI/apFolders" id="contentRepository/folders">
          <description>Content and Folders</description>
        </actionDetail>
        <actionDetail name="crUI/apServers" id="contentRepository/servers">
          <description>Servers</description>
        </actionDetail>
        <actionDetail name="Export Content" id="contentRepository/export">
          <description>Export Content</description>
        </actionDetail>
        <actionDetail name="Import Content" id="contentRepository/import">
          <description>Import Content</description>
        </actionDetail>
        <actionDetail name="MIME Types" id="configuration/MimeManager">
          <description>Manage MIME types</description>
        </actionDetail>
        <actionDetail name="Define Roles" id="security/roleDefinition">
          <description>Manage Actions associated with Roles</description>
        </actionDetail>
        <actionDetail name="Configuration" id="configuration/Editor">
          <description>Customize Configuration Settings</description>
        </actionDetail>
        <actionDetail name="Topics" id="contentRepository/topics">
          <description>Manage Topics</description>
        </actionDetail>
        <actionDetail name="Schedules" id="prms/schedules">
          <description>PRMS Schedules</description>
        </actionDetail>
        <actionDetail name="Jobs" id="prms/jobs">
          <description>PRMS Jobs</description>
        </actionDetail>
        <actionDetail name="crUI/apCredentials" id="contentRepository/credentials">
          <description>Credentials</description>
        </actionDetail>
        <actionDetail name="Configure Security Providers" id="security/config">
          <description>Configure Security Providers</description>
        </actionDetail>
      </actionList>
    </getActionListResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

The `getConfigurableDirectories` operation

Returns a list of directories that can be configured for use with the system. Use the `getConfigurableDirectories` operation to access the configuration information for a specific configurable directory. To specify configuration parameters for a configurable directory, use the `putConfigurableDirectory` operation.

Return information

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

Table 13. Return Value.

Type	Description
<code>configurableProviders</code>	List of available configurable providers. For each provider, the list includes the name, ID, and a boolean indicating whether or not the provider is enabled.

Java example

The following function uses the `getConfigurableDirectories` operation to return an array of `ConfigurableProvider` objects. Each object includes:

- The provider identifier
- The provider name
- An indicator of whether or not the provider is enabled

The information can be extracted as needed using the corresponding `get` method, such as `getId`.

```
public ConfigurableProvider[] getConfigurableDirectories()
    throws RemoteException, IOException, ServiceException {
    ConfigurableProviders providers = stub.getConfigurableDirectories();
    return providers.getConfigurableProvider();
}
```

SOAP request example

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

SOAP response example

The server responds to a `getConfigurableDirectories` 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>
    <getConfigurableDirectoriesResponse xmlns="http://xml.spss.com/security/remote">
      <configurableProviders xmlns="http://xml.spss.com/security">
        <configurableProvider enabled="true" name="Native" id="Native"/>
        <configurableProvider enabled="true" name="Active Directory" id="AD"/>
        <configurableProvider enabled="true" name="Active Directory with Local Override"
          id="ADL"/>
        <configurableProvider enabled="false" name="OpenLDAP" id="devldapOpenLDAP"/>
      </configurableProviders>
    </getConfigurableDirectoriesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

```

    </configurableProviders>
  </getConfigurableDirectoriesResponse>
</soapenv:Body>
</soapenv:Envelope>

```

The getConfiguration operation

Retrieves the configuration parameters for a specified configurable directory as a set of providerConfigItems. To modify the parameters, use the putConfiguration operation.

Input fields

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

Table 14. Fields for getProviderConfiguration.

Field	Type/Valid Values	Description
providerID	string	An internal identifier for a provider.

Return information

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

Table 15. Return Value.

Type	Description
providerConfiguration	Configuration parameters. For each parameter, the information returned includes the name, ID, description, and value.

Java example

The following sample uses getConfiguration to return the configuration settings for the Active Directory provider, which has an ID of AD. The isEnabled method of the returned object indicates whether or not the Active Directory provider is enabled. If so, getProviderConfigItem returns an array containing the configuration items. This array can subsequently be queried for individual parameters and their values.

```

String id = "AD";
ProviderConfiguration proConfig = stub.getConfiguration(id);
if (proConfig.isEnabled()) {
    ProviderConfigItem[] pcItems = proConfig.getProviderConfigItem();
}

```

SOAP request example

Client invocation of the getConfiguration 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>
    <getProviderConfiguration xmlns="http://xml.spss.com/security/remote">

```

```

        <ns2:providerID xmlns:ns2="http://xml.spss.com/security">AD</ns2:providerID>
    </getProviderConfiguration>
</soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a `getConfiguration` 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>
    <getProviderConfigurationResponse xmlns="http://xml.spss.com/security/remote">
      <providerConfiguration enabled="true" canDisable="true"
        name="$$security/providerActiveDirectory" id="AD"
        xmlns="http://xml.spss.com/security">
        <providerConfigItem type="text" name="Host URL" id="host">
          <description>Active Directory Server host URL, for example:
            ldap://domaincontroller.mycompany.com:389</description>
          <value>ldap://aaw2kbb1.spss.com</value>
        </providerConfigItem>
        <providerConfigItem type="text" name="User Base DN" id="UbaseDN">
          <description>Base distinguished name, for example: OU=USERS,
            OU=myorganization, DC=mycompany,DC=com</description>
          <value>OU= USERS, OU= CHICAGO, DC=spss, DC=com</value>
        </providerConfigItem>
        <providerConfigItem type="text" name="Group Base DN" id="GbaseDN">
          <description>Base distinguished name, for example: OU=GROUPS,
            OU=myorganization, DC=mycompany,DC=com</description>
          <value>OU = USERS, OU= CHICAGO, DC=spss, DC=com</value>
        </providerConfigItem>
        <providerConfigItem type="text" name="Domain" id="domain">
          <description>Active Directory Domain Name</description>
          <value>mydomain</value>
        </providerConfigItem>
        <providerConfigItem type="text" name="Domain User" id="domainUser">
          <description>UserID of a valid user for directory searches.</description>
          <value>mydomain\kkrueter</value>
        </providerConfigItem>
        <providerConfigItem type="password" name="Domain User Password"
          id="domainUserPassword">
          <description>Password for that user.</description>
          <value>password</value>
        </providerConfigItem>
      </providerConfiguration>
    </getProviderConfigurationResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

The `getManageableDirectories` operation

Returns a list of manageable directories currently in use in the system.

Return information

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

Table 16. Return Value.

Type	Description
<code>manageableProviders</code>	List of manageable providers. For each provider, the returned information includes the name, ID, and list of principal types that can be defined.

Java example

The following function uses the `getManageableDirectories` operation to return an array of `ManageableProvider` objects. Each object includes:

- The provider identifier

- The provider name
- An indicator of whether or not the provider allows importing of principals
- A list of allowed principal types

The information can be extracted as needed using the corresponding get method, such as getId.

```
public ManageableProvider[] getManageableDirectories()
    throws RemoteException, IOException, ServiceException {
    ManageableProviders providers = stub.getManageableDirectories();
    return providers.getManageableProvider();
}
```

SOAP request example

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

SOAP response example

The server responds to a getManageableDirectories 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>
    <getManageableDirectoriesResponse xmlns="http://xml.spss.com/security/remote">
      <manageableProviders xmlns="http://xml.spss.com/security">
        <manageableProvider canImport="true" name="Local User Repository" id="Native">
          <allowablePrincipalTypes>
            <principalType>role</principalType>
            <principalType>user</principalType>
            <principalType>group</principalType>
          </allowablePrincipalTypes>
        </manageableProvider>
        <manageableProvider canImport="false"
          name="Active Directory with Local Override" id="ADL">
          <allowablePrincipalTypes>
            <principalType>group</principalType>
          </allowablePrincipalTypes>
        </manageableProvider>
      </manageableProviders>
    </getManageableDirectoriesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getManageablePrincipals operation

Returns a list of principals that are defined for manageable directories. The list can be restricted to:

- A specific provider
- A type of principal, such as user or role
- Principals with names beginning with a specified string

See the topic “Manageable directories” on page 10 for more information.

Input fields

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

Table 17. Fields for `getManageablePrincipals`.

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

Return information

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

Table 18. Return Value.

Type	Description
<code>principalList</code>	List of principals. For each principal, the information returned includes the display name, ID, and principal type.

Java example

The following sample uses the `getManageablePrincipals` operation to return a list of principals for the *Native* security provider. The `getPrincipalInfo` method returns general information available for any principal, regardless of type. The `getIsRole`, `getIsUser`, and `getIsGroup` methods each return a boolean indicating whether or not the principal is of the corresponding type, which can be used to determine what additional information is available, if any.

```
DirectoryCriterion directoryCriterion = new DirectoryCriterion();
directoryCriterion.setProviderKey("Native");
PrincipalList pList = stub.getManageablePrincipals(directoryCriterion);
int count = pList.getPrincipalInfoCount();
PrincipalList retList = new PrincipalList();
PrincipalInfo pi = null;
for (int i = 0; i < count; i++) {
    pi = pList.getPrincipalInfo(i);
    if (pi.getIsRole()) {
        retList.addPrincipalInfo(pi);
    }
}
```

SOAP request example

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

```

    </directoryCriterion>
  </getManageablePrincipals>
</soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a `getManageablePrincipals` 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>
    <getManageablePrincipalsResponse xmlns="http://xml.spss.com/security/remote">
      <principalList id="Native" xmlns="http://xml.spss.com/security">
        <principalInfo ID="//uNative/admin" isUser="true" isGroup="false"
          isRole="false" principalType="user" displayName="admin"
          typeName="security/principalTypeUser"/>
        <principalInfo ID="//rNative/roleAdministrators" isUser="false"
          isGroup="false" isRole="true" principalType="role"
          displayName="administrators" typeName="security/principalTypeRole"/>
        <principalInfo ID="//uNative/bbrewer" isUser="true" isGroup="false"
          isRole="false" principalType="user" displayName="bbrewer"
          typeName="security/principalTypeUser"/>
        <principalInfo ID="//uNative/kkrueter" isUser="true" isGroup="false"
          isRole="false" principalType="user" displayName="kkrueter"
          typeName="security/principalTypeUser"/>
        <principalInfo ID="//rNative/pemUser" isUser="false" isGroup="false"
          isRole="true" principalType="role" displayName="pemUser"
          typeName="security/principalTypeRole"/>
        <allowablePrincipalTypes>
          <principalType>user</principalType>
          <principalType>group</principalType>
          <principalType>role</principalType>
        </allowablePrincipalTypes>
      </principalList>
    </getManageablePrincipalsResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

The getPrincipalData operation

Retrieves information for a specified user, group, or role, such as the principal type or associations with other principals.

Input fields

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

Table 19. Fields for getPrincipalData.

Field	Type/Valid Values	Description
principalID	string	An internal identifier for a principal.

Return information

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

Table 20. Return Value.

Type	Description
principalData	Information about a principal, including its type, display name, and list of associated principals. Additional information depends on the principal type. In addition, for roles, <code>principalData</code> includes associated actions.

Java example

The following sample returns the list of actions associated with a specified role. The `getPrincipalData` operation returns the data for the role as a `PrincipalData` object. The `getPrincipalDataRole` method returns information specific for roles as a `PrincipalDataRole` object, from which information about the actions can be retrieved.

```
public ActionList getActionsForRole(String roleId)
    throws RemoteException, IOException, ServiceException {
    PrincipalData pd = stub.getPrincipalData(roleId);
    PrincipalDataRole pdr = pd.getPrincipalDataRole();
    ActionList list = new ActionList();
    int count = pdr.getActionDetailCount();
    for (int i = 0 ; i < count; i++) {
        list.addActionDetail(pdr.getActionDetail(i));
    }
    return list;
}
```

SOAP request example

Client invocation of the `getPrincipalData` 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>
    <getPrincipalData xmlns="http://xml.spss.com/security/remote">
      <ns2:principalID xmlns:ns2="http://xml.spss.com/security">//rNative//newRole</ns2:principalID>
    </getPrincipalData>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getPrincipalData` 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>
    <getPrincipalDataResponse xmlns="http://xml.spss.com/security/remote">
      <principalData name="kkreuter" xmlns="http://xml.spss.com/security">
        <principalDataUser password=".iCA-blca,Soda,Cracker"/>
        <principalInfo ID="//uNative//kkreuter" isUser="true" isGroup="false"
          isRole="false" principalType="user" displayName="kkroeger" typeName="user"/>
        <associatedPrincipals>
          <principalInfo ID="//rNative//$$security/roleAdministrators" isUser="false"
            isGroup="false" isRole="true" principalType="role" displayName="administrators"
            typeName="role"/>
        </associatedPrincipals>
        <allowablePrincipalTypes>
          <principalType>group</principalType>
          <principalType>role</principalType>
        </allowablePrincipalTypes>
      </principalData>
    </getPrincipalDataResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getPrincipalDataEx operation

Input fields

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

Table 21. Fields for getPrincipalDataEx.

Field	Type/Valid Values	Description
principalID	string	An internal identifier for a principal.
additionalAttributes	string[]	

Return information

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

Table 22. Return Value.

Type	Description
principalData	Information about a principal, including its type, display name, and list of associated principals. Additional information depends on the principal type. In addition, for roles, principalData includes associated actions.

The getVersion operation

Returns the version number of the service.

Return information

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

Table 23. Return Value.

Type	Description
string	The version of the web service.

Java example

The following code uses the WServiceConnections 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>

```

The importPrincipals operation

Imports principals defined in an external file into the system. The file being imported must adhere to the structure defined by the schema `nativestore.xsd`.

Input fields

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

Table 24. Fields for `importPrincipals`.

Field	Type/Valid Values	Description
<code>importPrincipals</code>	<code>importPrincipals</code>	Data to bulk load into the Local User Repository. The actual data is in an attached file. The input includes a mode property with one of two values: update or replace. Update mode adds new principals to the existing set. Replace mode deletes existing principals before adding new ones.

Return information

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

Table 25. Return Value.

Type	Description
<code>statusDetails</code>	Summary of changes made to the Local User Repository, such as the number of users and groups added and the number of users and groups made obsolete.

Java example

Importing principals defined in an external file requires the specification of two pieces of information:

- The name of the file containing the principals to import
- The behavior, or **mode**, for the import

To import principals, create an `ImportPrincipals` object. Use the `setMode` method to define the import mode. Create an attachment for the file to import and call the `importPrincipals` operation.

```
DirectoryManagement directoryManagement = wsConnections.getDirectoryManagement();

    ImportPrincipals importPrincipals = new ImportPrincipals();
    importPrincipals.setMode(ImportPrincipalsModeType.UPDATE);

    // Create an attachment for the binary content
    java.io.File exampleImport = new java.io.File("exampleImport.xml");
    FileDataSource dataSource = new FileDataSource(exampleImport);
    DataHandler dataHandler = new DataHandler(dataSource);
    AttachmentPart attachmentPart = new AttachmentPart(dataHandler);
    Attachment attachment = new Attachment();
    attachment.setHref(attachmentPart.getContentId());

    // Do Axis stub hack to add the attachment
    ((org.apache.axis.client.Stub) directoryManagement).addAttachment(attachmentPart);

    directoryManagement.importPrincipals(importPrincipals);
```

SOAP request example

Client invocation of the `importPrincipals` 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>
    <importPrincipals xmlns="http://xml.spss.com/security/remote">
      <importPrincipals mode="update" xmlns="http://xml.spss.com/security"/>
    </importPrincipals>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `importPrincipals` 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>
    <importPrincipalsResponse xmlns="http://xml.spss.com/security/remote">
      <importPrincipalsStatus success="true" xmlns="http://xml.spss.com/security">
        <statusItem name="New Groups">
          <value>2</value>
        </statusItem>
        <statusItem name="New Users">
          <value>4</value>
        </statusItem>
        <statusItem name="Obsolete Groups">
          <value>1</value>
        </statusItem>
        <statusItem name="Obsolete Users">
          <value>2</value>
        </statusItem>
      </importPrincipalsStatus>
    </importPrincipalsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

```

    </importPrincipalsStatus>
  </importPrincipalsResponse>
</soapenv:Body>
</soapenv:Envelope>

```

The putConfiguration operation

Assigns values to configuration parameters for configurable directories. To retrieve a list of parameters that can be set for a particular directory, use the getConfiguration operation.

Input fields

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

Table 26. Fields for putProviderConfiguration.

Field	Type/Valid Values	Description
providerConfigurationUpdate	providerConfigurationUpdate	A set of configuration item/value pairs for a specific provider, identified by its ID.

Return information

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

Table 27. Return Value.

Type	Description
string	General purpose response. Just a string.

Java example

Configuring a security provider requires three primary pieces of information:

- The identifier for the provider being configured
- An indicator of whether or not to enable the provider
- A set of configuration parameters with assigned values

This information is contained within a ProviderConfigurationUpdate object.

The following sample configures an Active Directory with Local Override provider, which has an identifier of ADL. The setId method assigns this identifier to a ProviderConfigurationUpdate object for the configuration information. The value of true for the setEnabled method specifies that the provider should be enabled for use in the system.

An array of ProviderItemValue objects contains the configuration items and their values. For each item, use the setId and setValue methods to define the identifier and value. After defining all items, use the setProviderItemValue method to assign the items to the ProviderConfigurationUpdate object.

```

ProviderConfigurationUpdate config = new ProviderConfigurationUpdate();
config.setId("ADL");
config.setEnabled(true);
ProviderItemValue[] piv;
piv = new ProviderItemValue[8];
piv[0].setId("domainUserPassword");
piv[0].setValue("qw12as34%");
piv[1].setId("UbaseDN");
piv[1].setValue("OU = USERS, OU= NEWYORK, DC=com");
piv[2].setId("domainUser");
piv[2].setValue("kkreuter");
piv[3].setId("domain");
piv[3].setValue("mydomain");

```

```

piv[4].setId("host");
piv[4].setValue("ldap://aaw2kbb1.spss.com");
piv[5].setId("principalFilter");
piv[5].setValue("false");
piv[6].setId("localGroups");
piv[6].setValue("true");
piv[7].setId("GbaseDN");
piv[7].setValue("OU = USERS, OU= NEWYORK, DC=com");
config.setProviderItemValue(piv);
getDirectoryManagement().putConfiguration(config);

```

Use the putConfiguration operation to add the configuration information to the system.

SOAP request example

Client invocation of the putConfiguration 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>
    <putProviderConfiguration xmlns="http://xml.spss.com/security/remote">
      <providerConfigurationUpdate ID="ADL" enabled="true"
        xmlns="http://xml.spss.com/security">
        <providerItemValue id="domainUserPassword">
          <value>qw12as34%</value>
        </providerItemValue>
        <providerItemValue id="UbaseDN">
          <value>OU = USERS, OU= NEWYORK, DC=com</value>
        </providerItemValue>
        <providerItemValue id="domainUser">
          <value>mydomain\kkreuter</value>
        </providerItemValue>
        <providerItemValue id="domain">
          <value>mydomain</value>
        </providerItemValue>
        <providerItemValue id="host">
          <value>ldap://aaw2kbb1.spss.com</value>
        </providerItemValue>
        <providerItemValue id="principalFilter">
          <value>>false</value>
        </providerItemValue>
        <providerItemValue id="localGroups">
          <value>>true</value>
        </providerItemValue>
        <providerItemValue id="GbaseDN">
          <value>OU = USERS, OU= NEWYORK, DC=com</value>
        </providerItemValue>
      </providerConfigurationUpdate>
    </putProviderConfiguration>
  </soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a putConfiguration 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>
    <putProviderConfigurationResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:status xmlns:ns1="http://xml.spss.com/security"/>
    </putProviderConfigurationResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

The removeConfiguration operation

Deletes a security provider from the system. Credentials defined in the deleted provider will no longer be able to access IBM SPSS Collaboration and Deployment Services.

Input fields

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

Table 28. Fields for removeProviderConfiguration.

Field	Type/Valid Values	Description
providerID	string	An internal identifier for a provider.

Return information

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

Table 29. Return Value.

Type	Description
string	General purpose response. Just a string.

SOAP request example

Client invocation of the removeConfiguration operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsse:Username>Native//validUser</wsse:Username>
        <wsse:Password wsse:Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">
          password</wsse:Password>
        <wsse:Nonce>qwjJN5dMn99rsqCso0qfQw=</wsse:Nonce>
        <wsu:Created xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
          2011-08-31T14:39:13Z</wsu:Created>
        </wsse:UsernameToken>
      </wsse:Security>
      <ns1:client-accept-language soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
        soapenv:mustUnderstand="0"
        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>
      <removeProviderConfiguration xmlns="http://xml.spss.com/security/remote">
        <ns2:providerID xmlns:ns2="http://xml.spss.com/security">AD_MyAD</ns2:providerID>
      </removeProviderConfiguration>
    </soapenv:Body>
  </soapenv:Envelope>
```

SOAP response example

The server responds to a removeConfiguration 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>
    <removeProviderConfigurationResponse xmlns="http://xml.spss.com/security/remote">
```

```

    <ns1:status xmlns:ns1="http://xml.spss.com/security"></ns1:status>
  </removeProviderConfigurationResponse>
</soapenv:Body>
</soapenv:Envelope>

```

The updateConfiguration operation

Modifies the configuration parameters of an existing security provider.

Input fields

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

Table 30. Fields for updateProviderConfiguration.

Field	Type/Valid Values	Description
providerConfigurationUpdate	providerConfigurationUpdate	A set of configuration item/value pairs for a specific provider, identified by its ID.

Return information

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

Table 31. Return Value.

Type	Description
string	General purpose response. Just a string.

SOAP request example

Client invocation of the updateConfiguration operation generates a SOAP request message that is sent to the server for processing. An example of such a message follows.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsse:Username>Native/validUser</wsse:Username>
        <wsse:Password wsse:Type=
          "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">
          password</wsse:Password>
        <wsse:Nonce>qwjN5dMn99rsqCso0qfQw=</wsse:Nonce>
        <wsu:Created xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
          2011-08-31T14:39:13Z</wsu:Created>
        </wsse:UsernameToken>
      </wsse:Security>
    <ns1:client-accept-language soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      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>
    <updateProviderConfiguration xmlns="http://xml.spss.com/security/remote">
      <providerConfigurationUpdate ID="AD_MyAD" enabled="false"
        xmlns="http://xml.spss.com/security">
        <providerItemValue id="host"><value>ldap://ldap.myfirm.com</value>
        </providerItemValue>
        <providerItemValue id="useSSL"><value>false</value>
        </providerItemValue>
        <providerItemValue id="usePaging"><value>true</value>
        </providerItemValue>
        <providerItemValue id="useAnonymousBind"><value>false</value>
        </providerItemValue>
        <providerItemValue id="domainUser"><value>mydomain\userID</value>
        </providerItemValue>
      </updateProviderConfiguration>
    </soapenv:Body>
  </soapenv:Envelope>

```



```

<providerItemValue id="domainUserPassword"><value>.iCA-b1ca,Soda,Cracker</value>
</providerItemValue>
<providerItemValue id="UContextBindg"><value>>true</value>
</providerItemValue>
<providerItemValue id="UPasswordAttribute"><value>userPassword</value>
</providerItemValue>
<providerItemValue id="UPasswordDigest"><value>SHA</value>
</providerItemValue>
<providerItemValue id="UbaseDN"><value>ou=users,ou=myorg,dc=mycompany,dc=com</value>
</providerItemValue>
<providerItemValue id="UObjectFilterExpr"><value>(objectCategory=user)</value>
</providerItemValue>
<providerItemValue id="USearchFilterExpr"><value>(sAMAccountName=$ID)</value>
</providerItemValue>
<providerItemValue id="UUniqueNameAttribute"><value>sAMAccountName</value>
</providerItemValue>
<providerItemValue id="UsearchGroupMembershipAttribute"><value>memberOf</value>
</providerItemValue>
<providerItemValue id="GbaseDN"><value>ou=groups,ou=myorg,dc=mycompany,dc=com</value>
</providerItemValue>
<providerItemValue id="GObjectFilterExpr"><value>(objectCategory=group)</value>
</providerItemValue>
<providerItemValue id="GObjectSearchExpr"><value>(sAMAccountName=$ID)</value>
</providerItemValue>
<providerItemValue id="GUniqueNameAttribute"><value>sAMAccountName</value>
</providerItemValue>
<providerItemValue id="GMembershipAttribute"><value>member</value>
</providerItemValue>
<providerItemValue id="GRefreshPeriod"><value>350000</value>
</providerItemValue>
<providerItemValue id="domain"><value>mydomain</value>
</providerItemValue>
<providerItemValue id="allowedUsers"><value>>true</value>
</providerItemValue>
<providerItemValue id="extendedGroups"><value>>true</value>
</providerItemValue>
<providerItemValue id="newName"><value>MyAD</value>
</providerItemValue>
</providerConfigurationUpdate>
</updateProviderConfiguration>
</soapenv:Body>
</soapenv:Envelope>

```

SOAP response example

The server responds to a `updateConfiguration` 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>
    <updateProviderConfigurationResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:status xmlns:ns1="http://xml.spss.com/security"></ns1:status>
    </updateProviderConfigurationResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

The updatePrincipal operation

Updates the password for an existing user, or modifies associations between existing users, groups, and roles in the system.

Input fields

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

Table 32. Fields for `updatePrincipal`.

Field	Type/Valid Values	Description
<code>modifiedPrincipal</code>	<code>modifiedPrincipal</code>	Updated information for a principal identified by its ID. This information may include a new password, or a new set of associations with other principals.

Return information

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

Table 33. Return Value.

Type	Description
string	An internal identifier for a principal.

Java example

Updating a principal involves defining a new password or assigning new relationships to other principals. This information is contained within a `ModifiedPrincipal` object.

The following sample uses `setPrincipalID` to define the principal to be updated. The `setUserPassword` method specifies the new password for the principal. Finally, the `setAssociatedPrincipalID` method defines the other principals to be associated. The `updatePrincipal` operation updates the system according to the updated information in the `ModifiedPrincipal` object.

```
ModifiedPrincipal mp = new ModifiedPrincipal();
mp.setPrincipalID("//uNative//bbrevert");
mp.setUserPassword("qw12as34");
mp.setAssociatedPrincipalID("//gNative//$$security/everyoneGroup");
stub.updatePrincipal(mp);
```

SOAP request example

Client invocation of the `updatePrincipal` 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>
    <updatePrincipal xmlns="http://xml.spss.com/security/remote">
      <modifiedPrincipal principalID="//uNative//bbrevert" userPassword="qw12as34"
        xmlns="http://xml.spss.com/security">
        <associatedPrincipalID>//gNative//$$security/everyoneGroup</associatedPrincipalID>
      </modifiedPrincipal>
    </updatePrincipal>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updatePrincipal` 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>
    <updatePrincipalResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:principalID xmlns:ns1="http://xml.spss.com/security"></ns1:principalID>
    </updatePrincipalResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateRoleDefinition operation

Updates the list of actions associated with an existing role. See the topic “Roles” on page 10 for more information.

Input fields

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

Table 34. Fields for updateRoleDefinition.

Field	Type/Valid Values	Description
<code>modifiedRoleDefinition</code>	<code>modifiedRoleDefinition</code>	Updated action associations for a role, identified by its principal ID.

Return information

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

Table 35. Return Value.

Type	Description
string	General purpose response. Just a string.

Java example

Updating a role requires two pieces of information:

- The role being updated
- The new actions to associate with the role

This information is contained within a `ModifiedRoleDefinition` object.

The following sample creates an array of strings corresponding to action identifiers. The `setActionID` method assigns these identifiers to the `ModifiedRoleDefinition` object. The `setPrincipalID` method defines the principal to update. Use the `updateRoleDefinition` operation to apply the changes to the system.

```
String[] actionArray = {
  "configuration/Editor",
  "security/config",
  "security/roleDefinition",
  "prms/jobs",
  "configuration/MimeManager"
};
ModifiedRoleDefinition mrd = new ModifiedRoleDefinition();
mrd.setActionID(actionArray);
mrd.setPrincipalID("//rNative//nrole");
stub.updateRoleDefinition(mrd);
```

SOAP request example

Client invocation of the updateRoleDefinition 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>
    <updateRoleDefinition xmlns="http://xml.spss.com/security/remote">
      <modifiedRoleDefinition xmlns="http://xml.spss.com/security">
        <actionID>configuration/Editor</actionID>
        <actionID>security/config</actionID>
        <actionID>security/roleDefinition</actionID>
        <actionID>prms/jobs</actionID>
        <actionID>configuration/MimeManager</actionID>
        <principalID>//rNative//nrole</principalID>
      </modifiedRoleDefinition>
    </updateRoleDefinition>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a updateRoleDefinition 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>
    <updateRoleDefinitionResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:status xmlns:ns1="http://xml.spss.com/security"></ns1:status>
    </updateRoleDefinitionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Chapter 5. JAX-WS clients

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

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

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

Generating a JAX-WS client

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

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

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

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

Packaging a JAX-WS client

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

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

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

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

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

Configuring a JAX-WS client

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

The following example demonstrates obtaining the WSDL from the server:

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

The URL includes the host and port for your server.

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

```
com.spss.scoring.ws.jaxws.ScoringServices service =
    new com.spss.scoring.ws.jaxws.ScoringServices(
        DemoClass.class.getResource("/META-INF/wsd1/scoring.wsd1"),
        new QName("http://xml.spss.com/scoring/wsd1", "ScoringServices"));
```

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

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

Next, access the service endpoint:

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

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

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

SOAPHandler example

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

The following code provides an example of this object.

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

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

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

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

    // prefixes
```

```

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

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

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

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

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

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

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

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

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

    return languageHeader;
}

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

```

```

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

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

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

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

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

    return securityHeader;
}

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

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

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

```

Exercising web services from JAX-WS clients

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

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

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

Chapter 6. Microsoft® .NET Framework-based clients

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

1. Add a Service Reference. See the topic “Adding a service reference” for more information.
2. Configure the web service endpoint. See the topic “Configuring the web service endpoint” on page 46 for more information.
3. Programmatically configure the necessary endpoint behaviors. See the topic “Configuring endpoint behaviors” on page 47 for more information.
4. Exercise the web service. See the topic “Exercising the service” on page 47 for more information.

Adding a service reference

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

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

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

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

Service reference modifications

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

To modify a service reference:

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

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

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

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

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

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

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

Configuring the web service endpoint

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

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

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

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

```

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

```

Configuring endpoint behaviors

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

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

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

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

```

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

```

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

Exercising the service

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

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

```

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

```

The corrected code is as follows:

```

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

```

Single sign-on authentication

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

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

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

Chapter 7. Message header reference

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

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

Security headers

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

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

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

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

Table 36. SOAP header namespaces

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

Security element

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

Table 37. Attributes of wsse:Security

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

UsernameToken element

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

Table 38. Attributes of `wsse:UsernameToken`

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

Table 39. Child elements of `wsse:UsernameToken`

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

BinarySecurityToken and BinarySecuritySSOToken elements

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

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

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

Table 40. Attributes of `wsse:BinarySecurityToken`

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

Table 40. Attributes of `wsse:BinarySecurityToken` (continued)

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

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

The client-accept-language element

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

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

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

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

HTTP headers

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

Table 41. HTTP headers

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

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

Notices

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

Privacy policy considerations

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

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

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

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

Trademarks

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

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

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

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

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

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

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

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

Glossary

Index

Special characters

.NET framework 45
.NET proxies 5

A

accessing the Directory Management Service 7
actions 21
Active Directory
 with local override 10
addConfiguration operation 13
app.config files
 WCF clients 46

B

BinarySecuritySSToken element
 in SOAP headers 50
BinarySecurityToken element
 in SOAP headers 50
bindings
 in WSDL files 4
body elements
 in SOAP messages 2

C

calling Directory Management Service operations 7
client-accept-language element
 in SOAP headers 51
configurable directories 9, 13, 22, 24, 33, 35, 36
Content Repository service
 WCF clients 45
Content Repository URI service
 WCF clients 45
Created element
 in SOAP headers 50
createPrincipal operation 15
createRoleDefinition operation 17

D

deletePrincipals operation 18
deleteRoleDefinition operation 19
directories 9
 adding 13
 configurable 9, 13, 22, 24, 33, 35, 36
 creating 13
 deleting 35
 editing 36
 manageable 10, 25
 updating 36
Directory Management Service
 access 7
 configurable directories 9
 directories 9

Directory Management Service
(*continued*)
 manageable directories 10
 overview 7
 principals 10
 roles 11
 stubs 7

G

getActionList operation 21
getConfigurableDirectories operation 22
getConfiguration operation 13, 22, 24, 33
getManageableDirectories operation 25
getManageablePrincipals operation 26
getPrincipalData operation 28
getPrincipalDataEx operation 30
getVersion operation 30
groups 28
 creating 15
 deleting 18
 importing 31
 updating 37

H

header elements
 in SOAP messages 2, 49
 SOAP security elements 49
Holder classes
 in JAX-WS 5
HTTP 2
HTTP headers
 for SOAP messages 51
HTTPS 2

I

importPrincipals operation 31

J

Java clients 41, 42, 44
Java proxies 5
JAX-WS 5, 41, 42, 44

L

List collections
 in JAX-WS 5
local override
 for Active Directory 10
local user repository 10

M

manageable directories 10, 25

MessageBodyMemberAttribute
 for WCF clients 47
messages
 in WSDL files 4

N

namespaces
 for SOAP security elements 49
nativestore.xsd 31
Nonce element
 in SOAP headers 50

O

overview 7

P

Password element
 in SOAP headers 50
PevServices service
 WCF clients 45
port types
 in WSDL files 4
principals 10, 28
 creating 15
 deleting 18
 importing 31
 manageable 26
 updating 37
Process Management service
 WCF clients 45
protocols
 in web services 2
proxies 5
 .NET 5
 Java 5
putConfiguration operation 22, 24, 33

R

removeConfiguration operation 35
roles 28
 actions for 21
 creating 15, 17
 deleting 18, 19
 importing 31
 updating 37, 39

S

Scoring service
 WCF clients 45
Security element
 in SOAP headers 49
services
 in WSDL files 5

- single sign-on
 - for WCF clients 48
 - WCF clients 45
- SOAP 2
- SOAPHandler 42
- SSO
 - See* single sign-on
- stubs
 - Directory Management Service 7

T

- types
 - in WSDL files 3

U

- updateConfiguration operation 13, 36
- updatePrincipal operation 37
- updateRoleDefinition operation 39
- Username element
 - in SOAP headers 50
- UsernameToken element
 - in SOAP headers 50
- users 28
 - creating 15
 - deleting 18
 - importing 31
 - updating 37

V

- Visual Studio 45

W

- WCF clients 45, 47, 48
 - endpoint behaviors 47
 - endpoint configuration 46
 - limitations 45
 - service reference 45
 - single sign-on 45
- web services
 - introduction to web services 1
 - protocol stack 2
 - system architecture 1
 - what are web services? 1
- web.config files
 - WCF clients 46
- Windows Communication Foundation 45
- WSDL files 2, 3
 - bindings 4
 - messages 4
 - port types 4
 - services 5
 - types 3
- wSDL.exe 5
- wSDL2java 5
- wsimport 5, 41

X

- XmlElementAttribute
 - for WCF clients 47



Printed in USA