IBM SPSS Collaboration and Deployment Services 5 Data Services Developer's Guide

Note: Before using this information and the product it supports, read the general information under Notices on p. 40.

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

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

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

Preface

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

About IBM Business Analytics

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

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

Technical support

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

Contents

Appendices

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.

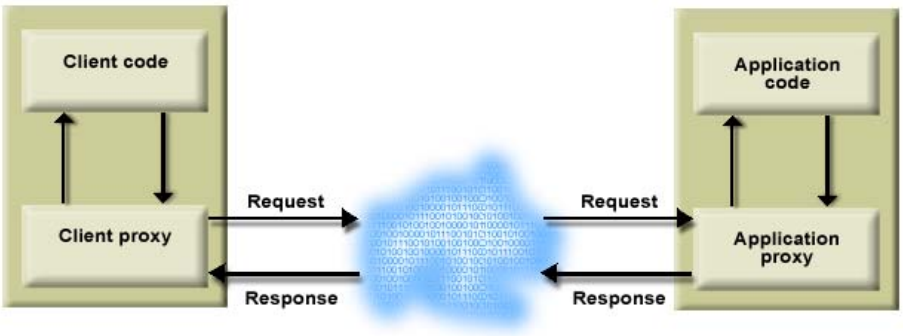
Advantages to implementing application functionality as web services include the following:

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

Figure 1-1
Web service architecture



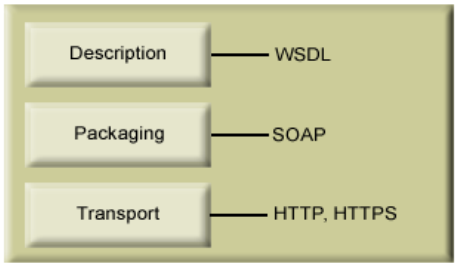
The client code supplies input to an operation offered by a proxy class. The proxy class generates a request containing a standardized representation of the input and sends it across the network to the application. A proxy class on the server receives the request and unmarshals the contents into objects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

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

Web service protocol stack

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

Figure 1-2
Web service protocol stack



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

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

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

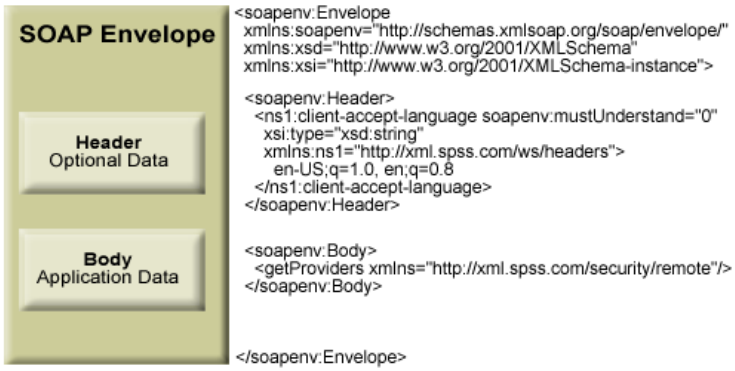
Simple Object Access Protocol

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

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

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

Figure 1-3
An example SOAP packet



Web Service Description Language

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

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

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

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

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

Types

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

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

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

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

Port types

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

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

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

Bindings

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

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

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

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

Services

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

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

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

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

Proxies

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

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

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

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

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

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

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

Chapter
2

Data Services Service overview

The Data Services Service provides functionality used when working with the data sources defined in the IBM® SPSS® Collaboration and Deployment Services Repository for analytic and scoring tasks. In general, the service provides the ability to perform the following tasks:

- Retrieve metadata about the tables available in data sources
- Retrieve information about table columns and links

The Data Services Service is often used in conjunction with the Scoring Service. Use the Data Services Service to access information about the data used for a particular scoring configuration.

Workflow

The actual sequence of web service calls you need when working with data sources will depend on your particular application. However, the input requirements for the Data Services Service operations leads to the following general workflow for a specific data source:

1. Determine the types of tables used in the data source by using the getTableTypes operation.
2. Retrieve the metadata for the data source tables of the desired type or types by using the getTableMetaData operation.
3. Using the metadata for a specific table, access information about the table columns by using the getTableSimpleColumns operation. In addition, you can you use the table metadata to retrieve information about the datasets within the data source by using the getDataSets operation.

Accessing the Data Services Service

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

http://<host-name>:<port-number>/scoring/services/Data

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

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

http://cads_server:80/scoring/services/Data?wsdl

Calling Data Services 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 = "/scoring/services/data";
URL url = new URL("http", "cads_server", 80, context);
dataService service = new dataServiceLocator();
stub = service.getStatus(url);
```

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

```
stub.getTableMetaData(dsURI, credURI, type);
```

Chapter
3

Data Service concepts

Uniform Resource Identifiers

Resources within the IBM® SPSS® Collaboration and Deployment Services Repository are often referenced using a uniform resource identifier. A content repository URI consists of the following items:

- The scheme *spsscr:*
- A hierarchical specification consisting of an authority definition and an optional object path
- An optional query specifying an object identifier
- Optional fragments defining version information

The URI has the following format:

spsscr://[*host*][:*port*]/[*path/filename* [?hierarchyType=*type*] | ?id=*repositoryID*][#l.*label* | #m.*marker*]

The hierarchical portion begins with two slashes, followed by the authority definition. This information identifies the host name and port number for the repository containing the object, followed by a slash. The authority definition may be omitted, in which case the URI indicates a relative location within the repository processing the service request.

spsscr:///[*path/filename* [?hierarchyType=*type*] | ?id=*repositoryID*][#l.*label* | #m.*marker*]

The URI continues with either the full path to the object, including its name, or a question mark and a query term consisting of the key *id*, an equals sign, and the repository resource identifier for the object. This identifier can be obtained from the information returned by the getResource operation of the Content Repository Service.

If the URI specifies an object path, the path may be followed by a query parameter designating the type of hierarchy containing the object. This parameter begins with a question mark, followed by the key *hierarchyType*, an equals sign, and the hierarchy type designator. Valid hierarchy types include *folder*, *topic*, *configuration*, *server*, *credential*, *datasource*, *enterprise*, and *submitted*. If the *hierarchyType* parameter is omitted, the *folder* hierarchy is used by default. The *hierarchyType* parameter is valid only when using the path to identify the object.

Optional version fragments follow the object information. The fragments begin with a hash symbol (#), followed by a single letter indicating whether the fragment is a version label (l) or a version timestamp marker (m). The fragment ends with a period and the actual label or marker for the version. Replace any spaces in the label or marker with escape characters. For example, the URI:

spsscr://myserver:80/marketing/campaign1#m.0:2006-10-08%2012:34:10.223

refers to the version of the *campaign1* job in the *marketing* folder saved at 12:34 on October 8, 2006. A URI that does not include a version fragment references the latest version of the object. For instance, the URI:

```
spsscr://localhost/campaign2
```

refers to the latest version of the job *campaign2*.

Data sources

A data source definition defines the connection information necessary to connect to a data source. The connection properties depend on the data source type. For example, open database connectivity (ODBC) provides a mechanism for client programs to access databases or data sources. An ODBC definition consists of the data source name (DSN). Similarly, Java database connectivity (JDBC) determines how Java applications access databases. A JDBC definition consists of the driver name and URL. The data service API also provides public Java interfaces for implementing custom drivers to access nonstandard data sources. For information on creating custom drivers, see the IBM® SPSS® Collaboration and Deployment Services customization documentation.

Data source definitions allow other components, such as the Enterprise View, to access the data sources used in the system. Data source definitions are typically created using the IBM® SPSS® Collaboration and Deployment Services Deployment Manager and are stored in the IBM® SPSS® Collaboration and Deployment Services Repository. For information on creating data source definitions, see the Deployment Manager documentation. Data sources can be referenced using their IBM SPSS Collaboration and Deployment Services Repository uniform resource identifiers. For more information, see the topic Uniform Resource Identifiers on p. 10.

Credentials

Some IBM® SPSS® Collaboration and Deployment Services Repository resources, such as Enterprise View and Data Provider Definition items, require access to data from physical data sources. In order to connect to these data sources, credential definitions must be defined and stored within the IBM SPSS Collaboration and Deployment Services Repository. Each credential definition contains a user identifier and password. In addition, some credentials require the specification of a security provider against which to validate the credential information. Credentials stored within the repository can be referenced using their uniform resource identifiers. For more information, see the topic Uniform Resource Identifiers on p. 10.

Tables

Information within the databases referenced by the data source definitions is stored in tables having defined characteristics. Each table has a defined type within the architecture, with the list of types varying across database vendors. Commonly occurring types include *TABLE* and *VIEW*.

In addition to the table type, each table has a set of metadata values providing information useful when referencing the table. These metadata properties include the following:

- **Catalog name.** Name of the catalog containing the table.

- **Schema name.** Name of the schema on which the table is based.
- **Table name.** Name of the table within the database.
- **Qualifier separator.** In a fully qualified name, the character used between the catalog and the remainder of the name.
- **Catalog prefix.** A boolean flag indicating whether the catalog is used as a prefix or a suffix in qualified names. If the flag is true, the name format is *catalog.schema.table*. If the flag is false, the format is *schema.table.catalog*. The actual delimiter used for the catalog is defined by the qualifier separator.
- **Catalog use**. A boolean flag indicating whether or not the catalog is used in qualified names. If the flag is false, the name format is *schema.table*. If the flag is true, the name format is determined by the catalog prefix value.
- **Identifier quote character.** The character the database uses to quote user-defined identifiers. For example, a table name containing a space character, such as *My Table*, is invalid in SQL queries unless quoted. For a SQL Server database, the quote character is double quotes so the table would be referenced as *"My Table"*. In contrast, for a mySQL database, the quote character is a backtick so the table would be referenced as *`My Table`*.

The "Example SELECT statements" table displays example SELECT statements for various values of the metadata properties. Each example uses a catalog named *admin*, a schema named *testdb*, and a table named *My Table*.

Table 3-1
Example SELECT statements

Qualifier Separator	Catalog Use	Catalog Prefix	Identifier Quote Character	SELECT Statement
.	True	True	"	select * from admin.testdb."My Table"
@	True	False	"	select * from testdb."My Table"@admin
&	False	False	`	select * from testdb.`My Table`

The Data Services Service includes operations for retrieving the table types and the table metadata.

Chapter
4

Operation reference

The getDataSets operation

Returns information about the datasets for tables contained within a specified data source. Supply one or more table metadata definitions to limit the results to specific tables. For more information, see the topic The getTableMetaData operation on p. 25.

The information returned consists of escaped XML definitions of the columns in the tables, as well as any links involving the tables.

Input fields

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

Table 4-1
Fields for getDataSets

Field	Type/Valid Values	Description
dataSourceURI	string	The datasource URI.
credentialsURI	string	The credentials URI.
tableMeta	tableMeta[]	This element is used to describe the table meta data within the datasource.

Return information

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

Table 4-2
Return Value

Type	Description
dataSetBundle	

Java example

To access the datasets information for a data source, supply the getDataSets operation with strings corresponding to the repository uniform resource identifiers for the following:

- The data source
- Valid credentials for accessing the data source

To limit the datasets information to specific tables, include an array of TableMeta objects describing the tables. The meta objects would typically be selected from the results of the getTableMetaData operation.

The following sample returns the datasets information for the fourth table returned by the getTableMetaData operation.

```
String dsURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080be";
String credURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080c4";
String type = "TABLE";
TableMeta[] meta = stub.getTableMetaData(dsURI, credURI, type);
DataSetBundle bundle = stub.getDataSets(dsURI, credURI, meta[3]);

String[] datasets = bundle.getDataSets();
for (int i = 0; i < datasets.length; i++) {
  System.out.println(datasets[i]);
}

String[] links = bundle.getLinks();
for (int j = 0; j < links.length; j++) {
  System.out.println(links[j]);
}
```

For web service clients based on JAX-WS, replace the arrays in the sample with List collections and update the array processing accordingly. For example:

```
String dsURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080be";
String credURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080c4";
List<String> typeList = Arrays.asList("TABLE", "VIEW");
List<TableMeta> metaList = stub.getTableMetaData(dsURI, credURI, typeList);
DataSetBundle bundle = stub.getDataSets(dsURI, credURI, metaList.get(3));

List<String> datasetsList = bundle.getDataSets();
for (String datasets : datasetsList)
{
  System.out.println(datasets);
}

List<String> linksList = bundle.getLinks();
for (String links : linksList)
{
  System.out.println(links);
}
```

SOAP request example

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

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header>
      <wsse:Security soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
         soapenv:mustUnderstand="0"
         xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <wsse:UsernameToken
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
```

```
        <wsse:Username>Native//admin</wsse:Username>
        <wsse:Password
          wsse:Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
          #PasswordText"
          >pass</wsse:Password>
        <wsse:Nonce>ofOShsZMIgHcdD0o6A8PkQ==</wsse:Nonce>
        <wsu:Created
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          >2009-01-08T20:36:10Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
    <ns1:client-accept-language soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0" xmlns:ns1="http://xml.spss.com/ws/headers">en-US;q=1.0,
      en;q=0.8</ns1:client-accept-language>
  </soapenv:Header>
  <soapenv:Body>
    <getDataSets xmlns="http://xml.spss.com/data/remote">
      <dataSourceURI>spsscr:///?id=0a010a07e123f4280000011f5babe02080be</dataSourceURI>
      <credentialsURI>spsscr:///?id=0a010a07e123f4280000011f5babe02080c4</credentialsURI>
      <tableMeta xmlns="http://xml.spss.com/data">
        <catalogName>cq_ecm_data</catalogName>
        <schemaName>cq</schemaName>
        <tableName>Defect</tableName>
        <qualifierSeparator>.</qualifierSeparator>
        <catalogPrefixBool>true</catalogPrefixBool>
        <useCatalogBool>true</useCatalogBool>
        <identifierQuoteCharacter>&quot;</identifierQuoteCharacter>
      </tableMeta>
    </getDataSets>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a getDataSets 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>
    <getDataSetsResponse xmlns="http://xml.spss.com/data/remote">
      <datasets xmlns="http://xml.spss.com/data">
        <dataSets>&lt;?xml version=&quot;1.0&quot;
          encoding=&quot;UTF-8&quot;?&gt; &lt;dataSet
          xmlns=&quot;http://xml.spss.com/pev&quot;
          name=&quot;Defect&quot;
          qualifiedTableName=&quot;&amp;quot;cq_ecm_data&amp;quot;
          .&amp;quot;cq&amp;quot;.&amp;quot;Defect&amp;quot;&quot;&gt;&lt;column
          name=&quot;ratl_mastership&quot;
```

type="integer"/><column
name="dbid"
type="integer"/><column
name="is_active"
type="integer"/><column
name="id"
type="string"/><column
name="verificationotes"
type="string"/><column
name="reflist"
type="integer"/><column
name="release_commitment_date"
type="timestamp"/><column
name="release_commitment"
type="integer"/><key
name="fk_16779512_1"
isUnique="true"><keyColumn
name="reflist"
type="integer"/></key><key
name="fk_16780379_2"
isUnique="true"><keyColumn
name="deflist"
type="integer"/></key><key
name="fk_16780379_3"
isUnique="true"><keyColumn
name="release_commitment"
type="integer"/></key></dataSet>
</dataSets>
</datasets>
</getDataSetsResponse>
</soapenv:Body>
</soapenv:Envelope>

The getSamples operation

Returns a data sample from a data provider based on specified inputs. For example, the operation can retrieve values for the variables *var1*, *var2*, and *var3* based on the value for the key variable *var0*. The information returned consists of a table structure containing the sample values in the rows.

Input fields

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

Table 4-3
Fields for getSamples

Field	Type/Valid Values	Description
sampleDetails	sampleDetails	This element is used to describe the information needed for getSamples request.

Return information

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

Table 4-4
Return Value

Type	Description
sampleResult	Return type of getSamples() Web Service.

Java example

To retrieve a sample from a data provider:

1. Create a SampleDetails object.
2. Use the setEvURI method to specify the uniform resource identifier for the Enterprise View.
3. Use the setAvURI method to specify the uniform resource identifier for the Application View.
4. Provide the setDpdXML method with a string corresponding to escaped XML for the data provider definition.
5. Use the setTableName method to define the name of the table containing the data to sample.
6. Supply the setResultColumnNames method with an array of strings identifying the result columns.
7. Create a TableType object for the input key values. Use the setName method to define the name for the key. The setColumnName method specifies the names for the columns whose values are being supplied.
8. Create a Value object for the actual input values. Use the setValue method to specify the values.
9. Create a RowValuesType object for the value objects. Use the setValue method to assign the objects to the RowValuesType. Add this object to the TableType object using the setRowValues method.
10. Assign the key values to the details object using the setKeyValues method.
11. Use the setSample method to define the sample type as either *AV* or *DATASET*.
12. Supply the getSamples operation with the details object.

    The following sample returns the *component* values in the *CQdefects* table for a *customerid* value of 19029.00000.

```
SampleDetails details = new SampleDetails();
details.setEvURI("spsscr:///?id=ac140f8000072ffb0000010b290a16b28003#m.4:2009-02-12%2013:10:33.289");
details.setAvURI("spsscr:///?id=0a010a077e08b4a40000011f5cabdad38098#m.2:2009-02-11%2011:37:24.848");
// The XML for the dpd is rather long. Rather than duplicate it here,
// we direct you to the SOAP request for an example.
String dpdXML = "SEE THE SOAP REQUEST FOR AN EXAMPLE OF DPD XML";
details.setDpdXML(dpdXML);
details.setTableName("CQdefects");

String[] resultCols = {
 "customerid",
```

```
 "component"
 };
details.setResultColumnNames(resultCols);

TableType keys = new TableType();
keys.setName("custID");
keys.setColumnName("customerid");
Value val = new Value();
val.setValue("19029.00000");
RowValuesType rowVals = new RowValuesType();
rowVals.setValue(val);
keys.setRowValues(rowVals);
details.setKeyValues(keys);

details.setSample(SampleType.AV);

TableType sampleTable = stub.getSamples(details);

String[] colNames = sampleTable.getColumnName();
RowValuesType[] rowValType = sampleTable.getRowValues();
for (int i = 0; i < rowValType.length; i++) {
  Value[] resultVals = rowValType[i].getValue();
  for (int j = 0; j < resultVals.length; j++) {
    System.out.println(colNames[j] + " = " + resultVals[j].getValue());
  }
}
```

For web service clients based on JAX-WS, replace the arrays in the sample with List collections and update the array processing accordingly. For example:

```
SampleDetails details = new SampleDetails();
details.setEvURI("spsscr:///?id=ac140f8000072ffb0000010b290a16b28003#m.4:2009-02-12%2013:10:33.289");
details.setAvURI("spsscr:///?id=0a010a077e08b4a40000011f5cabdad38098#m.2:2009-02-11%2011:37:24.848");
// The XML for the dpd is rather long. Rather than duplicate it here,
// we direct you to the SOAP request for an example.
String dpdXML = "SEE THE SOAP REQUEST FOR AN EXAMPLE OF DPD XML";
details.setDpdXML(dpdXML);
details.setTableName("CQdefects");

details.getResultColumnNames().add("customerid");
details.getResultColumnNames().add("component");

TableType keys = new TableType();
keys.setName("custID");
keys.setColumnName("customerid");
Value val = new Value();
val.setValue("19029.00000");
RowValuesType rowVals = new RowValuesType();
rowVals.setValue(val);
keys.setRowValues(rowVals);
details.setKeyValues(keys);
```

```
details.setSample(SampleType.AV);

TableType sampleTable = stub.getSamples(details);

List<String> colNamesList = sampleTable.getColumnName();
for (String colNames : colNamesList)
{
  System.out.println(colNames + "\t");
}
System.out.println("\n");
List<RowValuesType> rowValTypeList = sampleTable.getRowValues();
for (RowValuesType rowValType : rowValTypeList)
{
  List<Value> resultValsList = rowValType.getValue();
  for (Value resultVals : resultValsList)
  {
    System.out.println(resultVals.getValue() + "\t");
  }
  System.out.println("\n");
}
```

SOAP request example

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

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header>
      <wsse:Security soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
         soapenv:mustUnderstand="0"
         xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <wsse:UsernameToken
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <wsse:Username>Native//admin</wsse:Username>
            <wsse:Password
               wsse:Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
               #PasswordText"
               >pass</wsse:Password>
            <wsse:Nonce>ofOShsZMIgHcdD0o6A8PkQ==</wsse:Nonce>
            <wsu:Created
               xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
               >2009-01-08T20:36:10Z</wsu:Created>
         </wsse:UsernameToken>
      </wsse:Security>
      <ns1:client-accept-language soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
         soapenv:mustUnderstand="0" xmlns:ns1="http://xml.spss.com/ws/headers">en-US;q=1.0,
         en;q=0.8</ns1:client-accept-language>
   </soapenv:Header>
```

```
<soapenv:Body>
  <getSamples xmlns="http://xml.spss.com/data/remote">
    <sampleDetails xmlns="http://xml.spss.com/data">
      <evURI>spsscr:///?id=ac140f8000072ffb0000010b290a16b28003#m.4:2009-02-12%2013:10:33.289</evURI>
      <avURI>spsscr:///?id=0a010a077e08b4a40000011f5cabdad38098#m.2:2009-02-11%2011:37:24.848</avURI>
      <dpdXML>&lt;?xml version=&quot;1.0&quot;
        encoding=&quot;UTF-8&quot;?&gt; &lt;rtDataProviderDefinition
        xmlns=&quot;http://xml.spss.com/pev&quot;
        applicationViewReference=&quot;spsscr:///?id=0a010a077e08b4a40000011f5cabdad38098&quot;&gt;
        &lt;dataSet name=&quot;Defect&quot;
        datasourceReference=&quot;spsscr:///?id=0a010a07e123f4280000011f5babe02080be&quot;
        qualifiedTableName=&quot;&amp;quot;cq_ecm_data&amp;quot;.&amp;quot;cq&amp;quot;
        .&amp;quot;Defect&amp;quot;&quot;
        credentialReference=&quot;spsscr:///?id=0a010a07e123f4280000011f5babe02080c4&quot;
        transactionIsolationLevel=&quot;NONE&quot;&gt;&lt;column
        name=&quot;internalcustomer&quot;
        type=&quot;integer&quot;/&gt;&lt;column
        name=&quot;ucm_stream&quot;
        type=&quot;string&quot;/&gt;&lt;column
        name=&quot;customer_severity&quot;
        type=&quot;string&quot;/&gt;&lt;column
        name=&quot;state&quot;
        type=&quot;integer&quot;/&gt;&lt;column
        name=&quot;release_commitment&quot;
        type=&quot;integer&quot;/&gt;&lt;column
        name=&quot;target_milestone&quot;
        type=&quot;string&quot;/&gt;&lt;column
        name=&quot;project&quot;
        type=&quot;integer&quot;/&gt;&lt;column
        name=&quot;release_commitment_date&quot;
        type=&quot;timestamp&quot;/&gt;&lt;column
        name=&quot;submitter&quot;
        type=&quot;integer&quot;/&gt;&lt;column
        name=&quot;internalcustomer_1&quot;
        type=&quot;string&quot;/&gt;&lt;column
        name=&quot;vendor&quot;
        type=&quot;integer&quot;/&gt;&lt;column
        name=&quot;locked_by&quot;
        type=&quot;integer&quot;/&gt;&lt;column
        name=&quot;deliverable&quot;
        type=&quot;integer&quot;/&gt;&lt;column
        name=&quot;verification_1&quot;
        type=&quot;string&quot;/&gt;&lt;column
        name=&quot;headline&quot;
        type=&quot;string&quot;/&gt;&lt;column
        name=&quot;ucm_vob_object&quot;
        type=&quot;string&quot;/&gt;&lt;column
        name=&quot;submit_date&quot;
        type=&quot;timestamp&quot;/&gt;&lt;column
        name=&quot;severity_1&quot;
        type=&quot;string&quot;/&gt;&lt;column
        name=&quot;targetbuild&quot;
```

type="string"/><column
name="automatedregression"
type="string"/><column
name="ucm_stream_object"
type="string"/><column
name="description"
type="string"/><column
name="priority"
type="string"/><column
name="component_os"
type="string"/><column
name="dbid"
type="integer"/><column
name="ucm_project"
type="integer"/><column
name="overall_status"
type="string"/><column
name="incident_type"
type="string"/><column
name="publicdescription"
type="string"/><column
name="ucm_view"
type="string"/><column
name="numberoftestvariations"
type="integer"/><column
name="lock_version"
type="integer"/><column
name="build_found"
type="string"/><column
name="unduplicate_state"
type="string"/><column
name="reportedby"
type="string"/><column
name="is_active"
type="integer"/><column
name="userimpact"
type="string"/><column
name="build_fixed"
type="string"/><column
name="reflist"
type="integer"/><column
name="risk_assessment"
type="string"/><column
name="rank"
type="string"/><column
name="oslanguage"
type="string"/><column
name="devcurest"
type="integer"/><column
name="old_system_id"
type="string"/><column
name="oldstate"

type="string"/><column
name="otherenvironment"
type="string"/><column
name="contactname"
type="string"/><column
name="note_entry"
type="string"/><column
name="contactemail"
type="string"/><column
name="applanguage"
type="string"/><column
name="legacy_id"
type="string"/><column
name="notes_log"
type="string"/><column
name="verification"
type="string"/><column
name="field_history"
type="string"/><column
name="feature"
type="integer"/><column
name="documentationfixed"
type="string"/><column
name="enhancement_ind"
type="string"/><column
name="component_project"
type="string"/><column
name="timeestimate"
type="string"/><column
name="devstart"
type="timestamp"/><column
name="version"
type="integer"/><column
name="qa_owner"
type="integer"/><column
name="fix_decision"
type="string"/><column
name="deflist"
type="integer"/><column
name="os"
type="integer"/><column
name="targetdate"
type="timestamp"/><column
name="verificationotes"
type="string"/><column
name="attachment_exists"
type="integer"/><column
name="symptoms"
type="string"/><column
name="component_suite"
type="string"/><column
name="field_history_1"

type="string"/><column
name="oemcustomer"
type="integer"/><column
name="ratl_mastership"
type="integer"/><column
name="documentationimpact"
type="string"/><column
name="devorigest"
type="integer"/><column
name="devend"
type="timestamp"/><column
name="old_id"
type="string"/><column
name="developmentimpact"
type="string"/><column
name="severity"
type="string"/><column
name="devactual"
type="integer"/><column
name="owner"
type="integer"/><column
name="testing_blocked_ind"
type="string"/><column
name="component"
type="integer"/><column
name="vendor_failure"
type="string"/><column
name="component_feature"
type="string"/><column
name="customer_reference"
type="string"/><column
name="suite"
type="integer"/><column
name="targetversion"
type="integer"/><column
name="resolution"
type="string"/><column
name="hardware"
type="string"/><column
name="keywords"
type="string"/><column
name="at_field_history"
type="integer"/><column
name="fixincurrentiteration"
type="string"/><column
name="contactphone"
type="string"/><column
name="id"
type="string"/><column
name="is_duplicate"
type="integer"/><column
name="component_deliverable"

```
                type=&quot;string&quot;/&gt;&lt;column
                name=&quot;customerid&quot;
                type=&quot;string&quot;/&gt;&lt;key
                name=&quot;fk_16780379_2&quot;
                isUnique=&quot;true&quot;&gt;&lt;keyColumn
                name=&quot;deflist&quot;
                type=&quot;integer&quot;/&gt;&lt;/key&gt;&lt;key
                name=&quot;fk_16780379_3&quot;
                isUnique=&quot;true&quot;&gt;&lt;keyColumn
                name=&quot;release_commitment&quot;
                type=&quot;integer&quot;/&gt;&lt;/key&gt;&lt;key
                name=&quot;fk_16779512_1&quot;
                isUnique=&quot;true&quot;&gt;&lt;keyColumn
                name=&quot;reflist&quot;
                type=&quot;integer&quot;/&gt;&lt;/key&gt;&lt;/dataSet&gt;&lt;tableMapping
                rootDataSet=&quot;Defect&quot;
                cacheTimeout=&quot;0&quot;&gt;&lt;pevCatalogTable
                name=&quot;CQdefects&quot;/&gt;&lt;columnMapping
                sourceColumnName=&quot;component&quot;
                isCached=&quot;false&quot;&gt;&lt;pevCatalogColumn
                name=&quot;component&quot;
                type=&quot;integer&quot;/&gt;&lt;/columnMapping&gt;&lt;columnMapping
                sourceColumnName=&quot;customerid&quot;
                isCached=&quot;false&quot;&gt;&lt;pevCatalogColumn
                name=&quot;customerid&quot;
                type=&quot;string&quot;/&gt;&lt;/columnMapping&gt;&lt;/tableMapping&gt;
                &lt;/rtDataProviderDefinition&gt;</dpdXML>
            <tableName>CQdefects</tableName>
            <resultColumnNames>customerid</resultColumnNames>
            <resultColumnNames>component</resultColumnNames>
            <keyValues name="custID">
               <columnName>customerid</columnName>
               <rowValues>
                  <value value="19029.00000"/>
               </rowValues>
            </keyValues>
            <sample>AV</sample>
         </sampleDetails>
      </getSamples>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a getSamples 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>
  <getSamplesResponse xmlns="http://xml.spss.com/data/remote">
    <sampleResult name="ResultTable" xmlns="http://xml.spss.com/data">
      <columnName>customerid</columnName>
      <columnName>component</columnName>
      <rowValues>
        <value value="19029.00000"/>
        <value value="0"/>
      </rowValues>
    </sampleResult>
  </getSamplesResponse>
</soapenv:Body>
</soapenv:Envelope>
```

The getTableMetaData operation

Returns the metadata for tables contained within a specified data source. Supply one or more table types to limit the results to specific types. For more information, see the topic The getTableTypes operation on p. 31.

The table metadata information returned can be used when calling the getDataSets operation. For more information, see the topic The getDataSets operation on p. 13.

Input fields

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

Table 4-5
Fields for getTableMetaData

Field	Type/Valid Values	Description
dataSourceURI	string	The datasource URI.
credentialsURI	string	The credentials URI.
typeNames	string[]	The table types.

Return information

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

Table 4-6
Return Value

Type	Description
tableMeta[]	This element is used to describe the table meta data within the datasource.

Java example

To access the table metadata for a data source, supply the getTableMetaData operation with three strings corresponding to the following:

- Repository uniform resource identifier for the data source

- Repository uniform resource identifier for valid credentials that can access the data source
- Table type(s)

The following sample returns the metadata for all tables of type *TABLE* in the data source.

```
String dsURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080be";
String credURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080c4";
String type = "TABLE";
TableMeta[] meta = stub.getTableMetaData(dsURI, credURI, type);

System.out.println("Table type = " + type);
System.out.println("CATALOG NAME\tSCHEMA NAME\tTABLE NAME\t" +
  "QUALIFIER SEPARATOR\tCATALOG PREFIX\tUSE CATALOG\t" +
  "IDENTIFIER QUOTE CHARACTER\n");
for (int i = 0; i < meta.length; i++) {
  System.out.println(meta[i].getCatalogName() + "\t" +
    meta[i].getSchemaName() + "\t" +
    meta[i].getTableName() + "\t" +
    meta[i].getQualifierSeparator() + "\t" +
    meta[i].getCatalogPrefixBool() + "\t" +
    meta[i].getUseCatalogBool() + "\t" +
    meta[i].getIdentifierQuoteCharacter());
}
```

For web service clients based on JAX-WS, replace the arrays in the sample with List collections and update the array processing accordingly. For example:

```
String dsURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080be";
String credURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080c4";
List<String> typeList = Arrays.asList("TABLE", "VIEW");
List<TableMeta> metaList = stub.getTableMetaData(dsURI, credURI, typeList);

System.out.println("Table type = " + type);
System.out.println("CATALOG NAME\tSCHEMA NAME\tTABLE NAME\t" +
  "QUALIFIER SEPARATOR\tCATALOG PREFIX\tUSE CATALOG\t" +
  "IDENTIFIER QUOTE CHARACTER\n");
for (TableMeta meta : metaList)
{
  System.out.println(meta[i].getCatalogName() + "\t" +
  meta.getSchemaName() + "\t" +
  meta.getTableName() + "\t" +
  meta.getQualifierSeparator() + "\t" +
  meta.getCatalogPrefixBool() + "\t" +
  meta.getUseCatalogBool() + "\t" +
  meta.getIdentifierQuoteCharacter());
}
```

SOAP request example

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

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsse:Username>Native//admin</wsse:Username>
        <wsse:Password
          wsse:Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
          #PasswordText"
          >pass</wsse:Password>
        <wsse:Nonce>ofOShsZMlgHcdD0o6A8PkQ==</wsse:Nonce>
        <wsu:Created
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          >2009-01-08T20:36:10Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
    <ns1:client-accept-language soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0" xmlns:ns1="http://xml.spss.com/ws/headers">en-US;q=1.0,
      en;q=0.8</ns1:client-accept-language>
  </soapenv:Header>
  <soapenv:Body>
    <getTableMetaData xmlns="http://xml.spss.com/data/remote">
      <dataSourceURI>spsscr:///?id=0a010a07e123f4280000011f5babe02080be</dataSourceURI>
      <credentialsURI>spsscr:///?id=0a010a07e123f4280000011f5babe02080c4</credentialsURI>
      <typeNames>TABLE</typeNames>
    </getTableMetaData>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a getTableMetaData 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>
    <getTableMetaDataResponse xmlns="http://xml.spss.com/data/remote">
      <tableMeta xmlns="http://xml.spss.com/data">
        <catalogName>cq_ecm_data</catalogName>
        <schemaName>cq</schemaName>
        <tableName>actiondef</tableName>
        <qualifierSeparator>.</qualifierSeparator>
        <catalogPrefixBool>true</catalogPrefixBool>
```

```
        <useCatalogBool>true</useCatalogBool>
        <identifierQuoteCharacter>&amp;quot;</identifierQuoteCharacter>
      </tableMeta>
      <tableMeta xmlns="http://xml.spss.com/data">
        <catalogName>cq_ecm_data</catalogName>
        <schemaName>cq</schemaName>
        <tableName>actiondef_usergroups</tableName>
        <qualifierSeparator>.</qualifierSeparator>
        <catalogPrefixBool>true</catalogPrefixBool>
        <useCatalogBool>true</useCatalogBool>
        <identifierQuoteCharacter>&amp;quot;</identifierQuoteCharacter>
      </tableMeta>
      <tableMeta xmlns="http://xml.spss.com/data">
        <catalogName>cq_ecm_data</catalogName>
        <schemaName>cq</schemaName>
        <tableName>attachments</tableName>
        <qualifierSeparator>.</qualifierSeparator>
        <catalogPrefixBool>true</catalogPrefixBool>
        <useCatalogBool>true</useCatalogBool>
        <identifierQuoteCharacter>&amp;quot;</identifierQuoteCharacter>
      </tableMeta>
      <tableMeta xmlns="http://xml.spss.com/data">
        <catalogName>cq_ecm_data</catalogName>
        <schemaName>cq</schemaName>
        <tableName>Defect</tableName>
        <qualifierSeparator>.</qualifierSeparator>
        <catalogPrefixBool>true</catalogPrefixBool>
        <useCatalogBool>true</useCatalogBool>
        <identifierQuoteCharacter>&amp;quot;</identifierQuoteCharacter>
      </tableMeta>
    </getTableMetaDataResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getTableSimpleColumns operation

Returns the names and types for columns in a specified table within a data source. The metadata describing the table can be obtained from the getTableMetaData operation. The column information returned can be used when comparing the column definitions for two tables.

Input fields

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

Table 4-7
Fields for getTableSimpleColumns

Field	Type/Valid Values	Description
dataSourceURI	string	The datasource URI.

Field	Type/Valid Values	Description
credentialsURI	string	The credentials URI.
tableMeta	tableMeta	This element is used to describe the table meta data within the datasource.

Return information

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

Table 4-8
Return Value

Type	Description
simpleColumn[]	This element is used to describe the column meta data.

Java example

To access the column information for a table in a data source, supply the getTableSimpleColumns operation with the following:

- String denoting the repository uniform resource identifier for the data source
- String denoting the repository uniform resource identifier for valid credentials that can access the data source
- TableMeta object describing the table

The following sample returns the column information for the table named *Defect* in the data source.

```
String dsURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080be";
String credURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080c4";
TableMeta meta = new TableMeta();
meta.setSchemaName("cq_ecm_data");
meta.setSchemaName("cq");
meta.setTableName("Defect");
meta.setQualifierSeparator(".");
meta.setCatalogPrefixBool("true");
meta.setUseCatalogBool("true");
meta.setIdentifierQuoteCharacter("&quot;");

SimpleColumn[] col = stub.getTableSimpleColumns(dsURI, credURI, meta);

System.out.println("COLUMN NAME\tTYPE\n");
for (int i = 0; i < col.length; i++) {
  System.out.println(col[i].getName() + "\t" +
    col[i].getType().toString());
}
```

For web service clients based on JAX-WS, replace the arrays in the sample with List collections and update the array processing accordingly. For example:

```
String dsURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080be";
String credURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080c4";
```

```
TableMeta meta = new TableMeta();
meta.setSchemaName("cq_ecm_data");
meta.setSchemaName("cq");
meta.setTableName("Defect");
meta.setQualifierSeparator(".");
meta.setCatalogPrefixBool("true");
meta.setUseCatalogBool("true");
meta.setIdentifierQuoteCharacter("&quot;");

List<SimpleColumn> colList = stub.getTableSimpleColumns(dsURI, credURI, meta);
System.out.println("COLUMN NAME\tTYPE\n");
for (SimpleColumn col : colList)
{
  System.out.println(col.getName() + "\t" +
    col.getType().toString());
}
```

SOAP request example

Client invocation of the getTableSimpleColumns 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>
    <getTableSimpleColumns xmlns="http://xml.spss.com/data/remote">
      <dataSourceURI>spsscr:///?id=0a010a07e123f4280000011f5babe02080be</dataSourceURI>
      <credentialsURI>spsscr:///?id=0a010a07e123f4280000011f5babe02080c4</credentialsURI>
      <tableMeta xmlns="http://xml.spss.com/data">
        <catalogName>cq_ecm_data</catalogName>
        <schemaName>cq</schemaName>
        <tableName>Defect</tableName>
        <qualifierSeparator>.</qualifierSeparator>
        <catalogPrefixBool>true</catalogPrefixBool>
        <useCatalogBool>true</useCatalogBool>
        <identifierQuoteCharacter>&quot;</identifierQuoteCharacter>
      </tableMeta>
    </getTableSimpleColumns>
  </soapenv:Body>
```

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

SOAP response example

The server responds to a getTableSimpleColumns 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>
    <getTableSimpleColumnsResponse xmlns="http://xml.spss.com/data/remote">
      <simpleColumn xmlns="http://xml.spss.com/data">
        <name>ratl_mastership</name>
        <type>integer</type>
      </simpleColumn>
      <simpleColumn xmlns="http://xml.spss.com/data">
        <name>dbid</name>
        <type>integer</type>
      </simpleColumn>
      <simpleColumn xmlns="http://xml.spss.com/data">
        <name>is_active</name>
        <type>integer</type>
      </simpleColumn>
      <simpleColumn xmlns="http://xml.spss.com/data">
        <name>id</name>
        <type>string</type>
      </simpleColumn>
      <simpleColumn xmlns="http://xml.spss.com/data">
        <name>state</name>
        <type>integer</type>
      </simpleColumn>
      <simpleColumn xmlns="http://xml.spss.com/data">
        <name>version</name>
        <type>integer</type>
      </simpleColumn>
    </getTableSimpleColumnsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getTableTypes operation

Returns the types of tables contained within a specified data source. The information returned can be used to limit the getTableMetaData call results to a specific table type. For more information, see the topic The getTableMetaData operation on p. 25.

Input fields

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

Table 4-9
Fields for getTableTypes

Field	Type/Valid Values	Description
dataSourceURI	string	The datasource URI.
credentialsURI	string	The credentials URI.

Return information

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

Table 4-10
Return Value

Type	Description
string[]	The table types of the datasource.

Java example

To access the table types for a data source, supply the getTableTypes operation with two strings corresponding to the repository uniform resource identifiers for the data source and valid credentials for accessing the data source.

```
String dsURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080be";
String credURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080c4";
String[] types = stub.getTableTypes(dsURI, credURI);

for (int i = 0; i < types.length; i++) {
  System.out.println(types[i]);
}
```

For web service clients based on JAX-WS, replace the arrays in the sample with List collections and update the array processing accordingly. For example:

```
String dsURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080be";
String credURI = "spsscr:///?id=0a010a07e123f4280000011f5babe02080c4";
List<String> typeList = stub.getTableTypes(dsURI, credURI);
for (String type : typeList)
{
   System.out.println(type);
}
```

SOAP request example

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

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsse:Username>Native//admin</wsse:Username>
        <wsse:Password
          wsse:Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
          #PasswordText"
          >pass</wsse:Password>
        <wsse:Nonce>ofOShsZMlgHcdD0o6A8PkQ==</wsse:Nonce>
        <wsu:Created
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          >2009-01-08T20:36:10Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
    <ns1:client-accept-language soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soapenv:mustUnderstand="0" xmlns:ns1="http://xml.spss.com/ws/headers">en-US;q=1.0,
      en;q=0.8</ns1:client-accept-language>
  </soapenv:Header>
  <soapenv:Body>
    <getTableTypes xmlns="http://xml.spss.com/data/remote">
      <dataSourceURI>spsscr:///?id=0a010a07e123f4280000011f5babe02080be</dataSourceURI>
      <credentialsURI>spsscr:///?id=0a010a07e123f4280000011f5babe02080c4</credentialsURI>
    </getTableTypes>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a getTableTypes 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>
    <getTableTypesResponse xmlns="http://xml.spss.com/data/remote">
      <typeNames>SYSTEM TABLE</typeNames>
      <typeNames>TABLE</typeNames>
      <typeNames>VIEW</typeNames>
    </getTableTypesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getVersion operation

Returns the version number of the service.

Return information

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

Table 4-11
Return Value

Type	Description
string	The service version number.

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/data/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/data/remote">
      <version>4.20.000</version>
    </getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Appendix

A

Microsoft® .NET Framework-based clients

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

1. Add a Service Reference. For more information, see the topic Adding a service reference on p. 35.
2. Configure the web service endpoint. For more information, see the topic Configuring the web service endpoint on p. 37.
3. Programmatically configure the necessary endpoint behaviors. For more information, see the topic Configuring endpoint behaviors on p. 38.
4. Exercise the web service. For more information, see the topic Exercising the service on p. 38.

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

Adding a service reference

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

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

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

Service reference modifications

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

To modify a service reference:

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

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

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

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

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

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

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

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

Configuring the web service endpoint

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

1. In Visual Studio, double-click the *app.config* file for the application (or *web.config* for a web-application).

2. Find the system.serviceModel element. Create it if it does not already exist.

3. Find the client element. Create it if it does not already exist.

4. Create a new endpoint element as a child of the client element.

5. Specify the appropriate service endpoint URL as the value of the *address* attribute.

6. Specify *basicHttpBinding* as the value of the *binding* attribute.

7. Specify the appropriate service contract as the value of the *contract* attribute. The service contract is the value of the service reference namespace appended with the service name.

8. Optionally specify a value for the *name* attribute that identifies a name for the endpoint configuration. If the *name* is blank, the configuration is used as the default for the service.

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

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

Configuring endpoint behaviors

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

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

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

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

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

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

Exercising the service

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

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

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

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

The corrected code is as follows:

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

Appendix

B

Notices

This information was developed for products and services offered worldwide.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Trademarks

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

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

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

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

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

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

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

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


Java

Index