IBM SPSS Collaboration and Deployment Services
Version 6 Release 0

Coordinator of Processes Service Developer's Guide

Note
Before using this information and the product it supports, read the information in "Notices" on page 51.

Product Information

This edition applies to version 6, release 0, modification 0 of IBM SPSS Collaboration and Deployment Services and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Introduction to web services

What are web services?

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

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

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

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

Web service system architecture

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

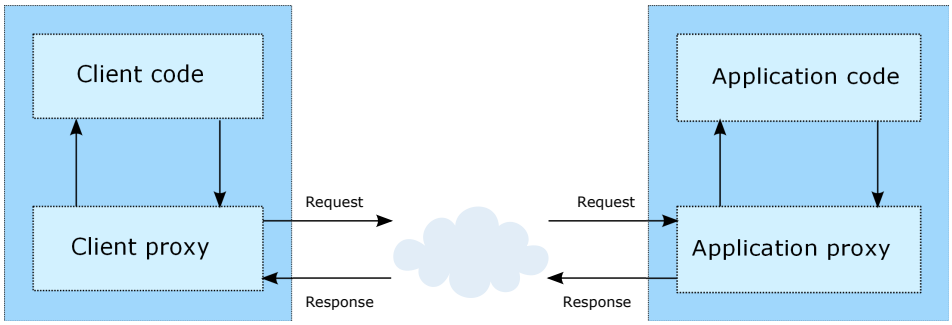

Figure 1. Web service architecture

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

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

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

Web service protocol stack

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

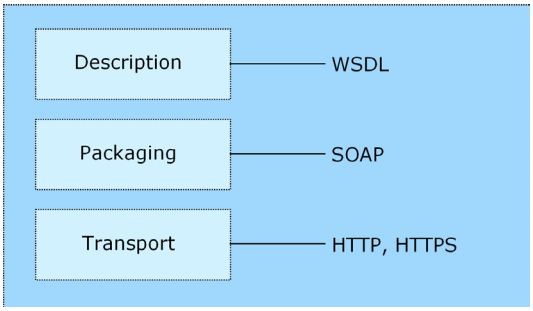

Figure 2. Web service protocol stack

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

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

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

Simple Object Access Protocol

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

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

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

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

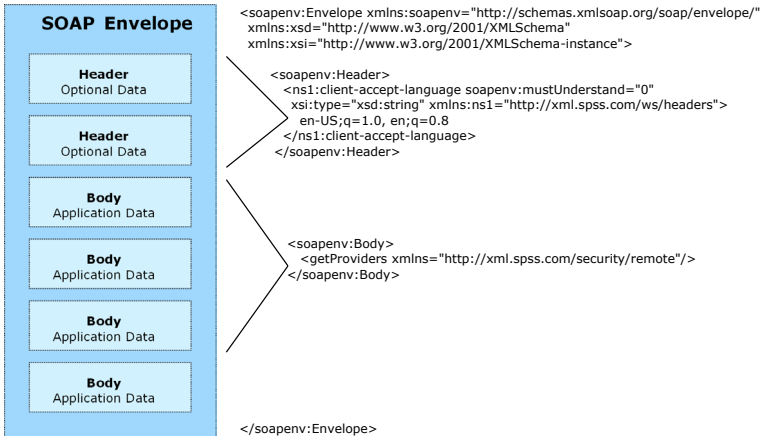

Figure 3. An example SOAP packet

Web Service Description Language

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

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

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

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

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

Types

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

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

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

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

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

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

Messages

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

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

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

Port types

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

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

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

Bindings

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

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

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

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

Services

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

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

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

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

Proxies

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

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

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

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

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

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

Chapter 2. Coordinator of Processes Service overview

The IBM® SPSS® Collaboration and Deployment Services architecture involves a variety of clients communicating with a number of servers. For example, IBM SPSS Collaboration and Deployment Services Repository integrates with execution servers to handle analytical processing tasks defined in jobs. In addition, some clients for those servers, such as IBM SPSS Statistics client or IBM SPSS Modeler client, are also IBM SPSS Collaboration and Deployment Services clients; users can store and retrieve files using the IBM SPSS Collaboration and Deployment Services Repository directly within the client application. After retrieving a file, the client connects to an analytical server to process the content and generate results.

The Coordinator of Processes (COP) provides server management capabilities designed to optimize client-server communication and processing. Services to be managed, such as IBM SPSS Statistics server or IBM SPSS Modeler server, register with the COP upon starting and periodically send updated status messages. Services can also store any necessary configuration files in the IBM SPSS Collaboration and Deployment Services Repository and retrieve them when initializing.

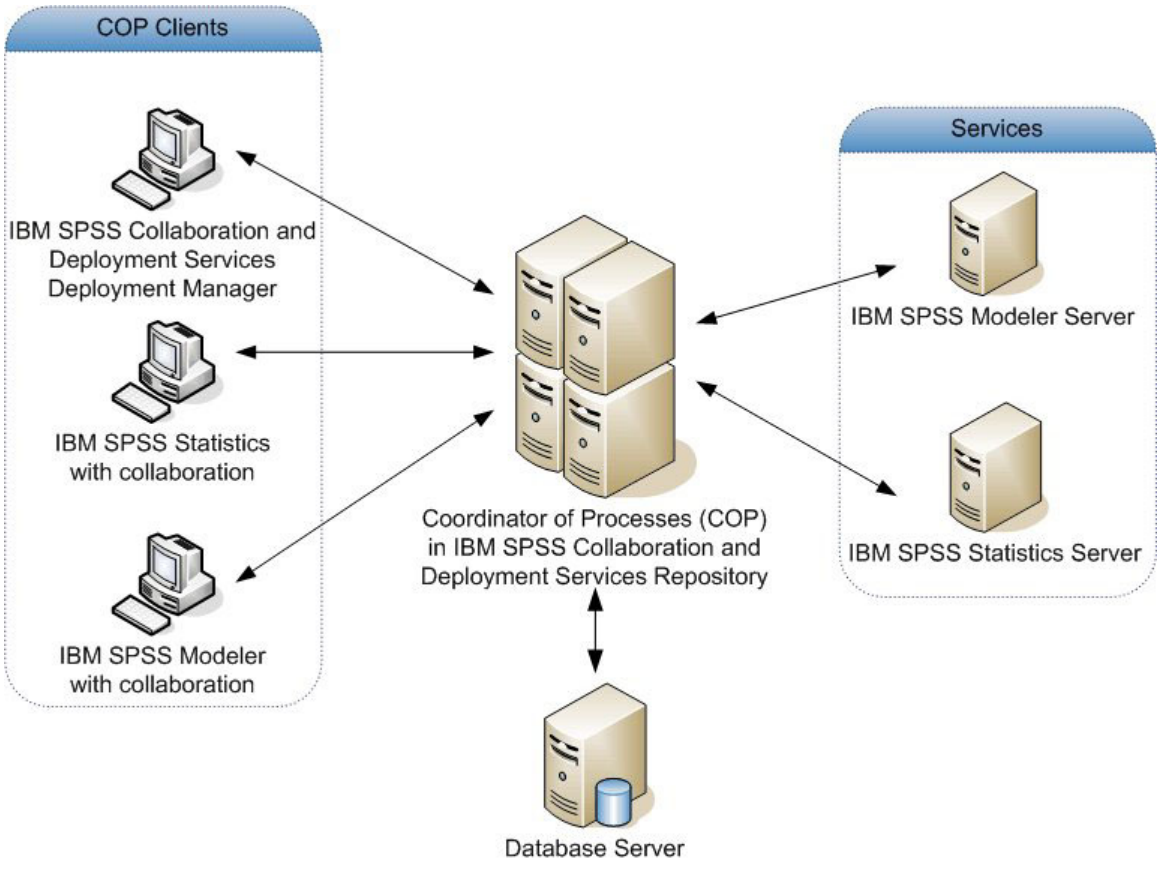

Figure 4. COP Architecture

Client applications use the COP to query for a list of available services to which to connect. Furthermore, services can be grouped into server clusters using the COP. When a client connects to a server cluster, the COP uses a load balancing algorithm to determine the optimal server in the cluster to be used for that client's requests.

The load balancing algorithm for routing server requests uses a weighted least-connection algorithm based on server scores and server loads. When a new connection to a cluster is requested, the system determines a score for each running server in the cluster by using the following formula:

$W_i*C_i/(N_i+1)$

The value of W_i is the weight associated with server *i*. The C_i value is the number of CPUs for server *i*. The value of N_i is the number of current and pending connections for server *i*.

Using the average server load, the system classifies each server as either *available* or *busy*. The new connection is assigned to the *available* server having the highest score. If there are no *available* servers, the connection is assigned to the *busy* server having the highest score. If multiple servers have the same score, the connection is assigned to the one that has the smallest server load.

If two servers in a cluster have the same number of CPUs and server load, the ratio of the number of connections for each will depend entirely on the server weights. For example, if server A has a weight that is twice the weight of server B, server A will handle twice the number of connections. Conversely, if two servers have the same weight and server load, the ratio of the number of connections for each will depend entirely on the number of CPUs for the servers. If server C has eight CPUs and server D has two, server C will handle four times the number of connections.

Notice that the server scores are based on both the number of current and pending connections. If many job steps initiate connections to a server cluster simultaneously, a server in the cluster may not be able to report new connections as current before another connection request is attempted. By including the pending connection count, the scores accurately reflect the impending server load allowing the algorithm to optimize the distribution of requests across all servers in the cluster. A configuration setting defines the time interval during which a connection is classified as pending. For information on modifying this value, consult the administrator documentation.

The Coordinator of Processes Service provides remote interaction with the COP server. Servers use the web service to store configuration files and status information in the IBM SPSS Collaboration and Deployment Services Repository. Clients can retrieve a list of available servers and server clusters for subsequent connections.

Accessing the Coordinator of Processes Service

To access the functionality offered by the Coordinator of Processes Service, create a client application using the proxy classes generated by your preferred web service tool. The service includes three port types, *Configuration*, *Routing*, and *Status*, with the following endpoints:

```
http://<host-name>:<port-number>/<context-root>/spsscop-ws/services/Configuration
http://<host-name>:<port-number>/<context-root>/spsscop-ws/services/Routing
http://<host-name>:<port-number>/<context-root>/spsscop-ws/services/Status
```

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

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

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

```
http://cads_server:80/spsscop-ws/services/Configuration?wsdl
http://cads_server:80/spsscop-ws/services/Routing?wsdl
http://cads_server:80/spsscop-ws/services/Status?wsdl
```

Note: Although there are multiple endpoints for this service, each endpoint returns the same WSDL file.

Calling Coordinator of Processes 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 = "/spsscop-ws/services/Status";
URL url = new URL("http", "cads_server", 80, context);
SPSSCOP service = new SPSSCOPLocator();
stub = service.getStatus(url);
```

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

```
stub.registerService(server);
```

Chapter 3. Coordinator of Processes concepts

Services

A service is a remote resource that processes input into output and communicates with the COP. Typically, a service registers itself with the COP when starting up and automatically sends status information at regular intervals while running. Once registered, a service is referred to as a **managed server**. Types of services available in IBM SPSS Collaboration and Deployment Services include IBM SPSS Statistics server, IBM SPSS Modeler server; IBM SPSS Collaboration and Deployment Services Repository server, STATISTICSB Service, SAS Service, and generic service.

Communication from a service to the COP includes a variety of information. For identification purposes, the service reports an identifier, type, name, and description. The COP uses the identifier internally to reference a particular service. The name and description provide alternative human-readable text useful for identifying a service in client applications, such as in a server administration tool. In addition, status messages include state information indicating whether the service is running, paused, stopped, or in an unknown state. Finally, the information may include a set of property values useful for monitoring and administration of the service. The actual content depends on the service type and may include any of the properties in the table.

Table 1. Service Properties.

Name	Description
`noConns`	Number of current client connections to the service.
`version`	Version number of the service.
`regtimeout`	Time (in minutes) before the service sends the next registration message.
`regtimestamp`	Timestamp associated with the last registration message.
`upsince`	Timestamp indicating when the service last started.
`hostName`	Host name or IP address for the service.
`portNumber`	Host port number on which the service runs.
`domain`	Domain in which the service is running.
`userauth`	User authentication scheme used by the service. Valid values include `win32`, `pem`, and `unix`.
`ssl`	Whether or not the service uses the Secure Sockets Layer protocol for client communications. A value of `true` indicates SSL is used.
`os`	Operating system of the machine on which the service runs.
`noCPUCores`	Number of CPU cores for the machine on which the service runs.
`weight`	Weight of the service.
`systemLoadAverage`	The number of pending and current connections, averaged over time.

The Coordinator of Processes Service includes operations for registering, retrieving, and updating services.

Server clusters

Managed servers can be grouped into **server clusters** to allow load balancing. A server cluster is characterized by the following:

- Internal identifier
- Name
- Description
- List of managed servers
- Load balancing algorithm

When client applications connect to a server cluster, the COP determines which managed server in the cluster is best suited to handle processing requests at that time. The algorithm determining which server reacts to a request depends on several criteria, including the server weights and current processing loads. See the topic Chapter 2, “Coordinator of Processes Service overview,” on page 7 for more information.

The Coordinator of Processes Service includes operations for creating, retrieving, updating, and deleting server clusters.

Configuration files

Some services use configuration files to define settings controlling their behavior. For example, IBM SPSS Statistics servers maintain configuration information in the file *spssd.conf* and IBM SPSS Modeler servers use *options.cfg*. These files can be stored in the repository for retrieval by a service when it starts up. Properties available for stored configuration files include the following:

- Name of the file
- Internal repository identifier used to reference the file
- Timestamp indicating when the file was last updated
- Optional description of the file

The Coordinator of Processes Service includes operations for storing and retrieving configuration files, using the base 64 binary format for encoding the file content.

Chapter 4. Operation reference

Configuration port type

The Configuration port type includes operations used for storing and retrieving service configuration files using the IBM SPSS Collaboration and Deployment Services Repository.

Operation reference

The getServiceConfigFiles operation

Returns the configuration file(s) for a specified service. When starting, a service uses this operation to request its configuration files. To retrieve the default configuration files for its server type, the service should specify an identification number of *-1*.

Input fields

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

Table 2. Fields for getServiceConfigFiles.

Field	Type/Valid Values	Description
configFilesRequest	configFilesRequest	Request for Configuration files.

Return information

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

Table 3. Return Value.

Type	Description
configFileArray	Configuration files root element

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. Return Value.

Type	Description
string	The version of the web service.

Java example

To access the version number of the service, call the `getVersion` operation from the service stub.

```
System.out.println("Service Version =  " + stub.getVersion());
```

SOAP request example

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

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getVersion xmlns="http://xml.spss.com/cop/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/cop/remote">
      <version xmlns="">4.2.0</version>
    </getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The storeServiceConfigFiles operation

Stores configuration file(s) for a service in the IBM SPSS Collaboration and Deployment Services Repository.

Input fields

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

Table 5. Fields for storeServiceConfigFiles.

Field	Type/Valid Values	Description
configFileArray	configFileArray	Configuration files root element

Return information

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

Table 6. Return Value.

Type	Description
response	Generic response

Routing port type

The Routing port type includes operations used for managing services and server clusters.

Operation reference

The createServerCluster operation

Creates a cluster containing managed servers in the repository.

Input fields

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

Table 7. Fields for createServerCluster.

Field	Type/Valid Values	Description
serverCluster	serverCluster	This represents a single ServerCluster contains references to Managed Servers

Return information

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

Table 8. Return Value.

Type	Description
clusterId	cluster Id

Java example

To create a server cluster:

1. Create a `ServerCluster` object.
2. Supply the `setName` and `setDescription` methods with strings to define the cluster name and description.
3. Create a string array containing the identifiers for the services to be included in the cluster. Service identifiers can be obtained from the information returned by the `getServices` operation.
4. Supply the `setServiceIdentifiers` method with the identifier array.
5. Provide the `createServerCluster` operation with the cluster object.

The following sample creates a server cluster containing two services.

```
ServerCluster servCluster = new ServerCluster();
servCluster.setDescription("Test cluster");
String[] serviceIdentifier = new String[2];
serviceIdentifier[0] = "0a0a4a35eabc54a600000111eb0df4588055";
serviceIdentifier[1] = "0a0a4a351de26ecf000001112cba0440807d";
servCluster.setServiceIdentifiers(serviceIdentifier);
ClusterId id = stub.createServerCluster(servCluster);
```

SOAP request example

Client invocation of the `createServerCluster` 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>
    <createServerCluster xmlns="http://xml.spss.com/cop/remote">
      <ns1:serverCluster ns1:description="Test cluster" xmlns:ns1="http://xml.spss.com/cop">
        <ns1:identifier>-1</ns1:identifier>
```

```
        <ns1:serviceIdentifiers>0a0a4a35eabc54a600000111eb0df4588055</ns1:serviceIdentifiers>
        <ns1:serviceIdentifiers>0a0a4a351de26ecf000001112cba0440807d</ns1:serviceIdentifiers>
      </ns1:serverCluster>
    </createServerCluster>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `createServerCluster` 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>
    <createServerClusterResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:clusterId ns1:id="0a0a4a35e5dbe13100000113ba800f9a80a5"
        xmlns:ns1="http://xml.spss.com/cop"/>
    </createServerClusterResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The deleteServerCluster operation

Deletes the cluster corresponding to the supplied identifier.

Input fields

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

Table 9. Fields for deleteServerCluster.

Field	Type/Valid Values	Description
clusterId	clusterId	cluster Id

Return information

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

Table 10. Return Value.

Type	Description
response	Generic response

Java example

To delete a server cluster:

1. Create a `ClusterId` object.
2. Supply the `setId` method with a string corresponding to the identifier of the server cluster to be deleted.
3. Provide the `deleteServerCluster` operation with the `ClusterId` object.

The following sample deletes a server cluster, sending the response from the operation to the standard output.

```
ClusterId clustid = new ClusterId();
clustid.setId("0a0a4a35e5dbe13100000113ba800f9a80a5");
Response resp = stub.deleteServerCluster(clustid);
System.out.println("Response =  " + resp.getResp());
```

SOAP request example

Client invocation of the `deleteServerCluster` 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>
    <deleteServerCluster xmlns="http://xml.spss.com/cop/remote">
      <ns1:clusterId ns1:id="0a0a4a35e5dbe13100000113ba800f9a80a5"
        xmlns:ns1="http://xml.spss.com/cop"/>
    </deleteServerCluster>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteServerCluster` 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>
    <deleteServerClusterResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:response ns1:resp="0" xmlns:ns1="http://xml.spss.com/cop"/>
    </deleteServerClusterResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getServerCluster operation

Returns the server cluster corresponding to a specified identifier.

Input fields

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

Table 11. Fields for getServerCluster.

Field	Type/Valid Values	Description
clusterId	clusterId	cluster Id

Return information

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

Table 12. Return Value.

Type	Description
serverCluster	This represents a single ServerCluster contains references to Managed Servers

Java example

To retrieve a server cluster:

1. Create a `ClusterId` object.
2. Supply the `setId` method with a string corresponding to the identifier of the server cluster to retrieve.
3. Provide the `getServerCluster` operation with the `ClusterId` object.

The `ServerCluster` object returned by the operation includes the name, description, and algorithm defined for the cluster. This information can be retrieved using the `getName`, `getDescription`, and `getAlgorithm` methods. In addition, the cluster object includes the identifiers for all services contained within the cluster.

The following sample sends the identifiers for all services in a specified server cluster to the standard output.

```
ClusterId clustid = new ClusterId();
clustid.setId("0a0a4a35e5dbe13100000113ba800f9a80ba");
ServerCluster servCluster = stub.getServerCluster(clustid);
String[] serviceIds = servCluster.getServiceIdentifiers();
System.out.println("Identifiers for services in the server cluster");
for (int j = 0; j < serviceIds.length; j++) {
   System.out.println(serviceIds[j]);
}
```

SOAP request example

Client invocation of the `getServerCluster` 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>
    <getServerCluster xmlns="http://xml.spss.com/cop/remote">
      <clusterId xmlns="http://xml.spss.com/cop" xmlns:ns1="http://xml.spss.com/cop"
        ns1:id="0a0a4a35e5dbe13100000113ba800f9a80ba"/>
    </getServerCluster>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getServerCluster` 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>
    <getServerClusterResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:serverCluster ns1:description="Test cluster" xmlns:ns1="http://xml.spss.com/cop">
        <ns1:identifier>0a0a4a35e5dbe13100000113ba800f9a80ba</ns1:identifier>
        <ns1:serviceIdentifiers>0a0a4a351de26ecf000001112cba0440807d</ns1:serviceIdentifiers>
        <ns1:serviceIdentifiers>0a0a4a35eabc54a600000111eb0df4588055</ns1:serviceIdentifiers>
        <ns1:serviceIdentifiers>0a0a4a350cd4c47200000111ffd26a3880eb</ns1:serviceIdentifiers>
        <ns1:algorithm>Test cluster</ns1:algorithm>
      </ns1:serverCluster>
    </getServerClusterResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getServerClusters operation

Returns all server cluster currently in the system.

Input fields

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

Table 13. Fields for getServerClusters.

Field	Type/Valid Values	Description
request	request	Generic request Parameter

Return information

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

Table 14. Return Value.

Type	Description
serverClusters	Cluster List

Java example

To retrieve all server clusters in the system:

1. Create a `Request` object.
2. Provide the `getServerClusters` operation with the request.

The `ServerCluster` objects returned by the operation include the name, description, and algorithm defined for the clusters. This information can be retrieved using the `getName`, `getDescription`, and `getAlgorithm` methods. In addition, the cluster objects include the identifiers for all services contained within the cluster.

The following sample sends the identifiers for all services in server clusters to the standard output.

```
Request req = new Request();
ServerCluster[] servCluster = stub.getServerClusters(req);
for (int i = 0; i < servCluster.length; i++) {
   System.out.println("Identifiers for services in server cluster " +
      servCluster[i].getDescription());
   String[] serviceIds = servCluster[i].getServiceIdentifiers();
   for (int j = 0; j < serviceIds.length; j++) {
      System.out.println(serviceIds[j]);
   }
}
```

SOAP request example

Client invocation of the `getServerClusters` 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>
```

```
    <getServerClusters xmlns="http://xml.spss.com/cop/remote">
      <ns1:request xmlns:ns1="http://xml.spss.com/cop"/>
    </getServerClusters>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getServerClusters` 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>
    <getServerClustersResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:serverClusters xmlns:ns1="http://xml.spss.com/cop">
        <ns1:serverClusters ns1:description="Cluster of SPSS Server on Windows" ns1:name="Cluster1">
          <ns1:identifier>0a0a48442b95127200000113fe6accb78519</ns1:identifier>
          <ns1:serviceIdentifiers>0a0a48442b95127200000113fe6accb78511</ns1:serviceIdentifiers>
          <ns1:serviceIdentifiers>0a0a48442b95127200000113fe6accb78501</ns1:serviceIdentifiers>
          <ns1:algorithm>Cluster of SPSS Server on Windows</ns1:algorithm>
        </ns1:serverClusters>
        <ns1:serverClusters ns1:description="SPSS Cluster" ns1:name="My Cluster">
          <ns1:identifier>0a0a48442b95127200000113fe6accb78592</ns1:identifier>
          <ns1:serviceIdentifiers>0a0a48442b95127200000113fe6accb78511</ns1:serviceIdentifiers>
          <ns1:serviceIdentifiers>0a0a48442b95127200000113fe6accb780d8</ns1:serviceIdentifiers>
          <ns1:serviceIdentifiers>0a0a4844a4660ba500000113f923e5a88027</ns1:serviceIdentifiers>
          <ns1:algorithm>SPSS Cluster</ns1:algorithm>
        </ns1:serverClusters>
      </ns1:serverClusters>
    </getServerClustersResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getService operation

Returns a managed server registered with the COP. The server returned can be limited to one having specific property values by specifying criteria that must be met.

Input fields

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

Table 15. Fields for getService.

Field	Type/Valid Values	Description
serviceRequest	serviceRequest	registered Services request
serviceCriteria	serviceCriteria	registered Services request based on Criteria

Return information

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

Table 16. Return Value.

Type	Description
managedServer	This represents a single Managed Server instance

The getServiceById operation

Returns the managed server corresponding to the supplied identifier.

Input fields

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

Table 17. Fields for getServiceById.

Field	Type/Valid Values	Description
serviceRequest	serviceRequest	registered Services request

Return information

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

Table 18. Return Value.

Type	Description
managedServer	This represents a single Managed Server instance

Java example

To retrieve a specific service using its identifier:

1. Create a `ServiceRequest` object.
2. Supply the `setId` method with a string corresponding to the identifier of the service to retrieve.
3. Provide the `getServiceById` operation with the request object.

The `ManagedServer` object returned by the operation includes the name, type, and state of the cluster. This information can be retrieved using the `getName`, `getServiceType`, and `getState` methods. In addition, the cluster object includes the property values available for the service.

The following sample sends the name, type, and state of a service to the standard output.

```
ServiceRequest req = new ServiceRequest();
req.setId("0a0a48442b95127200000113fe6accb78511");
ManagedServer server = stub.getServiceById(req);
System.out.println("Name: " + server.getName());
System.out.println("Type: " + server.getServiceType().toString());
System.out.println("State: " + server.getState().toString());
```

SOAP request example

Client invocation of the `getServiceById` 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>
    <getServiceById xmlns="http://xml.spss.com/cop/remote">
      <serviceRequest xmlns="http://xml.spss.com/cop" id ="0a0a48442b95127200000113fe6accb78511"/>
    </getServiceById>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getServiceById` 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>
    <getServiceByIdResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:managedServer ns1:description="PASW Statistics Server on Windows 32 bit"
        ns1:serviceType="SPSSServer" ns1:state="running" ns1:name="SPSSServerWin32"
        xmlns:ns1="http://xml.spss.com/cop">
        <ns1:identifier>0a0a48442b95127200000113fe6accb78511</ns1:identifier>
        <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
          <ns1:value>5.0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
          <ns1:value>0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
          <ns1:value>120</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
          <ns1:value>1185471549573</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
          <ns1:value>1185460981000</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
          <ns1:value>10.10.72.68</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
          <ns1:value>3016</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
          <ns1:value>false</ns1:value>
        </ns1:propertyValue>
      </ns1:managedServer>
    </getServiceByIdResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getServiceFromServerCluster operation

Returns a specified service from a cluster.

Input fields

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

Table 19. Fields for getServiceFromServerCluster.

Field	Type/Valid Values	Description
clusterCriteria	clusterCriteria	registered Services request based on Criteria

Return information

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

Table 20. Return Value.

Type	Description
managedServer	This represents a single Managed Server instance

The getServices operation

Returns all managed services registered with the COP. The list of services returned can be limited to those having specific property values by specifying criteria that must be met.

Input fields

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

Table 21. Fields for getServices.

Field	Type/Valid Values	Description
serviceCriteria	serviceCriteria	registered Services request based on Criteria

Return information

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

Table 22. Return Value.

Type	Description
managedServers	This represents array of Managed Server instances.

Java example

To retrieve services:

1. Create an array of `PropertyValue` objects defining the criteria that all returned services must meet.
2. Supply the `setId` method with a string corresponding to the identifier of the service to retrieve.
3. Provide the `getServices` operation with the property value array.

The array of `ManagedServer` objects returned by the operation includes the name, type, and state of each service meeting the defined criteria. This information can be retrieved using the `getName`, `getServiceType`, and `getState` methods. In addition, the cluster object includes the property values available for the services.

The following sample sends the name, type, and state of all services managed by COP to the standard output.

```
PropertyValue serviceCriteria = new PropertyValue();
ManagedServer[] services = stub.getServices(serviceCriteria);
for (int j = 0; j < services.length; j++) {
   System.out.println("Name: " + services[j].getName());
   System.out.println("Type: " + services[j].getServiceType().toString());
   System.out.println("State: " + services[j].getState().toString());
}
```

SOAP request example

Client invocation of the `getServices` 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>
    <getServices xmlns="http://xml.spss.com/cop/remote">
```

```
      <ns1:serviceCriteria xmlns:ns1="http://xml.spss.com/cop"/>
    </getServices>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getServices` 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>
    <getServicesResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:managedServers xmlns:ns1="http://xml.spss.com/cop">
        <ns1:managedServers ns1:description="SPSSServer on Linux" ns1:serviceType="SPSSServer"
          ns1:state="unknown" ns1:name="SPSSServerLinux1">
          <ns1:identifier>0a0a48442b95127200000113fe6accb780d8</ns1:identifier>
          <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
            <ns1:value>5.0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="string" ns1:name="userauth">
            <ns1:value>win32</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
            <ns1:value>0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
            <ns1:value>120</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
            <ns1:value>1185465825787</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
            <ns1:value>0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
            <ns1:value>10.11.40.216</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
            <ns1:value>3353</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
            <ns1:value>false</ns1:value>
          </ns1:propertyValue>
        </ns1:managedServers>
        <ns1:managedServers ns1:description="SPSS Server on Sol 64bit" ns1:serviceType="SPSSServer"
          ns1:state="unknown" ns1:name="SPSSServerSol11">
          <ns1:identifier>0a0a48442b95127200000113fe6accb781a6</ns1:identifier>
          <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
            <ns1:value>5.0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="string" ns1:name="userauth">
            <ns1:value>win32</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
            <ns1:value>0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
            <ns1:value>120</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
            <ns1:value>1185409221463</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
            <ns1:value>0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
            <ns1:value>10.11.10.201</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
            <ns1:value>3353</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
            <ns1:value>false</ns1:value>
          </ns1:propertyValue>
        </ns1:managedServers>
        <ns1:managedServers ns1:description="SPSS Server on Windows 32 bit"
          ns1:serviceType="SPSSServer" ns1:state="running" ns1:name="SPSSServerWin32">
          <ns1:identifier>0a0a48442b95127200000113fe6accb78511</ns1:identifier>
          <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
```

```
          <ns1:value>5.0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
          <ns1:value>0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
          <ns1:value>120</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
          <ns1:value>1185470949583</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
          <ns1:value>1185460981000</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
          <ns1:value>10.10.72.68</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
          <ns1:value>3016</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
          <ns1:value>false</ns1:value>
        </ns1:propertyValue>
      </ns1:managedServers>
      <ns1:managedServers ns1:description="SPSS Server on HP-UX" ns1:serviceType="SPSSServer"
        ns1:state="unknown" ns1:name="SPSSServerHPUX">
        <ns1:identifier>0a0a4844a4660ba500000113f923e5a88027</ns1:identifier>
        <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
          <ns1:value>5.0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
          <ns1:value>0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
          <ns1:value>120</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
          <ns1:value>1185409710410</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
          <ns1:value>0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
          <ns1:value>10.11.10.42</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
          <ns1:value>3353</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
          <ns1:value>false</ns1:value>
        </ns1:propertyValue>
      </ns1:managedServers>
    </ns1:managedServers>
  </getServicesResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

The getServicesByType operation

Returns all managed services of a specified type registered with the COP. The list of services returned can be limited to those having specific property values by specifying criteria that must be met.

Input fields

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

Table 23. Fields for getServicesByType.

Field	Type/Valid Values	Description
serviceRequest	serviceRequest	registered Services request
propertyValue	propertyValue[]	Represents a value of the property.

Return information

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

Table 24. Return Value.

Type	Description
managedServer[]	This represents a single Managed Server instance

Java example

To retrieve types of services:

1. Create a ServiceRequest object.
2. Supply the setType method with the ServiceType value corresponding to the type of service to retrieve.
3. Provide the getServicesByType operation with the request object.

The array of ManagedServer objects returned by the operation includes the name, type, and state for each returned service. This information can be retrieved using the getName, getServiceType, and getState methods. In addition, the array includes the property values available for the service.

The following sample sends the name, type, and state of for all IBM SPSS Statistics servers managed by COP to the standard output.

```
ServiceRequest req = new ServiceRequest();
req.setType(ServiceType.SPSSServer);
ManagedServer[] services = stub.getServicesByType(req);
for (int j = 0; j < services.length; j++) {
   System.out.println("Name: " + services[j].getName());
   System.out.println("Type: " + services[j].getServiceType().toString());
   System.out.println("State: " + services[j].getState().toString());
}
```

SOAP request example

Client invocation of the getServicesByType 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>
    <getServicesByType xmlns="http://xml.spss.com/cop/remote">
      <serviceRequest xmlns="http://xml.spss.com/cop" type="SPSSServer"/>
    </getServicesByType>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a getServicesByType 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>
```

```
<getServicesByTypeResponse xmlns="http://xml.spss.com/cop/remote">
  <ns1:managedServers xmlns:ns1="http://xml.spss.com/cop">
    <ns1:managedServers ns1:description="SPSSServer on Linux" ns1:serviceType="SPSSServer"
      ns1:state="unknown" ns1:name="SPSSServerLinux1">
      <ns1:identifier>0a0a48442b95127200000113fe6accb780d8</ns1:identifier>
      <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
        <ns1:value>5.0</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="string" ns1:name="userauth">
        <ns1:value>win32</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
        <ns1:value>0</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
        <ns1:value>120</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
        <ns1:value>1185465825787</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
        <ns1:value>0</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
        <ns1:value>10.11.40.216</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
        <ns1:value>3353</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
        <ns1:value>false</ns1:value>
      </ns1:propertyValue>
    </ns1:managedServers>
    <ns1:managedServers ns1:description="SPSS Server on Sol 64bit" ns1:serviceType="SPSSServer"
      ns1:state="unknown" ns1:name="SPSSServerSol11">
      <ns1:identifier>0a0a48442b95127200000113fe6accb781a6</ns1:identifier>
      <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
        <ns1:value>5.0</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="string" ns1:name="userauth">
        <ns1:value>win32</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
        <ns1:value>0</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
        <ns1:value>120</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
        <ns1:value>1185409221463</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
        <ns1:value>0</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
        <ns1:value>10.11.10.201</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
        <ns1:value>3353</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
        <ns1:value>false</ns1:value>
      </ns1:propertyValue>
    </ns1:managedServers>
    <ns1:managedServers ns1:description="SPSS Server on Windows 32 bit"
      ns1:serviceType="SPSSServer" ns1:state="running" ns1:name="SPSSServerWin32">
      <ns1:identifier>0a0a48442b95127200000113fe6accb78511</ns1:identifier>
      <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
        <ns1:value>5.0</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
        <ns1:value>0</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
        <ns1:value>120</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
        <ns1:value>1185472149577</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
        <ns1:value>1185460981000</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
        <ns1:value>10.10.72.68</ns1:value>
```

```
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
            <ns1:value>3016</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
            <ns1:value>false</ns1:value>
          </ns1:propertyValue>
        </ns1:managedServers>
        <ns1:managedServers ns1:description="SPSS Server on HP-UX" ns1:serviceType="SPSSServer"
          ns1:state="unknown" ns1:name="SPSSServerHPUX">
          <ns1:identifier>0a0a4844a4660ba500000113f923e5a88027</ns1:identifier>
          <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
            <ns1:value>5.0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
            <ns1:value>0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
            <ns1:value>120</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
            <ns1:value>1185409710410</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
            <ns1:value>0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
            <ns1:value>10.11.10.42</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
            <ns1:value>3353</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
            <ns1:value>false</ns1:value>
          </ns1:propertyValue>
        </ns1:managedServers>
      </ns1:managedServers>
    </getServicesByTypeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getServicesFromServerCluster operation

Returns all services contained within a specified cluster.

Input fields

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

Table 25. Fields for getServicesFromServerCluster.

Field	Type/Valid Values	Description
clusterId	clusterId	cluster Id

Return information

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

Table 26. Return Value.

Type	Description
managedServers	This represents array of Managed Server instances.

Java example

To retrieve the services within a server cluster:

1. Create a `ClusterId` object.
2. Supply the `setId` method with a string corresponding to the identifier of the server cluster to retrieve.
3. Provide the `getServicesFromServerCluster` operation with the `ClusterId` object.

The array of ManagedServer objects returned by the operation includes the name, type, and state for each service in the cluster. This information can be retrieved using the getName, getServiceType, and getState methods. In addition, the properties for each ManagedServer object in the array can be accessed using the getPropertyValue method.

The following sample sends information for all services in a specified server cluster to the standard output.

```
ClusterId clustid = new ClusterId();
clustid.setId("0a0a48442b95127200000113fe6accb78592");
ManagedServer[] services = stub.getServicesFromServerCluster(clustid);
for (int j = 0; j < services.length; j++) {
   System.out.println("Name: " + services[j].getName());
   System.out.println("Type: " + services[j].getServiceType().toString());
   System.out.println("State: " + services[j].getState().toString());
}
```

SOAP request example

Client invocation of the getServicesFromServerCluster 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>
    <getServicesFromServerCluster xmlns="http://xml.spss.com/cop/remote">
      <clusterId xmlns="http://xml.spss.com/cop" xmlns:ns1="http://xml.spss.com/cop"
        ns1:id="0a0a48442b95127200000113fe6accb78592"/>
    </getServicesFromServerCluster>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a getServicesFromServerCluster 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>
    <getServicesFromServerClusterResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:managedServers xmlns:ns1="http://xml.spss.com/cop">
        <ns1:managedServers ns1:description="SPSS Server on Windows 32 bit"
          ns1:serviceType="SPSSServer" ns1:state="running" ns1:name="SPSSServerWin32">
          <ns1:identifier>0a0a48442b95127200000113fe6accb78511</ns1:identifier>
          <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
            <ns1:value>5.0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
            <ns1:value>0</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
            <ns1:value>120</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
            <ns1:value>1185471909587</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
            <ns1:value>1185460981000</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
            <ns1:value>10.10.72.68</ns1:value>
          </ns1:propertyValue>
          <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
```

```
          <ns1:value>3016</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
          <ns1:value>false</ns1:value>
        </ns1:propertyValue>
      </ns1:managedServers>
      <ns1:managedServers ns1:description="SPSSServer on Linux" ns1:serviceType="SPSSServer"
        ns1:state="unknown" ns1:name="SPSSServerLinux1">
        <ns1:identifier>0a0a48442b95127200000113fe6accb780d8</ns1:identifier>
        <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
          <ns1:value>5.0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="userauth">
          <ns1:value>win32</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
          <ns1:value>0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
          <ns1:value>120</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
          <ns1:value>1185465825787</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
          <ns1:value>0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
          <ns1:value>10.11.40.216</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
          <ns1:value>3353</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
          <ns1:value>false</ns1:value>
        </ns1:propertyValue>
      </ns1:managedServers>
      <ns1:managedServers ns1:description="SPSS Server on HP-UX" ns1:serviceType="SPSSServer"
        ns1:state="unknown" ns1:name="SPSSServerHPUX">
        <ns1:identifier>0a0a4844a4660ba500000113f923e5a88027</ns1:identifier>
        <ns1:propertyValue ns1:typeCode="double" ns1:name="weight">
          <ns1:value>5.0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
          <ns1:value>0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
          <ns1:value>120</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
          <ns1:value>1185409710410</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="timestamp" ns1:name="upsince">
          <ns1:value>0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
          <ns1:value>10.11.10.42</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
          <ns1:value>3353</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
          <ns1:value>false</ns1:value>
        </ns1:propertyValue>
      </ns1:managedServers>
    </ns1:managedServers>
  </getServicesFromServerClusterResponse>
 </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 27. Return Value.

Type	Description
string	The version of the web service.

Java example

To access the version number of the service, call the `getVersion` operation from the service stub.

```
System.out.println("Service Version =  " + stub.getVersion());
```

SOAP request example

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

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getVersion xmlns="http://xml.spss.com/cop/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/cop/remote">
      <version xmlns="">4.2.0</version>
    </getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The sanityCheck operation

Verifies that the COP server is running and accessible to the client.

Input fields

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

Table 28. Fields for sanityCheck.

Field	Type/Valid Values	Description
request	request	Generic request Parameter

Return information

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

Table 29. Return Value.

Type	Description
response	Generic response

Java example

To verify communication between a client and the COP server:

1. Create a `Request` object.
2. Provide the `sanityCheck` operation with the request object.

The following sample sends the results of the verification to the standard output.

```
Request req = new Request();
Response resp = stub.sanityCheck(req);
System.out.println(resp.getResp());
```

SOAP request example

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

SOAP response example

The server responds to a `sanityCheck` 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>
    <sanityCheckResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:response ns1:resp="success" xmlns:ns1="http://xml.spss.com/cop"/>
    </sanityCheckResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateServerCluster operation

Updates an existing specified server cluster. For example, this operation can be used to change the name, description, or service composition of a server cluster.

Input fields

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

Table 30. Fields for updateServerCluster.

Field	Type/Valid Values	Description
serverCluster	serverCluster	This represents a single ServerCluster contains references to Managed Servers

Return information

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

Table 31. Return Value.

Type	Description
response	Generic response

Java example

To update a server cluster:

1. Retrieve the `ServerCluster` object for the cluster to be updated. The `getServerClusters` operation can be used to retrieve the cluster identifiers.
2. Modify the server cluster.
3. Provide the `updateServerCluster` operation with the revised cluster object.

The following sample adds a new service to an existing server cluster.

```
ClusterId clustid = new ClusterId();
clustid.setId("0a0a48442b95127200000113fe6accb78592");
ServerCluster servCluster = stub.getServerCluster(clustid);
String[] serviceIds = servCluster.getServiceIdentifiers();
String[] newIds = new String[serviceIds.length+1];
for (int j = 0; j < serviceIds.length; j++) {
   newIds[j] = serviceIds[j];
}
newIds[serviceIds.length] = "0a0a4844a4660ba500000113f923e5a88027";
servCluster.setServiceIdentifiers(newIds);
Response resp = stub.updateServerCluster(servCluster);
```

SOAP request example

Client invocation of the `updateServerCluster` 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>
    <updateServerCluster xmlns="http://xml.spss.com/cop/remote">
      <ns1:serverCluster ns1:description="SPSS Cluster" xmlns:ns1="http://xml.spss.com/cop">
        <ns1:identifier>0a0a48442b95127200000113fe6accb78592</ns1:identifier>
        <ns1:serviceIdentifiers>0a0a48442b95127200000113fe6accb780d8</ns1:serviceIdentifiers>
        <ns1:serviceIdentifiers>0a0a48442b95127200000113fe6accb78511</ns1:serviceIdentifiers>
        <ns1:serviceIdentifiers>0a0a4844a4660ba500000113f923e5a88027</ns1:serviceIdentifiers>
      </ns1:serverCluster>
    </updateServerCluster>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updateServerCluster` 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>
    <updateServerClusterResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:response ns1:resp="0" xmlns:ns1="http://xml.spss.com/cop"/>
    </updateServerClusterResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Status port type

The Status port type includes operations used by clients to keep the COP server aware of the current service property values.

Operation reference

The getVersion operation

Returns the version number of the service.

Return information

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

Table 32. Return Value.

Type	Description
string	The version of the web service.

Java example

To access the version number of the service, call the `getVersion` operation from the service stub.

```
System.out.println("Service Version =  " + stub.getVersion());
```

SOAP request example

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

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getVersion xmlns="http://xml.spss.com/cop/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/cop/remote">
      <version xmlns="">4.2.0</version>
    </getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The registerService operation

Upon starting, a COP-enabled service uses the `registerService` operation to alert the COP server to its availability and provide current values for the service properties. If the COP server has not previously assigned an identification number to the service, the server generates a new identifier.

Input fields

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

Table 33. Fields for registerService.

Field	Type/Valid Values	Description
managedServer	managedServer	This represents a single Managed Server instance

Return information

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

Table 34. Return Value.

Type	Description
statusResponse	Generic response

SOAP request example

Client invocation of the `registerService` 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>
    <registerService xmlns="http://xml.spss.com/cop/remote">
      <ns1:managedServer ns1:description="Tom's Statistics Server"
        ns1:serviceType="SPSSServer" ns1:state="running" ns1:name="Tom"
        xmlns:ns1="http://xml.spss.com/cop">
        <ns1:identifier>0a0a48442b95127200000113fe6accb78609</ns1:identifier>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
          <ns1:value>0</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
          <ns1:value>1185475760924</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
          <ns1:value>120</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="weight">
          <ns1:value>5</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
          <ns1:value>10.10.76.58</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
          <ns1:value>3016</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="upsince">
          <ns1:value>1185475759000</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
          <ns1:value>false</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="userauth">
          <ns1:value>win32</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="version">
          <ns1:value>16.0.0</ns1:value>
```

```
        </ns1:propertyValue>
      </ns1:managedServer>
    </registerService>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `registerService` 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>
    <registerServiceResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:statusResponse ns1:id="0a0a48442b95127200000113fe6accb78609" ns1:name="Tom"
        xmlns:ns1="http://xml.spss.com/cop"/>
    </registerServiceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateService operation

Values for service properties, such as the number of client connections or the time last started, can change from one point in time to another. A COP-enabled service uses the `updateService` operation to periodically send updated status and property values to the COP server.

Input fields

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

Table 35. Fields for updateService.

Field	Type/Valid Values	Description
managedServer	managedServer	This represents a single Managed Server instance

Return information

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

Table 36. Return Value.

Type	Description
statusResponse	Generic response

SOAP request example

Client invocation of the `updateService` 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>
    <updateService xmlns="http://xml.spss.com/cop/remote">
      <ns1:managedServer ns1:description="Tom's Statistics Server"
```

```
        ns1:serviceType="SPSSServer" ns1:state="running" ns1:name="Tom"
        xmlns:ns1="http://xml.spss.com/cop">
        <ns1:identifier>0a0a48442b95127200000113fe6accb78609</ns1:identifier>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="noConns">
          <ns1:value>1</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimestamp">
          <ns1:value>1185475880894</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="regtimeout">
          <ns1:value>120</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="weight">
          <ns1:value>5</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="hostName">
          <ns1:value>10.10.76.58</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="portNumber">
          <ns1:value>3016</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="long" ns1:name="upsince">
          <ns1:value>1185475759000</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="boolean" ns1:name="ssl">
          <ns1:value>false</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="userauth">
          <ns1:value>win32</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue ns1:typeCode="string" ns1:name="version">
          <ns1:value>16.0.0</ns1:value>
        </ns1:propertyValue>
      </ns1:managedServer>
    </updateService>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updateService` 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>
    <updateServiceResponse xmlns="http://xml.spss.com/cop/remote">
      <ns1:statusResponse ns1:id="0a0a48442b95127200000113fe6accb78609" ns1:name="Tom"
        xmlns:ns1="http://xml.spss.com/cop"/>
    </updateServiceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Chapter 5. JAX-WS clients

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

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

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

Generating a JAX-WS client

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

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

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

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

Packaging a JAX-WS client

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

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

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

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

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

Configuring a JAX-WS client

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

The following example demonstrates obtaining the WSDL from the server:

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

The URL includes the host and port for your server.

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

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

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

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

Next, access the service endpoint:

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

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

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

SOAPHandler example

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

The following code provides an example of this object.

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

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

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

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

    // prefixes
```

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

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

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

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

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

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

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

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

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

    return languageHeader;
}

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

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

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

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

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

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

        return securityHeader;
    }

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

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

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

}
```

Exercising web services from JAX-WS clients

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

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

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

Chapter 6. Microsoft® .NET Framework-based clients

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

1. Add a Service Reference. See the topic "Adding a service reference" for more information.
2. Configure the web service endpoint. See the topic "Configuring the web service endpoint" on page 44 for more information.
3. Programmatically configure the necessary endpoint behaviors. See the topic "Configuring endpoint behaviors" on page 45 for more information.
4. Exercise the web service. See the topic "Exercising the service" on page 45 for more information.

Adding a service reference

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

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

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

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

Service reference modifications

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

To modify a service reference:

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

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

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

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

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

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

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

Configuring the web service endpoint

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

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

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

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

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

Configuring endpoint behaviors

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

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

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

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

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

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

Exercising the service

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

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

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

The corrected code is as follows:

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

Single sign-on authentication

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

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

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

Chapter 7. Message header reference

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

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

Security headers

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

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

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

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

Table 37. SOAP header namespaces

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

Security element

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

Table 38. Attributes of `wsse:Security`

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

UsernameToken element

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

Table 39. Attributes of wsse:UsernameToken

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

Table 40. Child elements of wsse:UsernameToken

Attribute	Description	Example
wsse:Username	The xml value represents the identity of the user.	a_user
wsse:Password	The attribute Type specifies the type of password. PasswordText is currently the only supported type. The xml value can handle plain text passwords and encrypted data.	myPassword [{AES}KrY+KLlOYo4O6545tgGsYQ==]
wsse:Nonce	The xml value represents a cryptographically random nonce encoded as base64 data. This is currently ignored.	RUx1ugQo0o3g0Xyl+sUEsA==
wsu:Created	The xml value represents the creation time as a timestamp conforming to wsu:Timestamp. This is currently ignored.	2013-10-08T02:09:20Z

BinarySecurityToken and BinarySecuritySSOToken elements

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

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

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

Table 41. Attributes of wsse:BinarySecurityToken

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

Table 41. Attributes of wsse:BinarySecurityToken (continued)

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

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

The client-accept-language element

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

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

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

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

HTTP headers

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

Table 42. HTTP headers

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

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

Notices

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

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

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

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

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

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

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

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

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

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

Licensees of this program who 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
ATTN: Licensing
200 W. Madison St.
Chicago, IL; 60606
U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

Trademarks

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

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

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

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

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

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

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

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

Glossary

Index

Special characters

A

B

C

D

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Printed in USA