

**IBM SPSS Collaboration and  
Deployment Services 5 Single  
Sign-On Services Developer's Guide**



*Note:* Before using this information and the product it supports, read the general information under Notices on p. 58.

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

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

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

Licensed Materials - Property of IBM

**© Copyright IBM Corporation 2000, 2012.**

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

---

# Preface

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

## **About IBM Business Analytics**

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

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

## **Technical support**

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

---

# Contents

<b>1</b>	<b><i>Introduction to web services</i></b>	<b>1</b>
	What are web services? . . . . .	1
	Web service system architecture . . . . .	1
	Web service protocol stack . . . . .	2
	Simple Object Access Protocol . . . . .	3
	Web Service Description Language . . . . .	3
	Proxies . . . . .	6
<b>2</b>	<b><i>Single sign-on services overview</i></b>	<b>8</b>
	SSO Authentication Service overview. . . . .	9
	Workflow. . . . .	9
	Accessing the SSO Authentication Service . . . . .	10
	Calling SSO Authentication Service operations . . . . .	10
	SSO Directory Management Service overview . . . . .	10
	Accessing the SSO Directory Management Service. . . . .	10
	Calling SSO Directory Management Service operations . . . . .	11
<b>3</b>	<b><i>Single sign-on concepts</i></b>	<b>12</b>
	The Kerberos ticket . . . . .	12
	Security token . . . . .	12
	Single sign-on provider. . . . .	12
	Directories . . . . .	13
	Configurable directories . . . . .	14
	Manageable directories. . . . .	14
	Principals . . . . .	15
	Roles . . . . .	15
<b>4</b>	<b><i>Operation reference</i></b>	<b>17</b>
	SSO Authentication Service operations . . . . .	17
	The getSSOProviderConfig operation. . . . .	17
	The getToken operation . . . . .	19
	The isSSOEnabled operation . . . . .	20

SSO Directory Management Service operations . . . . .	22
The createPrincipal operation . . . . .	22
The createRoleDefinition operation . . . . .	24
The deletePrincipals operation . . . . .	26
The deleteRoleDefinition operation . . . . .	27
The getActionList operation . . . . .	29
The getConfigurableDirectories operation . . . . .	31
The getManageableDirectories operation . . . . .	33
The getManageablePrincipals operation . . . . .	35
The getPrincipalData operation . . . . .	37
The getSSOConfiguration operation . . . . .	39
The getVersion operation . . . . .	42
The importPrincipals operation . . . . .	43
The putSSOConfiguration operation . . . . .	46
The updatePrincipal operation . . . . .	48
The updateRoleDefinition operation . . . . .	50

## ***Appendices***

### ***A Microsoft® .NET Framework-based clients 53***

Adding a service reference. . . . .	53
Service reference modifications . . . . .	54
Configuring the web service endpoint. . . . .	55
Configuring endpoint behaviors . . . . .	56
Exercising the service. . . . .	56

### ***B Notices 58***

### ***Index 61***



# Introduction to web services

## What are web services?

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

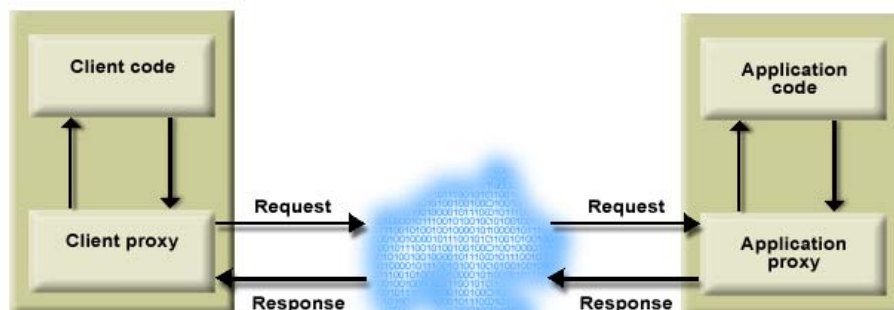
Advantages to implementing application functionality as web services include the following:

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

## Web service system architecture

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

Figure 1-1  
*Web service architecture*



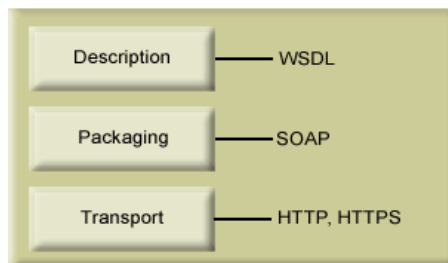
The client code supplies input to an operation offered by a proxy class. The proxy class generates a request containing a standardized representation of the input and sends it across the network to the application. A proxy class on the server receives the request and unmarshals the contents into objects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

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

## Web service protocol stack

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

Figure 1-2  
*Web service protocol stack*



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

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

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



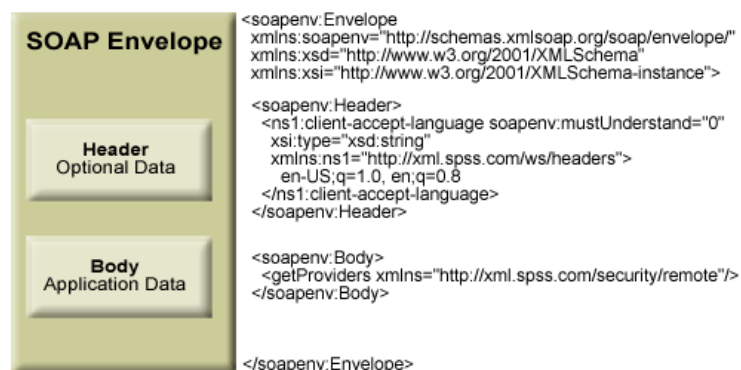
## Simple Object Access Protocol

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

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

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

Figure 1-3  
An example SOAP packet



## Web Service Description Language

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

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

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

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

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

## Types

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

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

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

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

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

## Messages

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

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

The *getProvidersResponse* message contains a single part, corresponding to the *getProvidersResponse* element defined in the types section of the WSDL file. Similarly, the *getProvidersRequest* message also contains a single part, as defined by the *getProviders* element in the types section. [For more information, see the topic Types on p. 4.](#)

## Port types

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

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

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

## Bindings

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

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

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

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

## Services

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

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

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

*http://pes\_server:8080/security-ws/services/ProviderInformation*

## Proxies

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

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

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

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

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

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

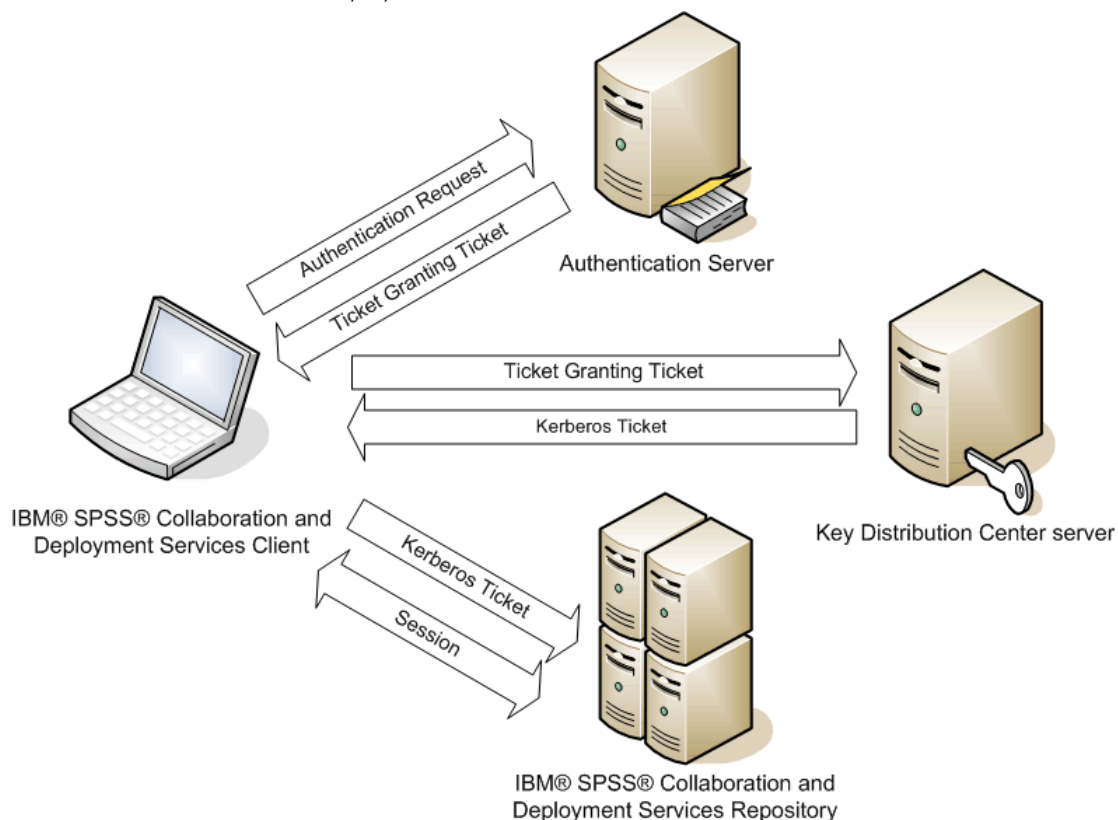
A variety of other tools exist for other programming languages. For details, consult the documentation for those tools. In each case, the tool creates native programming constructs that permit leveraging a web service regardless of the service implementation language.

# Single sign-on services overview

Single sign-on (SSO) is a method of access control that enables a user to log in once and gain access to the resources of multiple software systems without being prompted to log in again. IBM® SPSS® Collaboration and Deployment Services provides single sign-on capability by initially authenticating users through an external directory service based on the **Kerberos** security protocol, and subsequently using the credentials in all IBM SPSS Collaboration and Deployment Services applications (for example, IBM® SPSS® Collaboration and Deployment Services Deployment Manager, IBM® SPSS® Collaboration and Deployment Services Deployment Portal, or a portal server) without additional authentication.

*Note:* Single sign-on is not allowed for browser-based IBM® SPSS® Collaboration and Deployment Services Deployment Manager.

Figure 2-1  
IBM SPSS Collaboration and Deployment Services SSO architecture



For example, when IBM SPSS Collaboration and Deployment Services is used in conjunction with Windows Active directory, you must configure the **Kerberos Key Distribution Center (KDC)** service to enable single sign-on. The service will supply session tickets and temporary

session keys to users and computers within an Active Directory domain. The KDC must run on each domain controller as part of Active Directory Domain Services (AD DS). When single sign-on is enabled, IBM SPSS Collaboration and Deployment Services applications log into a Kerberos domain and use Kerberos tokens for web services authentication. If single sign-on is enabled, it is strongly recommended that SSL communication be configured for the repository.

Single sign-on functionality for IBM SPSS Collaboration and Deployment Services clients is enabled by the following web services:

- **SSO Authentication Service.** Enables single sign-on access.
- **SSO Directory Management Service.** Enables management and configuration of IBM SPSS Collaboration and Deployment Services single sign-on.

Proxy generation tools typically access the WSDL file for a service using the service endpoint. By appending *?wsdl* to the endpoint, you can view a generated WSDL. However, the WSDL file for this service imports the type definitions from a schema. That schema itself also references elements from other schemas. This nesting of schemas causes the WSDL created using the service endpoint to be incomplete, causing difficulties for proxy generation tools. To avoid these problems, use the separate WSDL and schema files provided by IBM Corp. for proxy generation instead of appending *?wsdl* to the endpoint. For this service, these files are:

- *security.wsdl*
- *security-remote.xsd*
- *security.xsd*

Java implementations of single sign-on clients for IBM SPSS Collaboration and Deployment Services can be based on the **Java Authentication and Authorization Service (JAAS)** or any other specification compatible with Kerberos.

## ***SSO Authentication Service overview***

The SSO Authentication Service provides methods for users of client applications to connect to a single sign-on-enabled IBM® SPSS® Collaboration and Deployment Services server by supplying the client single sign-on provider information and distributing Kerberos tokens.

### ***Workflow***

The initial exchange between a client and the server would be as follows:

- Determine if single sign-on is enabled
- Retrieve single sign-on provider configuration
- Use the **JGSS** API to obtain a **ticket-granting ticket** from the single sign-on provider
- Passing in the Kerberos ticket, obtain a security token. The token is used in subsequent calls to web services that require credential passing in non-SSO environments (for example, Authentication Service and Capability Information Service).

## **Accessing the SSO Authentication Service**

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

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

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

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

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

## **Calling SSO Authentication 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/SSOAuthentication";  
URL url = new URL("http", "cads_server", 80, context);  
SSOAuthenticationService service = new SSOAuthenticationServiceLocator();  
stub = service.getSSOAuthentication(url);
```

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

```
stub.isSSOEnabled();
```

## **SSO Directory Management Service overview**

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

- Configure the single sign-on provider
- 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 SSO Directory Management Service**

To access the functionality offered by the SSO 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>/security-ws/services/SSODirectoryManagement
```



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

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

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

### ***Calling SSO 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/SSODirectoryManagement";  
URL url = new URL("http", "cads_server", 80, context);  
SSODirectoryManagementService service = new SSODirectoryManagementServiceLocator();  
stub = service.getSSODirectoryManagement(url);
```

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

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

# *Single sign-on concepts*

## *The Kerberos ticket*

The client and server do not initially share an encryption key. Whenever a client authenticates itself to a new verifier it relies on the authentication server to generate a new encryption key and distribute it securely to both parties. This new encryption key is called a session key and the Kerberos ticket is used to distribute it to the verifier. The Kerberos ticket is a certificate issued by an authentication server, encrypted using the server key. Among other information, the ticket contains the random session key that will be used for authentication of the principal to the verifier, the name of the principal to whom the session key was issued, and an expiration time after which the session key is no longer valid. The ticket is not sent directly to the verifier, but is instead sent to the client who forwards it to the verifier as part of the application request. Because the ticket is encrypted in the server key, known only by the authentication server and intended verifier, it is not possible for the client to modify the ticket without detection.

A key distribution center (KDC) distributes Kerberos tickets to authenticated users. A KDC issues two types of tickets, as follows:

- A master ticket, also known as the ticket granting ticket (TGT)
- A service ticket

A KDC first issues a TGT to a client. The client can then request several service tickets against his or her TGT.

## *Security token*

Web services security provides a general-purpose mechanism to associate security tokens with messages for single message authentication. A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on. A security token is embedded in the SOAP message within the SOAP header. The security token within the SOAP header is propagated from the message sender to the intended message receiver.

## *Single sign-on provider*

IBM® SPSS® Collaboration and Deployment Services sign-on requires a configured Kerberos Key Distribution Server, referred to as a **single sign-on provider**. Configuration information for a Kerberos-based single sign-on provider must include the following information:

- **Enable.** Enables or disables the use of single sign-on provider.
- **Security Provider.** A configured external security providers, such as Windows Active Directory. Local security provider cannot be selected.

- **Kerberos Key Distribution Center Host Address.** Fully qualified name of the Kerberos Domain controller host. For Windows Active Directory, this is the name of the host where Microsoft Active Directory Services are installed.
- **Kerberos Realm.** The Kerberos realm. For Active Directory, this is the domain name.
- **Host.** The name of the IBM® SPSS® Collaboration and Deployment Services Repository host. For example, repositoryhost.mycompany.com.
- **Kerberos Service Principal Name.** The user name for the Kerberos Service Principal.
- **Kerberos Service Principal Password.** The password of the user Kerberos Service Principal.
- **Kerberos Key Table URL.** The URL of the keytab file for Kerberos principals authentication.
- **JAAS Configuration File.** The path of JAAS (Java Authentication and Authorization Service) configuration file on the IBM SPSS Collaboration and Deployment Services host file system. If specified, it overrides the default JAAS configuration. Depending on the application server, this may be necessary to configure the JRE to support SSO.

## Directories

A security provider is responsible for verifying the credentials supplied by a user against a particular user directory. IBM® SPSS® Collaboration and Deployment Services includes an internal directory for authentication, but an existing enterprise user directory can also be used. 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.
- **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.
- **IBM i.** IBM i user profiles directory can be used to authenticate IBM SPSS Collaboration and Deployment Services users. This provider can be enabled or disabled. If IBM i security provider is used with single sign-on-enabled IBM SPSS Collaboration and Deployment

Services installation, EIM (Enterprise Identity Management) must be enabled. Additionally, */QIBM/UserData/Java400/ext/eim.jar* must be copied into the library directory of the IBM SPSS Collaboration and Deployment Services application server if the application server is running on a non-IBM i host.

- **JDE Application User.** If you use IBM® ShowCase®, the ShowCase adapter installation installs this security provider into your IBM SPSS Collaboration and Deployment Services server. This security provider may be configured to allow JD Edwards (JDE) application users to log in and use the IBM SPSS Collaboration and Deployment Services environment. For instructions, see the *ShowCase Administrator's Guide*.

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

# Operation reference

Single sign-on functionality in IBM® SPSS® Collaboration and Deployment Services is enabled by the operations of the following web services described in this chapter:

- SSO Authentication Service
- SSO Directory Management Service

## SSO Authentication Service operations

This section describes the operations enabled by the SSO Authentication Service.

### The *getSSOProviderConfig* operation

Retrieves the configuration parameters for a specified single sign-on security provider.

#### Input fields

The following table lists the input fields for the *getSSOProviderConfig* operation.

Table 4-1  
*Fields for getSSOProviderConfig*

Field	Type/Valid Values	Description
uuid	string	Unique identifier of Provider

#### Return information

The following table identifies the information returned by the *getSSOProviderConfig* operation.

Table 4-2  
*Return Value*

Type	Description
property[]	

#### Java example

The following example function calls *getSSOProviderConfig* from the stub for the service, returning an array of information about single sign-on provider.

```
Property[] properties = stub.getSSOProviderConfig(""); //$NON-NLS-1$

for (int i = 0; i < properties.length; i++)
{
    c_log.warn("name : " + properties[i].getName());
    c_log.warn("value : " + properties[i].getValue());
}
```

```

    }

    return properties;
}

```

### **SOAP request example**

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

```

### **SOAP response example**

The server responds to a `getSSOProviderConfig` 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>
    <getSSOProviderConfigResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:property name="SERVICE_PRINCIPAL_NAME"
        value="HTTP/paswcads4.myCompany.com@myCompany.COM"
        xmlns:ns1="http://xml.spss.com/security"/>
      <ns2:property name="REALM" value="MYCOMPANY.COM"
        xmlns:ns2="http://xml.spss.com/security"/>
      <ns3:property name="SERVER_ADDRESS" value="paswcads4.myCompany.com"
        xmlns:ns3="http://xml.spss.com/security"/>
      <ns4:property name="KDC_ADDRESS" value="kdc_host"
        xmlns:ns4="http://xml.spss.com/security"/>
      <ns5:property name="SECURITY_PROVIDER" value="AD"
        xmlns:ns5="http://xml.spss.com/security"/>
      <ns6:property name="KERBEROS_OID" value="1.2.840.113554.1.2.2"
        xmlns:ns6="http://xml.spss.com/security"/>
      <ns7:property name="PROVIDER_ID" value="ssoKerberos"
        xmlns:ns7="http://xml.spss.com/security"/>
      <ns8:property name="NTLM_CLIENT_DISABLED" value="NTLM_CLIENT_DISABLED"

```



```

    xmlns:ns8="http://xml.spss.com/security"/>
  </getSSOProviderConfigResponse>
</soapenv:Body>
</soapenv:Envelope>

```

## The `getToken` operation

Retrieves the single sign-on authentication token.

### Input fields

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

Table 4-3  
Fields for `getToken`

Field	Type/Valid Values	Description
<code>inputByteArray</code>	<code>byte[]</code>	Input byte array.

### Return information

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

Table 4-4  
Return Value

Type	Description
<code>byte[]</code>	Output byte array containing service ticket

### Java example

The following example function calls `getToken` from the stub for the service, returning the token as a byte array.

```

Oid krb5Oid = new Oid("1.2.840.113554.1.2.2");
GSSManager manager = GSSManager.getInstance();
GSSName serverName = manager.createName(this.clientName, null);
GSSContext context = manager.createContext(serverName, krb5Oid, null, GSSContext.DEFAULT_LIFETIME);
context.requestMutualAuth(true);
context.requestCredDeleg(true);
byte [] token = new byte[0];
token = context.initSecContext(token, 0, token.length);

byte [] retToken = stub.getToken(token);

return retToken;

```

**SOAP request example**

Client invocation of the `getToken` 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>
    <getToken xmlns="http://xml.spss.com/security/remote">
      <inputByteArray>96</inputByteArray>
      <inputByteArray>-126</inputByteArray>
      <inputByteArray>11</inputByteArray>
      <!--The rest of the byte array data -->
    </getToken>
  </soapenv:Body>
</soapenv:Envelope>
```

**SOAP response example**

The server responds to a `getToken` 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>
    <getTokenResponse xmlns="http://xml.spss.com/security/remote">
      <outputByteArray>-84</outputByteArray>
      <outputByteArray>-19</outputByteArray>
      <outputByteArray>0</outputByteArray>
      <outputByteArray>5</outputByteArray>
      <!--The rest of byte array data -->
    </getTokenResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

**The *isSSOEnabled* operation**

Retrieves a Boolean value indicating whether single sign-on is enabled.

**Return information**

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

Table 4-5  
Return Value

Type	Description
boolean	Indicates if Single Sign On is enabled.

### Java example

The following example function calls `isSSOEnabled` from the stub for the service, returning a Boolean value.

```
// Web service call to determine if SSO is enabled.
boolean isSsoEnabled = stub.isSSOEnabled();

c_logger.warn("sso enabled = " + isSsoEnabled); //$NON-NLS-1$

return isSsoEnabled;
}
```

### SOAP request example

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

### SOAP response example

The server responds to a `isSSOEnabled` 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>
    <isSSOEnabledResponse xmlns="http://xml.spss.com/security/remote">
      <uuid>true</uuid>
    </isSSOEnabledResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

## SSO Directory Management Service operations

This section describes the operations enabled by the SSO Directory Management Service.

### 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 4-6  
Fields for `createPrincipal`

Field	Type/Valid Values	Description
<code>newPrincipal</code>	<code>newPrincipal</code>	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 4-7  
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("bbrevere");
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="bbrevere" 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//bbrevere</ns1:principalID>
    </createPrincipalResponse>
  </soapenv:Body>

```

```
</soapenv:Envelope>
```

## The createRoleDefinition operation

Creates a new named role in the system.

### Input fields

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

Table 4-8  
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 4-9  
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 4-10  
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 4-11  
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 *prinipalID*.

```
PrincipalIDList principalIDList = new PrincipalIDList();
principalIDList.addPrincipalID(prinipalID);
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//bbrever</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 4-12  
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 4-13  
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 4-14

Fields for `getActionList`

Field	Type/Valid Values	Description
<code>actionListRequest</code>	string	

### Return information

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

Table 4-15

Return Value

Type	Description
<code>actionList</code>	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>
      </actionList>
    </getActionListResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

```

<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 4-16  
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="devdapOpenLDAP"/>
      </configurableProviders>
    </getConfigurableDirectoriesResponse>
  </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 4-17  
Return Value

Type	Description
manageableProviders	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

### **Input fields**

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

Table 4-18  
*Fields for getManageablePrincipals*

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

### **Return information**

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

Table 4-19  
*Return Value*

Type	Description
principalList	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 `getPrincipallInfo` 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.getPrincipallInfoCount();
PrincipalList retList = new PrincipalList();
PrincipallInfo pi = null;
for (int i = 0; i < count; i++) {
    pi = pList.getPrincipallInfo(i);
    if (pi.getIsRole()) {
        retList.addPrincipallInfo(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>
```

**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/$$$security/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 4-20  
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 4-21  
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.

**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-b1ca,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 getSSOConfiguration operation**

Retrieves configuration parameters for the single sign-on providers.

### **Input fields**

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

Table 4-22  
Fields for *getSSOProviderConfiguration*

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

### Return information

The following table identifies the information returned by the *getSSOConfiguration* operation.

Table 4-23  
Return Value

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

### Java example

The following example function calls *getSSOConfiguration* from the stub for the service, returning an array of SSO configuration parameters for the provider.

```
SSOProviderConfiguration SSOProviderConfig = stub.getSSOConfiguration(id);
return SSOproviderConfig;
```

### SOAP request example

Client invocation of the *getSSOConfiguration* 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//adminUser</wsse:Username>
<wsse:Password wsse:Type=
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">
secret
</wsse:Password>
<wsse:Nonce>OV7BEGvXgCjjxmcL+sL9TA==</wsse:Nonce>
<wsu:Created
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
```

```

2009-04-03T16:20:42Z
</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>
  <getSSOProviderConfiguration xmlns="http://xml.spss.com/security/remote">
    <ns2:SSOProviderID xmlns:ns2="http://xml.spss.com/security">ssoKerberos</ns2:SSOProviderID>
  </getSSOProviderConfiguration>
</soapenv:Body>
</soapenv:Envelope>

```

### **SOAP response example**

The server responds to a `getSSOConfiguration` 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>
    <getSSOProviderConfigurationResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:SSOProviderConfiguration enabled="true" canDisable="true" name="Kerberos SSO Provider"
        id="ssoKerberos" xmlns:ns1="http://xml.spss.com/security">
        <ns1:SSOProviderConfigItem type="text" name="KDC Host Address" id="kdcAddress">
          <ns1:description>Kerberos Key Distribution Center machine address, for example:
            kdc.mycompany.com</ns1:description>
          <ns1:value>kdc.mycompany.com</ns1:value></ns1:SSOProviderConfigItem>
          <ns1:SSOProviderConfigItem type="text" name="Kerberos Realm" id="realm">
            <ns1:description>Kerberos Realm, for example: MYCOMPANY.COM</ns1:description>
            <ns1:value>MYCOMPANY.COM</ns1:value></ns1:SSOProviderConfigItem>
          <ns1:SSOProviderConfigItem type="text" name="Host Address" id="hostAddress">
            <ns1:description>PASW Deployment and Collaboration Services Server machine address, for example:
              paswserver.mycompany.com</ns1:description>
            <ns1:value>paswserver.mycompany.com</ns1:value></ns1:SSOProviderConfigItem>
          <ns1:SSOProviderConfigItem type="text" name="Kerberos Service Principal" id="spn">
            <ns1:description>Kerberos Service Principal Name, for example:
              HTTP/paswserver.mycompany.com@MYCOMPANY.COM</ns1:description>
            <ns1:value>HTTP/paswserver.mycompany.com@MYCOMPANY.COM</ns1:value>
          </ns1:SSOProviderConfigItem>
          <ns1:SSOProviderConfigItem type="password" name="Kerberos Service Principal Password"
            id="spnPassword">
            <ns1:description>Kerberos Service Principal Password</ns1:description>
            <ns1:value>.iCA-b1ca,Soda,Cracker</ns1:value></ns1:SSOProviderConfigItem>
          <ns1:SSOProviderConfigItem type="text" name="Kerberos Key Table URL" id="keytabURL">
            <ns1:description>Kerberos Key Table URL, for example: FILE:c:/keytab/krb5.keytab</ns1:description>

```

```

    <ns1:value>FILE:C:/keytab/krb5.keytab</ns1:value></ns1:SSOProviderConfigItem>
<ns1:SSOProviderConfigItem type="text" name="JAAS Configuration File" id="jaasConfigURL">
  <ns1:description>JAAS configuration file that will be used instead. Consult Administrator's Guide
    to determine format and content of this file.</ns1:description>
  <ns1:value>#USE_SUPPLIED#</ns1:value></ns1:SSOProviderConfigItem>
<ns1:SSOProviderConfigItem type="text" name="Security Provider" id="securityProvider">
  <ns1:description>Security Provider bound to Kerberos provider, for example:
  Active Directory</ns1:description>
  <ns1:value>AD</ns1:value>
</ns1:SSOProviderConfigItem>
</ns1:SSOProviderConfiguration>
</getSSOProviderConfigurationResponse>
</soapenv:Body>
</soapenv:Envelope>

```

## The *getVersion* operation

Returns the version number of the service.

### **Return information**

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

Table 4-24  
Return Value

Type	Description
string	The version of the web service.

### **Java example**

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

```

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

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

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

DirectoryManagement directoryManagement = wsConnections.getDirectoryManagement();

```



```

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

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

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

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

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

```

### **SOAP request example**

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

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getVersion xmlns="http://xml.spss.com/security/remote"/>
  </soapenv:Body>
</soapenv:Envelope>

```

### **SOAP response example**

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

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getVersionResponse xmlns="http://xml.spss.com/security/remote">
      <version>4.20.000</version>
    </getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

## **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 4-25  
*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 4-26  
*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>
      </importPrincipalsStatus>
    </importPrincipalsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

```

</statusItem>
<statusItem name="Obsolete Groups">
  <value>1</value>
</statusItem>
<statusItem name="Obsolete Users">
  <value>2</value>
</statusItem>
</importPrincipalsStatus>
</importPrincipalsResponse>
</soapenv:Body>
</soapenv:Envelope>

```

## The putSSOConfiguration operation

Saves the changes to single sign-on provider configuration parameters.

### Input fields

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

Table 4-27  
Fields for putSSOProviderConfiguration

Field	Type/Valid Values	Description
SSOProviderConfigurationUpdate	SSOProviderConfigurationUpdate	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 putSSOConfiguration operation.

Table 4-28  
Return Value

Type	Description
string	General purpose response. Just a string.

### Java example

The following example function calls putSSOConfiguration from the stub for the service, saving SSO configuration parameters for the provider.

```

SSOProviderConfigurationUpdate updatedConfig = new SSOProviderConfigurationUpdate();
updatedConfig.setSSOProviderItemValue(newItems);
stub.putSSOConfiguration(updatedConfig);

```

**SOAP request example**

Client invocation of the putSSOConfiguration 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/admin</wsse:Username>
        <wsse:Password
          wsse:
            Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">
            spss
          </wsse:Password>
            <wsse:Nonce>OV7BEGvXgCjjxmcl+sL9TA==</wsse:Nonce>
            <wsu:Created
              xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
              2009-04-03T16:20:42Z
            </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>
        <putSSOProviderConfiguration xmlns="http://xml.spss.com/security/remote">
          <ns2:SSOProviderConfigurationUpdate ID="ssoKerberos" enabled="true"
            xmlns:ns2="http://xml.spss.com/security">
            <ns2:SSOProviderItemValue id="kdcAddress">
              <ns2:value>kdc.mycompany.com</ns2:value>
            </ns2:SSOProviderItemValue>
            <ns2:SSOProviderItemValue id="spnPassword">
              <ns2:value>iCA-b1ca,Soda,Cracker</ns2:value>
            </ns2:SSOProviderItemValue>
            <ns2:SSOProviderItemValue id="keytabURL">
              <ns2:value>FILE:C:/keytab/krb5.keytab</ns2:value>
            </ns2:SSOProviderItemValue>
            <ns2:SSOProviderItemValue id="hostAddress">
              <ns2:value>paswserver.mycompany.com</ns2:value>
            </ns2:SSOProviderItemValue>
            <ns2:SSOProviderItemValue id="jaasConfigURL">
              <ns2:value>#USE_SUPPLIED#</ns2:value>
            </ns2:SSOProviderItemValue>
          </ns2:SSOProviderConfigurationUpdate>
        </putSSOProviderConfiguration>
      </soapenv:Body>
    </soapenv:Envelope>
```

```

</ns2:SSOProviderItemValue>
<ns2:SSOProviderItemValue id="spn">
  <ns2:value>HTTP/paswserver.mycompany.com@MYCOMPANY.COM</ns2:value>
</ns2:SSOProviderItemValue>
<ns2:SSOProviderItemValue id="realm">
  <ns2:value>MYCOMPANY.COM</ns2:value>
</ns2:SSOProviderItemValue>
<ns2:SSOProviderItemValue id="securityProvider">
  <ns2:value>AD</ns2:value>
</ns2:SSOProviderItemValue>
</ns2:SSOProviderConfigurationUpdate>
</putSSOProviderConfiguration>
</soapenv:Body>
</soapenv:Envelope>

```

### **SOAP response example**

The server responds to a putSSOConfiguration 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>
    <putSSOProviderConfigurationResponse xmlns="http://xml.spss.com/security/remote">
      <ns1:status xmlns:ns1="http://xml.spss.com/security"></ns1:status>
    </putSSOProviderConfigurationResponse>
  </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 4-29  
Fields for updatePrincipal

Field	Type/Valid Values	Description
modifiedPrincipal	modifiedPrincipal	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 4-30  
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//bbrever");
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//bbrever" userPassword="qw12as34"
        xmlns="http://xml.spss.com/security">
```

```

    <associatedPrincipalID>//gNative//SSsecurity/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.

### **Input fields**

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

Table 4-31  
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 4-32  
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>
      </modifiedRoleDefinition>
    </updateRoleDefinition>
  </soapenv:Body>
</soapenv:Envelope>
```

```
<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>
```

# **Microsoft® .NET Framework-based clients**

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

1. Add a Service Reference. [For more information, see the topic Adding a service reference on p. 53.](#)
2. Configure the web service endpoint. [For more information, see the topic Configuring the web service endpoint on p. 55.](#)
3. Programmatically configure the necessary endpoint behaviors. [For more information, see the topic Configuring endpoint behaviors on p. 56.](#)
4. Exercise the web service. [For more information, see the topic Exercising the service on p. 56.](#)

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

## **Adding a service reference**

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

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

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

## Service reference modifications

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

To modify a service reference:

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

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

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

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

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

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

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

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

## Configuring the web service endpoint

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

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

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

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

## Configuring endpoint behaviors

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

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

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

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

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

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

## Exercising the service

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

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

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

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

The corrected code is as follows:

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

# Notices

This information was developed for products and services offered worldwide.

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

### **Trademarks**

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

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

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

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

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

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

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

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



---

# Index

- accessing the SSO Authentication Service, 10
- accessing the SSO Directory Management Service, 10
- actions, 29
  - roles, 15
- Active Directory, 8, 13–14
  - with local override, 13, 15
- app.config files
  - WCF clients, 55
- authentication, 8
  
- bindings
  - in WSDL files, 5
- body elements
  - in SOAP messages, 3
  
- calling SSO Authentication Service operations, 10
- calling SSO Directory Management Service operations, 11
- configurable directories, 14, 31
- Content Repository service
  - WCF clients, 54
- Content Repository URI service
  - WCF clients, 54
- createPrincipal operation, 22
- createRoleDefinition operation, 24
  
- deletePrincipals operation, 26
- deleteRoleDefinition operation, 27
- directories
  - configurable, 14, 31
  - manageable, 14, 33
- domains
  - for security providers, 14
  
- EIM, 13
- eim.jar, 13
- Enterprise Identity Management, 13
  
  
- getActionList operation, 29
- getConfigurableDirectories operation, 31
- getConfiguration operation, 31
- getManageableDirectories operation, 33
- getManageablePrincipals operation, 35
- getPrincipalData operation, 37
- getSSOConfiguration operation, 39
- getSSOProviderConfig operation, 17
- getToken operation, 19
- getVersion operation, 42
- groups, 15, 37
  - creating, 22
  - deleting, 26
  - importing, 43
  - updating, 48
  
- header elements
  - in SOAP messages, 3
- Holder classes
  - in JAX-WS, 6
- HTTP, 2
- HTTPS, 2
  
- IBM i user repository, 13
- importPrincipals operation, 43
- isSSOEnabled operation, 20
  
  
- Java proxies, 6
- JAX-WS, 6
- JD Edwards (JDE), 14
- JDE Application User security provider, 14
  
  
- Kerberos
  - domain, 8
  - Key Distribution Center, 8
  - Service Ticket, 8
- Kerberos ticket, 12
  
  
- LDAP, 13
- legal notices, 58
- List collections
  - in JAX-WS, 6
- local override
  - for Active Directory, 13, 15
- local user repository, 13, 15
  
  
- manageable directories, 14, 33
- MessageBodyMemberAttribute
  - for WCF clients, 56
- messages
  - in WSDL files, 5
  
  
- native provider, 13
- nativestore.xsd, 43
- .NET framework, 53
- .NET proxies, 7
  
  
- OpenLDAP, 13–14
  - operation reference, 17
    - SSO Authentication Service, 17
    - SSO Directory Management Service, 22
  
  
- PevServices service
  - WCF clients, 54
- port types
  - in WSDL files, 5

- principals, 37
  - creating, 22
  - deleting, 26
  - importing, 43
  - manageable, 35
  - updating, 48
- Process Management service
  - WCF clients, 54
- protocols
  - in web services, 2
- providers, 13
  - domains, 14
  - keys, 14
  - names, 14
- proxies, 6
  - Java, 6
  - .NET, 7
- putConfiguration operation, 31
- putSSOConfiguration operation, 46
  
- roles, 15, 37
  - actions for, 29
  - creating, 22, 24
  - deleting, 26–27
  - importing, 43
  - updating, 48, 50
  
- Scoring service
  - WCF clients, 54
- security providers, 13
  - IBM i user repository, 13
- security token, 12
- service endpoints
  - SSO Authentication Service, 10
  - SSO Directory Management Service, 10
- services
  - in WSDL files, 6
- single sign-on, 8, 13, 20
  - configurable directories, 14
  - directories, 13
  - Kerberos ticket, 12
  - manageable directories, 14
  - operation reference, 17
  - provider, 12
  - roles, 15
  - security token, 12
  - WCF clients, 53
- single sign-on provider, 12
- SOAP, 2–3
- SSO, 13
- SSO Authentication Service, 8
  - accessing, 10
  - calling operations, 10
  - service endpoint, 10
  - stubs, 10
- SSO Authentication Service operations, 17
- SSO Directory Management Service, 8
  - accessing, 10
  - calling operations, 11
  - service endpoint, 10
  - stubs, 11
- SSO Directory Management Service operations, 22
- SSO provider configuration, 17, 39, 46
- SSO Provider Information Service, 8
- SSO token, 19
- stubs
  - SSO Authentication Service, 10
  - SSO Directory Management Service, 11
  
- trademarks, 59
- types
  - in WSDL files, 4
  
- updatePrincipal operation, 48
- updateRoleDefinition operation, 50
- users, 15, 37
  - creating, 22
  - deleting, 26
  - importing, 43
  - updating, 48
  
- Visual Studio, 53
  
- WCF clients, 53, 56
  - endpoint behaviors, 56
  - endpoint configuration, 55
  - limitations, 53
  - service reference, 53–54
  - single sign-on, 53
- web services
  - introduction to web services, 1
  - protocol stack, 2
  - system architecture, 1
  - what are web services?, 1
- web.config files
  - WCF clients, 55
- Windows Communication Foundation, 53
- WSDL files, 2–3
  - accessing, 10–11
  - bindings, 5
  - messages, 5
  - port types, 5
  - services, 6
  - types, 4
- wSDL.exe, 7
- wSDL2java, 6
- wSimport, 6
  
- XmlElementAttribute
  - for WCF clients, 56