IBM SPSS Collaboration and Deployment
Services
8.4

*Subscription Repository Service
Developer's Guide*

IBM

Note

Before using this information and the product it supports, read the information in "Notices" on page 105.

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® or JBoss Application 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.
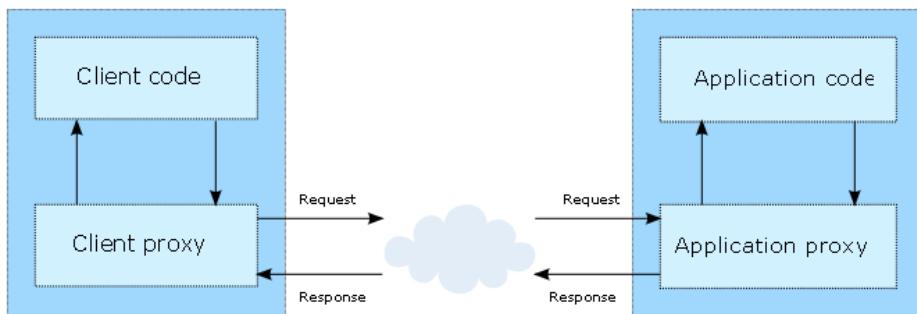
Figure 1. Web service architecture

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.
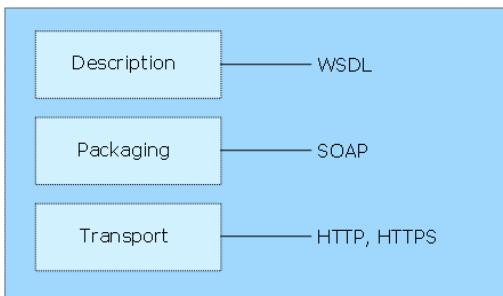
Figure 2. Web service protocol stack

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" on page 2 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:
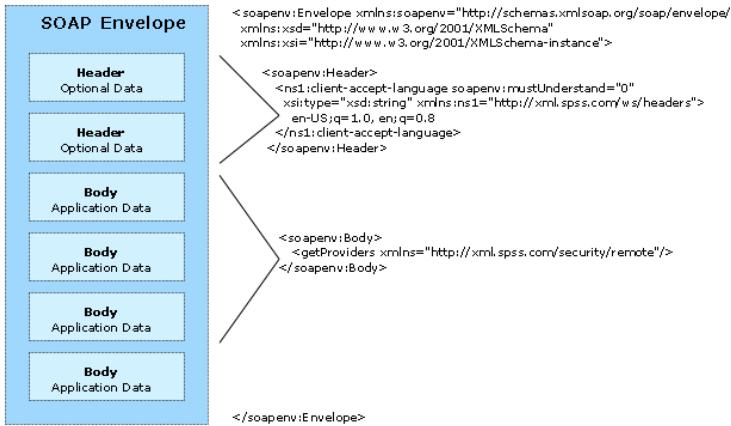
Figure 3. An example SOAP packet

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" on page 3 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" on page 4 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. Subscription Repository Service overview

The Subscription Repository Service provides a general facility for sending event notifications to subscribers. Components supplying events that can trigger notifications use the service to register the events and their properties in IBM SPSS Collaboration and Deployment Services. Clients can query the IBM SPSS Collaboration and Deployment Services Repository to discover the properties that they can expect to be contained in events of a certain type, and can write well-formed subscription expressions to match events in which they are interested. Subscribers subscribe to the events of interest, having the notification messages sent to any defined destinations.

Accessing the Subscription Repository Service

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

```
http://<host-name>:<port-number>/<context-root>/notification/services/SubscriptionRepository
```

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/notification/services/SubscriptionRepository?wsdl
```

Calling Subscription Repository 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 = "/notification/services/SubscriptionRepository";
URL url = new URL("http", "cads_server", 80, context);
SubscriptionRepositoryService service = new SubscriptionRepositoryServiceLocator();
stub = service.getSubscriptionRepository(url);
```

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

```
stub.createSubscription(subscriptionSpec);
```

Chapter 3. Subscription repository concepts

The Subscription Repository Service relies on the IBM SPSS Collaboration and Deployment Services Repository as both an event type repository and as a subscription repository. The event type repository stores information about the kinds of filterable data that named events provide to consumers, acting as a common reference so that the events transmitted by suppliers can contain the right properties to match against subscriptions based on expressions using those property names. The subscription repository, in contrast, stores information about the actual subscribers, subscriptions, and notification destinations.

Notification domains

Notification domains provide a classification scheme for events that can trigger notifications. Each domain represents a particular application or industry in which an event type is defined. For example, IBM SPSS Collaboration and Deployment Services includes the *PRMS* domain for job events and the *Repository* domain for repository object events. Each domain contains a set of event types. The Subscription Repository Service includes operations for adding, retrieving, updating, and deleting notification domains.

Event types

An event type represents a named system occurrence, such as the completion of a job, that can trigger the sending of a notification. Within the IBM SPSS Collaboration and Deployment Services Repository, event types are organized into notification domains to avoid naming conflicts. As a result, each event type is characterized by its domain name and its type name. In IBM SPSS Collaboration and Deployment Services, the *PRMS* notification domain includes event types for the completion of jobs and job steps. The *Repository* notification domain includes event types for changes in IBM SPSS Collaboration and Deployment Services Repository folders and files.

An event type defines zero or more properties on which a subscription can be based. Each property definition consists of a property name and a type code that identifies the type of data stored in the property. Valid type codes include *boolean*, *string, long, double, date, time, timestamp, duration, object, array*, and *ecma262*. The table identifies the properties available for the event types in the *PRMS* notification domain. Properties designated as *filterable* can be used in filter expressions for subscription selectors. See the topic "Subscription selectors" on page 12 for more information.

Table 1. PRMS domain event type properties				
Property	**Type Code**	**Filterable**	**Description**	**Event Type**
JobName	string	No	Name of a job	Completion
JobID	string	Yes	Internal identifier for a job	Completion
JobStart	timestamp	No	Time at which a job begins execution	Completion
JobEnd	timestamp	No	Time at which a job completes execution	Completion
JobSuccess	boolean	Yes	Indicator of whether or not a job successfully completes	Completion
JobStatusURL	string	No	URL at which the job status can be accessed	Completion
JobStepName	string	No	Name of a job step	JobStepCompletion

Table 1. PRMS domain event type properties (continued)				
Property	**Type Code**	**Filterable**	**Description**	**Event Type**
JobStepID	string	Yes	Internal identifier for a job step	JobStepCompletion
JobStepExecutionID	string	Yes	Unique internal identifier of the job step execution	JobStepCompletion
JobStepStart	timestamp	No	Time at which a job step begins execution	JobStepCompletion
JobStepEnd	timestamp	No	Time at which a job step completes execution	JobStepCompletion
JobStepSuccess	boolean	Yes	Indicator of whether or not a job step successfully completes	JobStepCompletion
JobStepCompletionCode	long	No	The return code from the job step process. A value of 0 indicates success.	JobStepCompletion
JobStepStatusURL	string	No	URL at which the job step status can be accessed	JobStepCompletion
Attachments	boolean	Yes	Indicator of whether or not the notification includes attachments	JobStepCompletion
JobStepArtifacts	array	No	List of results produced by a job step	JobStepCompletion

For example, a notification could be created for job completion indicating whether or not the job succeeded. The content of the notification could include the job name, ID, initiation time, completion time, and a URL for the status.

The table identifies the properties available for the event types in the *Repository* notification domain.

Table 2. Repository domain event type properties				
Property	**Type Code**	**Filterable**	**Description**	**Event Type(s)**
PathIdentifiers	string	No	Colon-delimited list of path identifiers of the resource	ResourceEvent, FolderEvent, FolderContentEvent, FileEvent
ResourceSpssCrUrl	string	No	The repository URL (`spsscr:`) of the resource	ResourceEvent, FolderEvent, FolderContentEvent, FileEvent
ResourcePath	string	No	Repository path of the resource	ResourceEvent, FolderEvent, FolderContentEvent, FileEvent

Property	Type Code	Filterable	Description	Event Type(s)
ResourceID	string	Yes	Identifier of the resource	ResourceEvent, FolderEvent, FolderContentEvent, FileEvent
ActionType	string	Yes	Action triggering the notification, such as *new version of a file*	ResourceEvent, FolderEvent, FolderContentEvent, FileEvent
ResourceHttpUrl	string	No	HTTP URL of the resource	ResourceEvent, FolderEvent, FolderContentEvent, FileEvent
ResourceName	string	No	Name of the resource	ResourceEvent, FolderEvent, FolderContentEvent, FileEvent
ChildHttpUrl	string	No	HTTP URL for the child resource for the folder event	FolderEvent, FolderContentEvent
ChildSpssCrUrl	string	No	The repository URL (`spsscr:`) for the child resource for the folder event	FolderEvent, FolderContentEvent
ChildPath	string	No	Path of the child resource for the folder event	FolderEvent, FolderContentEvent
ChildName	string	No	Name of the child resource for the folder event	FolderEvent, FolderContentEvent
MimeType	string	Yes	MIME type of the file resource in the folder	FolderContentEvent
Attachments	boolean	Yes	Indicator of whether or not the content of the updated file resource in the folder should be sent as a binary attachment.	FolderContentEvent
MimeType	string	Yes	MIME type of the file	FileEvent
Attachments	boolean	Yes	Indicator of whether or not the file content should be sent as a binary attachment	FileEvent

Table 2. Repository domain event type properties (continued)

The Subscription Repository Service includes operations for adding, retrieving, updating, and deleting event types.

Subscription selectors

Subscriptions typically share a common structure. For example, several users might need a notification if a scoring job fails. Rather than performing an individual query for each user, the Subscription Repository Service uses subscription selectors to provide a parameterized filtering mechanism for subscriptions. For example, consider the following query:

```
JobName='Scoring' and CompletionStatus='Failed'
```

This expression matches only the job named *Scoring* when the status is *Failed*. Alternatively, the query can be parameterized as:

```
JobName='$JobName' and CompletionStatus='$CompletionStatus'
```

$<JobName>* and *$<CompletionStatus> serve as placeholders for the actual property values. The parameterized query can be used for all job names and completion status values.

A subscription selector consists of a name and the filter expression used for matching an event with a subscription. Filter expressions can include the following items:

- Comparison operators: ==, !=, <, <=, >, >=
- Boolean connectors: and, or
- Property names for boolean, long, double, string, date, time, and timestamp properties defined for the event type
- Numeric and string constants
- Grouping operators

In the absence of grouping operators, the order of precedence for the comparison operators and boolean connectors is as shown, with comparisons performed before connections. Properties used in expressions correspond to those defined for the event type triggering the notification event. See the topic "Event types" on page 9 for more information. Strings in filtering expressions should be enclosed in single quotation marks, with two consecutive single quotation marks within a string representing a single quotation mark.

Typically, an application includes only a few subscription selectors. Individual subscribers supply values to use for the placeholders in the filtering expressions of the selectors to customize the subscription. The table lists subscription selectors available in IBM SPSS Collaboration and Deployment Services.

Table 3. IBM SPSS Collaboration and Deployment Services subscription selectors

Name	Filter Expression
`prms_completion`	`JobSuccess==true`
`prms_jobid_completion`	`JobID=='JobID' && JobSuccess==true`
`prms_jobstep_completion_success`	`JobStepID=='JobStepID' && JobStepSuccess==true && Attachments=true`
`prms_jobstep_completion_failure`	`JobStepID=='JobStepID' && JobStepSuccess==false && Attachments=true`
`selector_folder`	`ResourceID rlike '${ResourceID}'`
`selector_folder_link`	`ResourceID rlike '${ResourceID}' && Attachments==false && MimeType like '${MimeType}'`

Table 3. IBM SPSS Collaboration and Deployment Services subscription selectors (continued)	
Name	**Filter Expression**
`selector_folder_attachment`	`ResourceID rlike '${ResourceID}' && Attachments==true && MimeType like '${MimeType}'`
`selector_file_link`	`ResourceID=='${ResourceID}' && Attachments==false`
`selector_file_attachment`	`ResourceID=='${ResourceID}' && Attachments==true`

Property values specified for a subscription selector filtering expression serve as default values, and may be overridden by specific values supplied by the subscription using the selector.

The service generates an internal compiled version for each filter expression. In the compiled expression, the original property names are appended with a dollar sign and an integer indicating the order of appearance of the property in the filter expression. This notation allows for proper processing of expressions that contain the same property multiple times. For example, the filter expression:

```
(JobName='Modeling' and CompletionStatus='Failed') or
(JobName='Scoring' and NumRecord<1000000)
```

Yields a compiled expression of:

```
(JobName$0='${JobName}' and CompletionStatus$0='${CompletionStatus}') or
(JobName$1='${JobName}' and NumRecord$0<${NumRecord})
```

The compiled property names should be used when specifying values in subscriptions.

In addition to content based subscription selectors based on filter expressions, the Subscription Repository Service also allows subject based selectors. In this case, notification events are classified by groups or subjects and can be filtered only by their group. The system publishes an event using the corresponding item's category as its group name, and subscribers define the groups in which they are interested. Subject based subscriptions are defined only by the notification domains and event types.

The Subscription Repository Service includes operations for adding, retrieving, updating, and deleting subscription selectors.

Subscriptions

Typically, one subscription selector is associated with a large number of subscriptions. The subscriptions provide the specific property values used in the parameterized filter expression of the subscription selector. For example, the subscription selector contains the generic expression:

```
JobID = ${JobID}
```

The subscription provides a concrete value for the property placeholder, such as:

```
JobID = 0a0a4aac0161f10f0000010d8e63c9798ebd
```

A subscription consists of the following:

- A reference to a subscription selector.
- A list of values for use by the selector. Names for the property values correspond to the names used in the compiled filter expressions.

A subscription may also include an optional identifier for an object associated with the subscription, referred to as a *subscribable* object, such as a file. The subscribable object acts as the owner of the subscription. If the subscribable object is deleted from the system, the associated subscriptions are also removed.

The Subscription Repository Service includes operations for adding, retrieving, updating, and deleting subscriptions.

Subscribers

A subscriber represents the person receiving the notification message. The definition of a subscriber consists of the specification of the principal identifier for the person, which can be obtained from the Directory Information Service. Typically, a client application allows the subscriber to manage delivery devices, join shared subscriptions, or create individual subscriptions.

The Subscription Repository Service includes operations for adding, retrieving, updating, and deleting subscribers.

Distribution lists

Distribution lists serve as named containers for multiple subscribers. Adding a distribution list to a set of notification recipients is equivalent to adding each member of the distribution list individually. However, the ability to reference multiple subscribers through one list simplifies subscription management tasks. For example, instead of unsubscribing each member of the list, all list members can be removed from a notification simply by unsubscribing the list.

The Subscription Repository Service includes operations for adding, retrieving, updating, and deleting distribution lists. In addition, operations are available for adding subscribers to and removing subscribers from the lists.

Delivery devices

A delivery device represents any device that can receive notifications. The definition of a delivery device consists of a protocol type and a delivery address. IBM SPSS Collaboration and Deployment Services currently supports the SMTP and RSS/ATOM syndication protocols. For the SMTP protocol, the device address should be a valid email address. For syndication, the device address is the URL for the syndicated feed. The Subscription Repository Service includes operations for adding, retrieving, updating, and deleting delivery devices.

A subscriber can have any number of delivery devices. For a given subscriber, when a notification event occurs, the system attempts to deliver notification messages to all delivery devices.

Content formatters

Content formatters determine the structure and content of notification messages through the use of message templates. A template contains some fixed text and one or more content variables that are replaced with relevant information derived from the event triggering the notification. The content formatter defines the protocol for the notification and the name of the template file. Currently, the only supported protocol is SMTP. As a result, the template corresponds to an email message.

When processing a notification, for each content formatter, IBM SPSS Collaboration and Deployment Services searches the template directory tree for the named template matching the event type, locale, and protocol type in the following order:

```
<base-directory>/domain/event-type/locale/protocol
<base-directory>/domain/event-type/locale
<base-directory>/domain/event-type/protocol
<base-directory>/domain/event-type
<base-directory>/domain/locale/protocol
<base-directory>/domain/locale
<base-directory>/domain/protocol
<base-directory>/domain
<base-directory>/locale
<base-directory>/protocol
<base-directory>/
```

For example, the event types in the *PRMS* and *Repository* notification domains correspond to the following directory tree:

```
templates
    PRMS
        Completion
        JobStepCompletion
    Repository
        FileEvent
        FolderContentEvent
        FolderEvent
```

For IBM SPSS Collaboration and Deployment Services, the service finds the templates in the *<base-directory>/domain/event-type* directories.

The Subscription Repository Service includes operations for adding, retrieving, updating, and deleting content formatters.

Notification template structure

Notification message template structure

Notification templates transform event information into notification messages using Apache *Velocity* Template Language.

Velocity template structure

A Velocity template has a `*.vm` file extension. The template generates a message using the = operator to assign the `/mimeMessage/messageSubject`, `/mimeMessage/messageContent`, and `/mimeMessage/messageProperty` values that are subsequently parsed by the email processor. The following sample template generates a simple, generic email message indicating the success of the corresponding job.

```
/mimeMessage/messageSubject=Job Completion
/mimeMessage/messageContent[text/plain;charset=utf-8]=The job completed successfully.
```

For more information about Velocity templates, see the Apache Velocity project documentation.

Message properties

Email notification templates may include properties that determine how a message is processed in cases where SMTP settings different from repository defaults are to be used. For example, it may be necessary to specify a different SMTP server name and port number or the return email address assigned to the message. Default SMTP properties are listed under repository notification configuration options. If the Sun JVM is used with the repository installation, SMTP properties will correspond to the JavaMail API properties for message handling defined in the following table. Note that these properties may be different for different Java environments. For detailed information about SMTP properties, see the JVM vendor documentation.

Table 4. Message properties			
Message Property	**Attribute**	**Event Property**	**Description**
mail.debug	value	MailSmtpDebug	A Boolean value indicating the initial debug mode. The default is false.
mail.smtp.user	value	MailSmtpUser	The default SMTP username.
mail.smtp.password	value	MailSmtpPassword	The SMTP user password.
mail.smtp.host	value	MailSmtpHost	The SMTP server to which to connect.

Message Property	Attribute	Event Property	Description
Table 4. Message properties (continued)			
Message Property	**Attribute**	**Event Property**	**Description**
mail.smtp.port	value	MailSmtpPort	The SMTP server port to which to connect. The default is 25.
mail.smtp.connectiontimeout	value	MailSmtpConnectionTimeout	The socket connection timeout value in milliseconds. By default, the timeout is infinite.
	value	MailSmtpTimeout	The socket I/O timeout value in milliseconds. By default, the timeout is infinite.
mail.smtp.from	value	MailSmtpFrom	The email address used for the SMTP MAIL command. This sets the envelope return address.
mail.smtp.from	label	MailSmtpFromPersonal	The envelope return address label.
mail.smtp.localhost	value	MailSmtpLocalhost	The local hostname. The property should not normally need to be assigned if the JDK and name service are configured properly.
mail.smtp.ehlo	value	MailSmtpEhlo	A Boolean value indicating whether or not to sign on with the EHLO command. The default is true. Typically, failure of the EHLO command results in a fallback to the HELO command. This property should be used only for servers that do not fall back.
mail.smtp.auth	value	MailSmtpAuth	A Boolean value indicating whether or not to authenticate the user using the AUTH command. The default is false.
mail.smtp.dsn.notify	value	MailSmtpDsnNotify	Specifies the conditions under which the SMTP server should send delivery status notifications to the message sender. Valid values include: • NEVER indicates that no notification should be sent. • SUCCESS indicates that a notification should be sent on successful delivery only. • FAILURE indicates that a notification should be sent on a failed delivery only. • DELAY indicates that a notification should be sent when the message is delayed. Multiple values can be specified using a comma separator.

The syntax for defining these properties in a Velocity template is as follows:

- The property value must be assigned to `mimeMessage/messageProperty` with property name and label arguments in square brackets, as in the following example:

```
/mimeMessage/messageProperty[smtp.mail.smtp.from][Brian McGee]=bmagee@mycompany.com
```

- The value of property label is optional; therefore, the assignment statement can have the following syntax:

```
/mimeMessage/messageProperty[smtp.mail.smtp.from][]=bmagee@mycompany.com
```

- The values of property name and label can be assigned as static values or through variables referencing the corresponding event properties:

```
/mimeMessage/messageProperty[smtp.mail.smtp.from][$MailSmtpFromPersonal]=$MailSmtpFrom
```

Message content

The content of a notification message corresponds to the text supplied for the `messageSubject` and `messageContent` elements of the notification template. For either element, this text may include variable event property values.

- In Velocity templates, variable values are referenced using the $ notation. For example, `Job step $ {JobName}/${JobStepName} failed at ${JobStepEnd}` inserts the text with the current values for the *JobName*, *JobStepName*, and *JobStepEnd* properties.

The variables that can be inserted into a message reference the properties of the event that triggers the notification. Typical properties include:

- *JobName*, a string denoting the name of the job.
- *JobStart*, a timestamp indicating the time the job began.
- *JobEnd*, a timestamp indicating the time the job ended.
- *JobSuccess*, a Boolean value indicating whether or not the job was successful.
- *JobStatusURL*, a string corresponding to the URL at which the job status can be found.
- *JobStepName*, a string denoting the name of the job.
- *JobStepEnd*, a timestamp indicating the time the job ended.
- *JobStepArtifacts*, an array of string values denoting the URLs of the job step output.
- *JobStepStatusURL*, a string corresponding to the URL at which the job step status can be found.
- *ResourceName*, a string corresponding to the name of the object affected by the event, such as the file or folder name.
- *ResourcePath*, a string corresponding to the path of the object affected by the event.
- *ResourceHttpUrl*, a string corresponding to the HTTP URL at which the object can be found.
- *ChildName,* a string corresponding to the name of the child object of the parent object affected by the event. For example, when a file is created in a folder, this will be the name of the file.
- *ChildHttpUrl*, a string corresponding to the HTTP URL at which the child object can be found.
- *ActionType,* for repository events, the type of action that generated the event—for example, `FolderCreated`.

The available properties are defined by the event and will be different for different event types.

The following sample Velocity template for job step success notification inserts the names of the job and job step in the subject line. The content of the message also includes the end times for the step, the URL at which the status can be viewed, and a list of artifacts generated by the job step. Note that the template uses the *#foreach* loop structure to retrieve the URLs of the artifacts from the *JobStepArtifacts* property array.

```
<html>
<head>
```

```
<meta http-equiv='Content-Type' content='text/html;charset=utf-8'/>
</head>
<body>
<p>The job <b>${JobName}</b> started ${JobStart} and #if($JobSuccess) completed successfully #else failed #end ${JobEnd}.

<p>To review the job log, go to <a href='${JobStatusURL}'>${JobStatusURL}</a>.</p>

<hr><p>This is a machine-generated message. Please do not reply directly. If you do not want to receive this notification,
remove yourself from the notification list or contact your Repository administrator.</p>
</body>
</html>
```

The following code segments demonstrate how the Velocity template for folder content notification can be modified to remove the hyperlink to the job from the message. IBM SPSS Collaboration and Deployment Services jobs cannot be opened outside IBM SPSS Deployment Manager; therefore, it is strongly recommended to customize the notification message to remove the hyperlink. The additional if-condition in the example tests the MIME type of the object; if the object is a IBM SPSS Collaboration and Deployment Services job, the hyperlink is not included.

Original template:

```
#if($Attachments)
See attachment.
#else
<p>To review the content of the file, go to <a href='${ResourceHttpUrl}'>${ResourceHttpUrl}</a>.</p>
#end
```

Modified template:

```
#if($Attachments)
See attachment.
#else
#if($MimeType!='application/x-vnd.spss-prms-job')
<p>To review the content of the file, go to <a href='${ResourceHttpUrl}'>${ResourceHttpUrl}</a>.</p>
#end
#end
```

Message format

A notification template must specify the MIME type of the message content. In notification templates, the MIME type argument is specified in square brackets with `/mimeMessage/messageContent`.

The MIME type can have one of two values:

- *text/plain*. Notification messages appear in plain text. This is the default setting.
- *text/html*. Notification messages include HTML tags. Use this setting to control the appearance of the content within the message. The HTML within the message must be well-formed.

It is a good practice to always encode template output as Unicode (UTF-8).

HTML notification templates can take advantage of the functionality allowed in the markup. For example, the message can include a link to a Web page or to output from the job.

The following template generates a notification message for job step completion, formats content as a table, specifies background color for the message using an inline style for body, and defines a blue Verdana font for paragraphs using an internal style sheet. The message also includes a link to the job output.

```
/mimeMessage/messageSubject=${JobName}/${JobStepName} completed successfully
/mimeMessage/messageContent[text/html;charset=utf-8]=
          <html>
          <head>
          <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
          <style type="text/css">
          table {font-family: verdana; color: #000080}
          p {font-family: verdana; color: #000080}
          .foot {font-size: 75%; font-style: italic} </style>
          </head>
          <body style="background-color: #DCDCDC">
          <table border="8" align="center" width = 100%>
          <tr align="left">
          <th>Job/step name</th>
          <td>${JobName}/${JobStepName}</td>
          </tr>
          <tr align="left">
          <th>End time</th>
          <td> ${JobStepEnd}</td>
          </tr>
          <tr align="left">
          <th>Output</th>
          <td><p>
          #if ($JobStepArtifacts)
             #foreach($artifact in $JobStepArtifacts)
```

```
        <a href='$artifact.get("url")'>$artifact.get("filename")</a><br>
    #end
#else None <br>
#end
<p></td>
</tr>
</table>
<hr/>
<p class="foot">This is a machine generated message.
Please do not reply directly. If you do not wish to receive
this notification, unsubscribe or contact your
<a href="mailto:admin@mycompany.com"> your IBM SPSS Deployment
Services administrator.</a></p></body>
</html>
```

Chapter 4. Operation reference

The addContentFormatter operation

Adds a content formatter for a specified subscription.

Input fields

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

Table 5. Fields for addContentFormatter		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for a subscription.
contentFormatter	contentFormatter	Handles formatting of the notification message for the appropriate transport protocol.

Return information

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

Table 6. Return Value	
Type	**Description**
string	The identifier for the new content formatter.

Java example

To add a content formatter for a subscription:

1. Create a `ContentFormatter` object.
2. Define the protocol type for the formatter by supplying a `ProtocolType` value to the `setProtocolType` method.
3. Specify the name for the message template using the `setTemplateName` method.
4. Supply the `addContentFormatter` operation with a subscription identifier and the `ContentFormatter` object.

The operation returns a unique identifier for the new content formatter.

```
ContentFormatter cFormatter = new ContentFormatter();
cFormatter.setProtocolType(ProtocolType.SMTP);
cFormatter.setTemplateName("job_failure.xsl");
String subscriptionIdentifier = "0a0a4aac00072ffb000001094fa9f57ecace";
String contentFormatterIdentifier =
   stub.addContentFormatter(subscriptionIdentifier, cFormatter);
```

SOAP request example

Client invocation of the `addContentFormatter` 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>
      <addContentFormatter xmlns="http://xml.spss.com/notification/remote">
        <identifier>0a0a4aac00072ffb000001094fa9f57ecace</identifier>
        <contentFormatter templateName="job_failure.xsl" protocolType="smtp"
          xmlns="http://xml.spss.com/notification"/>
      </addContentFormatter>
    </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `addContentFormatter` 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>
    <addContentFormatterResponse xmlns="http://xml.spss.com/notification/remote">
      <contentFormatterIdentifier>0a0a4aac00072ffb000001094fa9f57ecad5</contentFormatterIdentifier>
    </addContentFormatterResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The addDeliveryDevice operation

Adds a delivery device for a specified subscriber. The Subscription Repository Service also provides a helper operation that allows the addition of a delivery device for the subscriber using a simplified form. This operation, `setSubscriber`, will create a new subscriber if necessary. See the topic "The setSubscriber operation" on page 76 for more information.

Input fields

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

Table 7. Fields for addDeliveryDevice

Field	Type/Valid Values	Description
identifier	string	An identifier for a subscriber.
deliveryDevice	deliveryDevice	Represents a device that can receive notifications.

Return information

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

Table 8. Return Value

Type	Description
string	The identifier for the new delivery device.

Java example

To add a delivery device for a subscriber, create a `DeliveryDevice` object. Define the protocol type for the device by supplying a `ProtocolType` value to `setProtocolType`. Specify the address to which to

send the notification using `setAddress`. The `addDeliveryDevice` operation adds the delivery device for the specified subscriber to the repository, returning a unique identifier for the delivery device.

```
DeliveryDevice deliveryDevice = new DeliveryDevice();
deliveryDevice.setProtocolType(ProtocolType.SMTP);
deliveryDevice.setAddress("mpower@spfd.com");
String subscriberIdentifier = "0a0a4aacfe9747c10000010dd18630f58036";
String deliveryDeviceIdentifier =
    stub.addDeliveryDevice(subscriberIdentifier, deliveryDevice);
```

SOAP request example

Client invocation of the `addDeliveryDevice` 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>
    <addDeliveryDevice xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aacfe9747c10000010dd18630f58036</identifier>
      <deliveryDevice xmlns="http://xml.spss.com/notification" protocolType="smtp"
          address="mpower@spfd.com"/>
    </addDeliveryDevice>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `addDeliveryDevice` 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>
    <addDeliveryDeviceResponse xmlns="http://xml.spss.com/notification/remote">
      <deliveryDeviceIdentifier>0a0a4aacfe9747c10000010dd18630f59de7</deliveryDeviceIdentifier>
    </addDeliveryDeviceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The addDerivedEventType operation

Creates a new event type derived from an existing parent event type. The property set for the new event type includes all properties defined for the parent plus any new properties defined for derived event type. The derived event type name, as well as the names for its properties, are subject to the following restrictions:

- Begins with a letter
- Consists of letters and digits
- Free from leading or trailing underscores, dollar signs, and spaces

Input fields

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

Table 9. Fields for addDerivedEventType		
Field	**Type/Valid Values**	**Description**
parentIdentifier	string	Identifier for the event type from which to derive the new event type.
eventType	eventType	Identifies a type of the event within its notification domain .

Return information

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

Table 10. Return Value	
Type	**Description**
string	An identifier for the new event type.

Java example

To add a derived event type to the system, create an `EventType` object. Use the `setDomainName` method to specify the name of the notification domain that contains the event type, and the `setTypeName` method to define the name of the event type.

To add a property to the event type, create a `Property` object. Use the `setName` method to specify a name for the property and the `setTypeCode` method to define the type of information stored in the property. The `setProperty` method of the `EventType` object adds the property to the event type.

Supply the `addDerivedEventType` operation with the identifier of the parent event type and the `EventType` object defining the new properties to add the derived event type to the notification domain. The following example derives the *FileAuthorEvent* event type from the *FileEvent* event. The new event type includes all *FileEvent* properties, plus a property corresponding to the author of the file.

```
EventType eventType = new EventType();
eventType.setDomainName("MyDomain");
eventType.setTypeName("FileAuthorEvent");
Property property = new Property();
property.setName("ResourceAuthor");
property.setTypeCode(TypeCode.STRING);
eventType.setProperty(property);
// Identifier for the FileEvent event type
String parentIdentifier = "00000000000000000000000000000001ff";
String identifier = stub.addDerivedEventType(parentIdentifier, eventType);
```

SOAP request example

Client invocation of the `addDerivedEventType` 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>
    <addDerivedEventType xmlns="http://xml.spss.com/notification/remote">
      <parentIdentifier>00000000000000000000000000000001ff</parentIdentifier>
      <eventType xmlns="http://xml.spss.com/notification" domainName="MyDomain"
          typeName="FileAuthorEvent">
        <property typeCode="string" name="ResourceAuthor"/>
      </eventType>
    </addDerivedEventType>
```

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

SOAP response example

The server responds to a `addDerivedEventType` 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>
    <addDerivedEventTypeResponse xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aacd90178eb0000010dea8eaedfdcaa</identifier>
    </addDerivedEventTypeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The addDistributionList operation

Adds a distribution list to the system.

Input fields

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

Table 11. Fields for addDistributionList

Field	Type/Valid Values	Description
distributionList	distributionList	Represents a logical grouping of the individual subscribers. For internal use only. This is reserved for future use.

Return information

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

Table 12. Return Value

Type	Description
string	The identifier for the new distribution list.

Java example

To add a distribution list, create a `DistributionList` object.Specify the name of the list using the `setName` method. The `addDistributionList` operation adds this object to the system, returning a string containing a unique system identifier for the new distribution list.

```
DistributionList distributionList = new DistributionList();
distributionList.setName("analysts");
String distributionListIdentifier = stub.addDistributionList(distributionList);
```

SOAP request example

Client invocation of the `addDistributionList` 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>
         <wsse:Username>validUser</wsse:Username>
         <wsse:Password>password</wsse:Password>
       </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>
     <addDistributionList xmlns="http://xml.spss.com/notification/remote">
       <distributionList xmlns="http://xml.spss.com/notification" name="analysts"/>
     </addDistributionList>
   </soapenv:Body>
 </soapenv:Envelope>
```

SOAP response example

The server responds to a `addDistributionList` 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>
    <addDistributionListResponse xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c698</identifier>
    </addDistributionListResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The addEventType operation

Adds an event type to the system. The event type name, as well as the names for its properties, are subject to the following restrictions:

- Begins with a letter
- Consists of letters and digits
- Free from leading or trailing underscores, dollar signs, and spaces

Input fields

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

Table 13. Fields for addEventType		
Field	**Type/Valid Values**	**Description**
eventType	eventType	Identifies a type of the event within its notification domain .

Return information

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

Table 14. Return Value	
Type	**Description**
string	The identifier for the new event type.

Java example

To add an event type to the system, create an `EventType` object. Use the `setDomainName` method to specify the name of the notification domain that contains the event type, and the `setTypeName` method to define the name of the event type.

To add a property to the event type, create a `Property` object. Use the `setName` method to specify a name for the property and the `setTypeCode` method to define the type of information stored in the property. The `setProperty` method of the `EventType` object adds the property to the event type.

When finished adding properties, use the `addEventType` operation to add the event type to the notification domain.

```
EventType eventType = new EventType();
eventType.setDomainName("MyDomain");
eventType.setTypeName("Completion");
Property property = new Property();
property.setName("JobID");
property.setTypeCode(TypeCode.STRING);
eventType.setProperty(property);
property = new Property();
property.setName("Success");
property.setTypeCode(TypeCode.BOOLEAN);
eventType.setProperty(property);
String identifier = stub.addEventType(eventType);
```

SOAP request example

Client invocation of the `addEventType` 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>
    <addEventType xmlns="http://xml.spss.com/notification/remote">
      <eventType xmlns="http://xml.spss.com/notification" domainName="MyDomain" typeName="Completion">
        <property typeCode="string" name="JobID"/>
        <property typeCode="boolean" name="Success"/>
      </eventType>
    </addEventType>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `addEventType` 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>
    <addEventTypeResponse xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aacfe9747c10000010dd18630f59dc9</identifier>
    </addEventTypeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The addNotificationDomain operation

Adds a notification domain to the system. The domain name is subject to the following restrictions:

- Begins with a letter
- Consists of letters and digits
- Free from leading or trailing underscores, dollar signs, and spaces

Input fields

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

Table 15. Fields for addNotificationDomain		
Field	**Type/Valid Values**	**Description**
notificationDomain	notificationDomain	Identifies the particular vertical industry domain or application in which the event type is defined.

Return information

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

Table 16. Return Value	
Type	**Description**
string	The identifier for the new notification domain.

Java example

To add a notification domain, create a `NotificationDomain` object. Use the `setName` method to define the name for the domain. The `addNotificationDomain` operation adds the notification domain to the system, returning a unique identifier for the domain.

```
NotificationDomain notificationDomain = new NotificationDomain();
notificationDomain.setName("MyDomain");
String identifier = stub.addNotificationDomain(notificationDomain);
```

SOAP request example

Client invocation of the `addNotificationDomain` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <addNotificationDomain xmlns="http://xml.spss.com/notification/remote">
      <notificationDomain xmlns="http://xml.spss.com/notification" name="MyDomain"/>
    </addNotificationDomain>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `addNotificationDomain` 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>
    <addNotificationDomainResponse xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979e87c</identifier>
    </addNotificationDomainResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The addSubscriber operation

Adds a subscriber to the system.

Input fields

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

Table 17. Fields for addSubscriber		
Field	**Type/Valid Values**	**Description**
subscriber	subscriber	Represents an individual subscriber.

Return information

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

Table 18. Return Value	
Type	**Description**
string	The identifier for the new subscriber.

Java example

To add a subscriber to the subscription repository, create a `Subscriber` object. Specify a unique principal identifier for the subscriber using `setPrincipalID`. The `addSubscriber` operation adds the subscriber to the repository, returning a unique identifier for the subscriber.

```
Subscriber subscriber = new Subscriber();
String principalID ="//u/smtp/lalgar@yahoo.com";
subscriber.setPrincipalID(principalID);
String subscriberIdentifier = stub.addSubscriber(subscriber);
```

SOAP request example

Client invocation of the `addSubscriber` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <addSubscriber xmlns="http://xml.spss.com/notification/remote">
      <subscriber xmlns="http://xml.spss.com/notification" principalID="//u/smtp/lalgar@yahoo.com"/>
    </addSubscriber>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `addSubscriber` 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>
    <addSubscriberResponse xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aacfe9747c10000010dd18630f581f5</identifier>
    </addSubscriberResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The addSubscribersToDistributionList operation

Adds one or more subscribers to a specified distribution list.

Input fields

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

Table 19. Fields for addSubscribersToDistributionList		
Field	**Type/Valid Values**	**Description**
subscriberIdentifier	string[]	Identifiers for subscribers being added.
distributionListIdentifier	string	Identifier for the distribution list.

Java example

To add subscribers to a distribution list, supply the `addSubscribersToDistributionList` operation with an array of identifiers for the subscribers and the identifier for the distribution list.

The following sample uses the `getObjects` method to return the `DistributionList` object having the name *analysts*. The `getIdentifier` method returns the identifier for this distribution list.

A string array defines the subscriber identifiers. The `addSubscribersToDistributionList` operation adds these subscribers to the *analysts* distribution list.

```
DistributionListCriterion dlCrit = new DistributionListCriterion();
dlCrit.setName("analysts");
QuerySpecification querySpec = new QuerySpecification();
querySpec.setDistributionListCriterion(dlCrit);
QueryResult result = stub.getObjects(querySpec);
DistributionList dList = result.getDistributionList(0);
String distributionListIdentifier = dList.getIdentifier();

String[] subscriberIdentifier = new String[2];
subscriberIdentifier[0] = "0a0a4aac0161f10f0000010d8e63c979c0d6";
subscriberIdentifier[1] = "0a0a4aac0161f10f0000010d8e63c979c09f";

stub.addSubscribersToDistributionList(subscriberIdentifier, distributionListIdentifier);
```

SOAP request example

Client invocation of the `addSubscribersToDistributionList` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
      <addSubscribersToDistributionList xmlns="http://xml.spss.com/notification/remote">
        <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c0d6</subscriberIdentifier>
        <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c09f</subscriberIdentifier>
        <distributionListIdentifier>0a0a4aacfe9747c10000010dd18630f581cb</distributionListIdentifier>
      </addSubscribersToDistributionList>
    </soapenv:Body>
  </soapenv:Envelope>
```

SOAP response example

The server responds to a addSubscribersToDistributionList 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>
    <addSubscribersToDistributionListResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The addSubscriberToDistributionList operation

Adds a subscriber to a specified distribution list.

Input fields

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

Table 20. Fields for addSubscriberToDistributionList		
Field	**Type/Valid Values**	**Description**
subscriberIdentifier	string	Identifier for the subscriber being added.
distributionListIdentifier	string	Identifier for the distribution list.

Java example

To add a subscriber to a distribution list, supply the addSubscriberToDistributionList operation
with identifiers for the subscriber and the distribution list. The following sample creates a new distribution
list, creates a new subscriber, and adds the subscriber to the list using the identifiers returned from the
creation steps.

```
DistributionList distributionList = new DistributionList();
distributionList.setName("analysts");
String distributionListIdentifier = stub.addDistributionList(distributionList);
Subscriber subscriber = new Subscriber();
String principalID ="//u/smtp/lalgar@yahoo.com";
subscriber.setPrincipalID(principalID);
String subscriberIdentifier = stub.addSubscriber(subscriber);
stub.addSubscriberToDistributionList(subscriberIdentifier, distributionListIdentifier);
```

SOAP request example

Client invocation of the addSubscriberToDistributionList 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>
```

```
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <addSubscriberToDistributionList xmlns="http://xml.spss.com/notification/remote">
      <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c09f</subscriberIdentifier>
      <distributionListIdentifier>0a0a4aac0161f10f0000010d8e63c979c698</distributionListIdentifier>
    </addSubscriberToDistributionList>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `addSubscriberToDistributionList` 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>
    <addSubscriberToDistributionListResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The addSubscriptionSelector operation

Adds a subscription selector for an event type.

Input fields

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

Table 21. Fields for addSubscriptionSelector		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for an event type.
subscriptionSelector	subscriptionSelector	Defines a subscription selector for the given event type.

Return information

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

Table 22. Return Value	
Type	**Description**
string	The identifier for the new subscription selector.

Java example

To add a subject based subscription selector for an event type, create a `SubscriptionSelector` object. Specify the name for the selector using the `setName` method. Supply the `addSubscriptionSelector` operation with an event type identifier and the `SubscriptionSelector` object to add the selector to the system, returning a unique identifier for the selector.

```
SubscriptionSelector subscriptionSelector = new SubscriptionSelector();
subscriptionSelector.setName("All_PRMS_Events");
String eventTypeIdentifier = "0a0a4aacfe9747c10000010dd18630f59dc9";
stub.addSubscriptionSelector(eventTypeIdentifier, subscriptionSelector);
```

SOAP request example

Client invocation of the `addSubscriptionSelector` 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>
    <addSubscriptionSelector xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aacfe9747c10000010dd18630f59dc9</identifier>
      <subscriptionSelector xmlns="http://xml.spss.com/notification" name="All_PRMS_Events"/>
    </addSubscriptionSelector>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `addSubscriptionSelector` 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>
    <addSubscriptionSelectorResponse xmlns="http://xml.spss.com/notification/remote">
      <subscriptionSelectorIdentifier>0a0a4aacfe9747c10000010dd18630f59e04</subscriptionSelectorIdentifier>
    </addSubscriptionSelectorResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The bulkSubscribe operation

Adds one or more subscribers to an existing subscription.

Input fields

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

Table 23. Fields for bulkSubscribe

Field	Type/Valid Values	Description
subscriberIdentifier	string[]	Identifiers for subscribers.
subscriptionIdentifier	string	An identifier for a subscription.

Java example

To add subscribers to a subscription, create an array of strings corresponding to the identifiers for the subscribers. Supply the `bulkSubscribe` operation with the array and a string corresponding to the subscription identifier.

```
String[] subscriberIdentifier = new String[2];
subscriberIdentifier[0] = "0a0a4aac0161f10f0000010d8e63c979c0d6";
subscriberIdentifier[1] = "0a0a4aac0161f10f0000010d8e63c979c09f";
String subscriptionIdentifier = "0a0a4aac0161f10f0000010d8e63c979c098";
stub.bulkSubscribe(subscriberIdentifier, subscriptionIdentifier);
```

SOAP request example

Client invocation of the `bulkSubscribe` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <bulkSubscribe xmlns="http://xml.spss.com/notification/remote">
      <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c0d6</subscriberIdentifier>
      <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c09f</subscriberIdentifier>
      <subscriptionIdentifier>0a0a4aac0161f10f0000010d8e63c979c098</subscriptionIdentifier>
    </bulkSubscribe>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `bulkSubscribe` 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>
    <bulkSubscribeResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The bulkUnsubscribe operation

Removes one or more subscribers from a subscription.

Input fields

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

Table 24. Fields for bulkUnsubscribe

Field	Type/Valid Values	Description
subscriberIdentifier	string[]	Identifiers for subscribers.
subscriptionIdentifier	string	An identifier for a subscription.

Java example

To remove subscribers from a subscription, create an array of strings corresponding to the identifiers for the subscribers. Supply the `bulkUnsubscribe` operation with the array and a string corresponding to the subscription identifier.

```
String[] subscriberIdentifier = new String[2];
subscriberIdentifier[0] = "0a0a4aac0161f10f0000010d8e63c979c0d6";
subscriberIdentifier[1] = "0a0a4aac0161f10f0000010d8e63c979c09f";
String subscriptionIdentifier = "0a0a4aac0161f10f0000010d8e63c979c098";
stub.bulkUnsubscribe(subscriberIdentifier, subscriptionIdentifier);
```

SOAP request example

Client invocation of the `bulkUnsubscribe` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <bulkUnsubscribe xmlns="http://xml.spss.com/notification/remote">
      <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c0d6</subscriberIdentifier>
      <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c09f</subscriberIdentifier>
      <subscriptionIdentifier>0a0a4aac0161f10f0000010d8e63c979c098</subscriptionIdentifier>
    </bulkUnsubscribe>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `bulkUnsubscribe` 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>
    <bulkUnsubscribeResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The createSubscription operation

Creates a subscription in the system.

Input fields

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

Table 25. Fields for createSubscription		
Field	**Type/Valid Values**	**Description**
subscriptionSpecification	subscriptionSpecification	Defines a set of parameters to create a subscription.

Return information

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

Table 26. Return Value	
Type	**Description**
subscription	Represents a single subscription.

Java example

To add a subscription to the subscription repository, create a `SubscriptionSpecification` object. Specify the principal ID for the subscriber using the `setPrincipalID` method. Define

the subscription selector using the `setSubscriptionSelectorIdentifier` method. Supply the `setSubscribableIdentifier` method with the resource ID of the repository object associated with the subscription.

To specify a value for a subscription property, create a `PropertyValue` object. Use the `setName` and `setTypeCode` methods to define the property name and type. Create a `Value` object for the property value and use the `setContent` method to actually specify the value. Use the `setValue` method to assign the value to the `PropertyValue` object. The `setPropertyValue` method assigns the value to the `SubscriptionSpecification` object.

Next, create an `EventType` object. Use the `setDomainName` and `setTypeName` methods to define the domain name and type name for the subscription. The `setEventType` method assigns the event type to the `SubscriptionSpecification` object.

The `createSubscription` operation adds the subscription to the repository, returning a `Subscription` object.

```
SubscriptionSpecification subscriptionSpecification =
    new SubscriptionSpecification();
String selectorIdentifier = "0000000000000000000000000000000000000200";
subscriptionSpecification.setSubscriptionSelectorIdentifier(selectorIdentifier);
subscriptionSpecification.setMulticasted(false);
subscriptionSpecification.setPrincipalID("//uNative//validUser");
subscriptionSpecification.setSubscribableIdentifier("0a0a4aac0161f10f0000010d8e63c9799c24");

PropertyValue propertyValue = new PropertyValue();
propertyValue.setName("ResourceID");
propertyValue.setTypeCode(TypeCode.STRING);
Value value = new Value();
value.setContent("0a0a4aac0161f10f0000010d8e63c9799c24");
propertyValue.setValue(value);
subscriptionSpecification.setPropertyValue(propertyValue);

EventType eventType = new EventType();
eventType.setDomainName("Repository");
eventType.setTypeName("FileEvent");
subscriptionSpecification.setEventType(eventType);

Subscription subscription = stub.createSubscription(subscriptionSpecification);
```

SOAP request example

Client invocation of the `createSubscription` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <createSubscription xmlns="http://xml.spss.com/notification/remote">
      <ns2:subscriptionSpecification
        subscriptionSelectorIdentifier="0000000000000000000000000000000000000200"
        multicasted="false" principalID="//uNative//validUser"
        xmlns:ns2="http://xml.spss.com/notification">
        <ns2:propertyValue typeCode="string" name="ResourceID">
          <ns2:value>0a0a4aac0161f10f0000010d8e63c9799c24</ns2:value>
        </ns2:propertyValue>
        <ns2:subscriptionSelectorIdentifier>
          0000000000000000000000000000000000000200
        </ns2:subscriptionSelectorIdentifier>
        <ns2:subscribableIdentifier>0a0a4aac0161f10f0000010d8e63c9799c24</ns2:subscribableIdentifier>
        <ns2:eventType domainName="Repository" typeName="FileEvent"/>
      </ns2:subscriptionSpecification>
    </createSubscription>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `createSubscription` 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>
      <createSubscriptionResponse xmlns="http://xml.spss.com/notification/remote">
         <ns1:subscription subscriptionSelectorIdentifier="00000000000000000000000000000000000200"
            multicasted="false" xmlns:ns1="http://xml.spss.com/notification">
            <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c03f</ns1:identifier>
            <ns1:propertyValue typeCode="string" name="ResourceID$0">
               <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c040</ns1:identifier>
               <ns1:value>0a0a4aac0161f10f0000010d8e63c9799c24</ns1:value>
            </ns1:propertyValue>
            <ns1:propertyValue typeCode="boolean" name="Attachments$0">
               <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c041</ns1:identifier>
               <ns1:value>false</ns1:value>
            </ns1:propertyValue>
         </ns1:subscription>
      </createSubscriptionResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

The deleteContentFormatter operation

Removes a specified content formatter from the system.

Input fields

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

Table 27. Fields for deleteContentFormatter

Field	Type/Valid Values	Description
identifier	string	A content formatter identifier.

Java example

To delete a content formatter, supply the `deleteContentFormatter` operation with the identifier for the content formatter.

```
String identifier = "0a0a4aac00072ffb000001094fa9f57ecad5";
stub.deleteContentFormatter(identifier);
```

SOAP request example

Client invocation of the `deleteContentFormatter` 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>
          <wsse:Username>validUser</wsse:Username>
          <wsse:Password>password</wsse:Password>
       </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>
      <deleteContentFormatter xmlns="http://xml.spss.com/notification/remote">
         <identifier>0a0a4aac00072ffb000001094fa9f57ecad5</identifier>
      </deleteContentFormatter>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteContentFormatter` 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>
    <deleteContentFormatterResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The deleteDeliveryDevice operation

Removes a specified delivery device from the system.

Input fields

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

Table 28. Fields for deleteDeliveryDevice

Field	Type/Valid Values	Description
identifier	string	A delivery device identifier.

Java example

To delete a delivery device, use the `deleteDeliveryDevice` operation. Supply the identifier of the delivery device to be deleted.

```
String deliveryDeviceIdentifier = "0a0a4aac0161f10f0000010d8e63c979c0a0";
stub.deleteDeliveryDevice(deliveryDeviceIdentifier);
```

SOAP request example

Client invocation of the `deleteDeliveryDevice` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <deleteDeliveryDevice xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c0a0</identifier>
    </deleteDeliveryDevice>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteDeliveryDevice` 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>
        <deleteDeliveryDeviceResponse xmlns="http://xml.spss.com/notification/remote"/>
    </soapenv:Body>
</soapenv:Envelope>
```

The deleteDistributionList operation

Removes a designated distribution list from the system.

Input fields

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

Table 29. Fields for deleteDistributionList		
Field	**Type/Valid Values**	**Description**
identifier	string	A distribution list identifier.

Java example

To delete a distribution list, supply the `deleteDistributionList` operation with the identifier for the distribution list.

```
String identifier = "0a0a4aac0161f10f0000010d8e63c979c68e";
stub.deleteDistributionList(identifier);
```

SOAP request example

Client invocation of the `deleteDistributionList` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <deleteDistributionList xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c68e</identifier>
    </deleteDistributionList>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteDistributionList` 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>
    <deleteDistributionListResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The deleteEventType operation

Removes a specified event type from the system.

Input fields

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

Table 30. Fields for deleteEventType		
Field	**Type/Valid Values**	**Description**
identifier	string	An event type identifier.

Java example

To delete an event type, supply the `deleteEventType` operation with the identifier for the event type.

```
String identifier = "0a0a4aacfe9747c10000010dd18630f59dc9";
stub.deleteEventType(identifier);
```

SOAP request example

Client invocation of the `deleteEventType` 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>
    <deleteEventType xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aacfe9747c10000010dd18630f59dc9</identifier>
    </deleteEventType>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteEventType` 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>
    <deleteEventTypeResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The deleteNotificationDomain operation

Removes a designated notification domain from the system.

Input fields

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

Table 31. Fields for deleteNotificationDomain		
Field	**Type/Valid Values**	**Description**
identifier	string	A notification domain identifier.

Java example

To delete a notification domain, supply the `deleteNotificationDomain` operation with the identifier for the domain.

```
String identifier = "0a0a4aac0161f10f0000010d8e63c979e87c";
stub.deleteNotificationDomain(identifier);
```

SOAP request example

Client invocation of the `deleteNotificationDomain` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <deleteNotificationDomain xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979e87c</identifier>
    </deleteNotificationDomain>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteNotificationDomain` 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>
    <deleteNotificationDomainResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The deleteSubscriber operation

Removes a specified subscriber from the system.

Input fields

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

Table 32. Fields for deleteSubscriber		
Field	**Type/Valid Values**	**Description**
identifier	string	A subscriber identifier.

Java example

To delete a subscriber, use `deleteSubscriber`. Supply the identifier of the subscriber to be deleted.

```
String subscriberIdentifier = "0a0a4aac0161f10f0000010d8e63c979c09f";
stub.deleteSubscriber(subscriberIdentifier);
```

SOAP request example

Client invocation of the `deleteSubscriber` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <deleteSubscriber xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c09f</identifier>
    </deleteSubscriber>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteSubscriber` 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>
    <deleteSubscriberResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The deleteSubscribers operation

Input fields

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

Field	Type/Valid Values	Description
identifier	string[]	A subscriber identifier.

Table 33. Fields for deleteSubscribers

Java example

To delete multiple subscribers, use the `deleteSubscribers` operation. Supply an array containing the identifiers of the subscribers to be deleted.

```
String[] subscriberIdentifier = new String[2];
subscriberIdentifier[0] = "0a0a4aacfe9747c10000010dd18630f581f5";
subscriberIdentifier[1] = "0a0a4aacfe9747c10000010dd18630f5809c";
stub.deleteSubscribers(subscriberIdentifier);
```

SOAP request example

Client invocation of the `deleteSubscribers` 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>
            <wsse:Username>validUser</wsse:Username>
            <wsse:Password>password</wsse:Password>
          </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>
        <deleteSubscribers xmlns="http://xml.spss.com/notification/remote">
          <identifier>0a0a4aacfe9747c10000010dd18630f581f5</identifier>
          <identifier>0a0a4aacfe9747c10000010dd18630f5809c</identifier>
        </deleteSubscribers>
    </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteSubscribers` 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>
      <deleteSubscribersResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The deleteSubscription operation

Removes a designated subscription from the system.

Input fields

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

Table 34. Fields for deleteSubscription		
Field	**Type/Valid Values**	**Description**
identifier	string	A subscription identifier.

Java example

To delete a subscription, supply the `deleteSubscription` operation with the identifier for the subscription.

```
String identifier = "0a0a4aac0161f10f0000010d8e63c979c098";
stub.deleteSubscription(identifier);
```

SOAP request example

Client invocation of the `deleteSubscription` 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>
          <wsse:Username>validUser</wsse:Username>
          <wsse:Password>password</wsse:Password>
        </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>
        <deleteSubscription xmlns="http://xml.spss.com/notification/remote">
          <identifier>0a0a4aac0161f10f0000010d8e63c979c098</identifier>
        </deleteSubscription>
      </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteSubscription` 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>
    <deleteSubscriptionResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The deleteSubscriptions operation

Input fields

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

Table 35. Fields for deleteSubscriptions		
Field	**Type/Valid Values**	**Description**
identifier	string[]	A subscription identifier.

Java example

To delete one or more subscriptions, supply the `deleteSubscriptions` operation with the identifiers for the subscriptions.

```
String identifier = "0a0a4aac0161f10f0000010d8e63c979c03f";
stub.deleteSubscriptions(identifier);
```

SOAP request example

Client invocation of the `deleteSubscriptions` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <deleteSubscriptions xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c03f</identifier>
    </deleteSubscriptions>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteSubscriptions` 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>
      <deleteSubscriptionsResponse xmlns="http://xml.spss.com/notification/remote"/>
   </soapenv:Body>
</soapenv:Envelope>
```

The deleteSubscriptionSelector operation

Removes a specified subscription selector from the system.

Input fields

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

Table 36. Fields for deleteSubscriptionSelector

Field	Type/Valid Values	Description
identifier	string	A subscription selector identifier.

Java example

To delete a subscription selector, supply the `deleteSubscriptionSelector` operation with the identifier for the subscription selector.

```
String identifier = "0000000000000000000000000000000001fd";
stub.deleteSubscriptionSelector(identifier);
```

SOAP request example

Client invocation of the `deleteSubscriptionSelector` 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>
            <wsse:Username>validUser</wsse:Username>
            <wsse:Password>password</wsse:Password>
         </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>
      <deleteSubscriptionSelector xmlns="http://xml.spss.com/notification/remote">
         <identifier>0000000000000000000000000000000001fd</identifier>
      </deleteSubscriptionSelector>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `deleteSubscriptionSelector` 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>
        <deleteSubscriptionSelectorResponse xmlns="http://xml.spss.com/notification/remote"/>
    </soapenv:Body>
</soapenv:Envelope>
```

The getContentFormatter operation

Retrieves the content formatter corresponding to the specified identifier.

Input fields

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

Table 37. Fields for getContentFormatter

Field	Type/Valid Values	Description
identifier	string	An identifier for a content formatter.

Return information

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

Table 38. Return Value

Type	Description
contentFormatter	Handles formatting of the notification message for the appropriate transport protocol.

Java example

The information available in the `ContentFormatter` object returned by the `getContentFormatter` operation includes the following:

- Protocol type
- Template name
- Identifier for the formatter

The following example uses the `getProtocolType`, `getTemplateName`, and `getIdentifier` methods to access the content formatter characteristics.

```
String formatterIdentifier = "0a0a4aacd90178eb0000010dea8eaedf802a";
ContentFormatter formatter = stub.getContentFormatter(formatterIdentifier);
System.out.println("Identifier: " + formatter.getIdentifier());
System.out.println("Protocol Type: " + formatter.getProtocolType());
System.out.println("Template Name: " + formatter.getTemplateName());
```

SOAP request example

Client invocation of the `getContentFormatter` 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>
```

```
    <getContentFormatter xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aacd90178eb0000010dea8eaedf802a</identifier>
    </getContentFormatter>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getContentFormatter` 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>
    <getContentFormatterResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:contentFormatter templateName="job_failure.xsl" protocolType="smtp"
        templateProviderType="file" xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a4aacd90178eb0000010dea8eaedf802a</ns1:identifier>
      </ns1:contentFormatter>
    </getContentFormatterResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getContentFormatters operation

Retrieves all content formatters associated with a specified subscription.

Input fields

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

Table 39. Fields for getContentFormatters		
Field	**Type/Valid Values**	**Description**
identifier	string	

Return information

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

Table 40. Return Value	
Type	**Description**
contentFormatter[]	Handles formatting of the notification message for the appropriate transport protocol.

Java example

The information available in the `ContentFormatter` objects returned by the `getContentFormatters` operation includes the following:

- Protocol type
- Template name
- Identifier for the formatter

The following example iterates through the returned objects, using the `getProtocolType`, `getTemplateName`, and `getIdentifier` methods to access the content formatter characteristics.

```
String subscriptionIdentifier = "0a0a4aacd90178eb0000010dea8eaedf801b";
ContentFormatter[] formatter = stub.getContentFormatters(subscriptionIdentifier);
for (int j = 0; j < formatter.length; j++) {
   System.out.println("Identifier: " + formatter[j].getIdentifier());
   System.out.println("Protocol Type: " + formatter[j].getProtocolType());
   System.out.println("Template Name: " + formatter[j].getTemplateName());
   System.out.println();
}
```

SOAP request example

Client invocation of the `getContentFormatters` 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>
        <getContentFormatters xmlns="http://xml.spss.com/notification/remote">
            <identifier>0a0a4aacd90178eb0000010dea8eaedf801b</identifier>
        </getContentFormatters>
    </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getContentFormatters` 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>
        <getContentFormattersResponse xmlns="http://xml.spss.com/notification/remote">
            <ns1:contentFormatter templateName="job_failure.xsl" protocolType="smtp"
                templateProviderType="file" xmlns:ns1="http://xml.spss.com/notification">
                <ns1:identifier>0a0a4aacd90178eb0000010dea8eaedf802a</ns1:identifier>
            </ns1:contentFormatter>
        </getContentFormattersResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

The getDeliveryDevice operation

Returns the delivery device for a specified device identifier.

Input fields

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

Table 41. Fields for getDeliveryDevice		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for a delivery device.

Return information

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

Table 42. Return Value	
Type	**Description**
deliveryDevice	Represents a device that can receive notifications.

Java example

The information available in the `DeliveryDevice` object returned by the `getDeliveryDevice` operation includes the following:

- Protocol type for the device
- Address associated with the device
- Device identifier

The following example uses the `getProtocolType`, `getAddress`, and `getIdentifier` methods to access the delivery device characteristics.

```
String deviceIdentifier = "0a0a4aac0161f10f0000010d8e63c979c0a0";
DeliveryDevice dDevice = stub.getDeliveryDevice(deviceIdentifier);
System.out.println("Identifier: " + dDevice.getIdentifier());
System.out.println("Protocol Type: " + dDevice.getProtocolType());
System.out.println("Address: " + dDevice.getAddress());
```

SOAP request example

Client invocation of the `getDeliveryDevice` 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>
    <getDeliveryDevice xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c0a0</identifier>
    </getDeliveryDevice>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getDeliveryDevice` 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>
    <getDeliveryDeviceResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:deliveryDevice protocolType="smtp" address="jjones@yahoo.com"
          xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c0a0</ns1:identifier>
      </ns1:deliveryDevice>
    </getDeliveryDeviceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getDeliveryDevices operation

Returns all delivery devices for a specified subscriber.

Input fields

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

Table 43. Fields for getDeliveryDevices		
Field	**Type/Valid Values**	**Description**
identifier	string	

Return information

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

Table 44. Return Value	
Type	**Description**
deliveryDevice[]	Represents a device that can receive notifications.

Java example

The information available in the `DeliveryDevice` objects returned by the `getDeliveryDevices` operation includes the following:

- Protocol type for the device
- Address associated with the device
- Device identifier

The following example iterates through the returned objects, using the `getProtocolType`, `getAddress`, and `getIdentifier` methods to access the delivery device characteristics.

```
String subscriberIdentifier = "0a0a4aac00072ffb000001094fa9f57ea2df";
DeliveryDevice[] dDevice = stub.getDeliveryDevices(subscriberIdentifier);
for (int j = 0; j < dDevice.length; j++) {
    System.out.println("Identifier: " + dDevice[j].getIdentifier());
    System.out.println("Protocol Type: " + dDevice[j].getProtocolType());
    System.out.println("Address: " + dDevice[j].getAddress());
    System.out.println();
}
```

SOAP request example

Client invocation of the `getDeliveryDevices` 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>
    <getDeliveryDevices xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac00072ffb000001094fa9f57ea2df</identifier>
    </getDeliveryDevices>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getDeliveryDevices` 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>
    <getDeliveryDevicesResponse xmlns="http://xml.spss.com/notification/remote">
      <deliveryDevice protocolType="smtp" address="pemuser@mycompany.com"
        xmlns="http://xml.spss.com/notification">
        <identifier>0a0a4aac00072ffb000001094fa9f57ea2e0</identifier>
      </deliveryDevice>
    </getDeliveryDevicesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getDerivedEventTypes operation

Returns all event types derived from a specified event type.

Input fields

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

Table 45. Fields for getDerivedEventTypes		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for an event type.

Return information

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

Table 46. Return Value	
Type	**Description**
eventType[]	Identifies a type of the event within its notification domain .

Java example

The information available in the `EventType` objects returned by the `getDerivedEventTypes` operation includes the following:

- Domain name
- Event type name
- Event type properties
- Identifier for the event type

The following example iterates through the returned objects, using accessor methods to retrieve the individual event type characteristics.

```
String typeIdentifier = "00000000000000000000000000000001f5";
EventType[] eventType = stub.getDerivedEventTypes(typeIdentifier);
for (int j = 0; j < eventType.length; j++) {
   System.out.println("Identifier: " + eventType[j].getIdentifier());
   System.out.println("Domain: " + eventType[j].getDomainName());
   System.out.println("Type name: " + eventType[j].getTypeName());
   PropertyValue[] propValue = eventType[j].getPropertyValue();
   for (int k = 0; k < propValue.length; k++) {
      System.out.println("Property: " + propValue[k].getName() +
         " (type = " + propValue[k].getTypeCode() + ") = " +
         propValue[k].getDescription);
   }
   System.out.println();
}
```

SOAP request example

Client invocation of the `getDerivedEventTypes` 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>
      <getDerivedEventTypes xmlns="http://xml.spss.com/notification/remote">
        <identifier>00000000000000000000000000000001f5</identifier>
      </getDerivedEventTypes>
    </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getDerivedEventTypes` 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>
    <getDerivedEventTypesResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:eventType domainName="Repository" typeName="FolderEvent"
          xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>00000000000000000000000000000001fa</ns1:identifier>
        <ns1:property descriptionKey="$$repository/etr_folder_child_http_desc"
          labelKey="$$repository/etr_folder_child_http"
          description="The HTTP URL for the the child resource for the folder event."
          label="Child HTTP" typeCode="string" name="ChildHttpUrl">
          <ns1:identifier>0000000000000000000000000000020d</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_folder_child_spsscr_desc"
          labelKey="$$repository/etr_folder_child_spsscr"
          description="The SPSSCR URL for the the child resource for the folder event."
          label="Child SPSSCR" typeCode="string" name="ChildSpssCrUrl">
          <ns1:identifier>0000000000000000000000000000020e</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_folder_child_path_desc"
          labelKey="$$repository/etr_folder_child_path"
          description="The path of the child resource for the folder event."
          label="Child Path" typeCode="string" name="ChildPath">
          <ns1:identifier>0000000000000000000000000000020c</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_folder_child_name_desc"
          labelKey="$$repository/etr_folder_child_name"
          description="The name of the child resource for the folder event."
          label="Child Name" typeCode="string" name="ChildName">
          <ns1:identifier>0000000000000000000000000000020b</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_path_ids_desc"
          labelKey="$$repository/etr_path_ids_name"
          description="The path identifiers of the resource separated by colons."
          label="Path Identifiers" typeCode="string" name="PathIdentifiers"
          inheritedFrom="00000000000000000000000000000001f5">
          <ns1:identifier>00000000000000000000000000000207</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_resource_spsscr_desc"
          labelKey="$$repository/etr_resource_spsscr_url"
          description="The SPSSCR URL of the resource." label="SPSSCR URL"
          typeCode="string" name="ResourceSpssCrUrl"
          inheritedFrom="00000000000000000000000000000001f5">
          <ns1:identifier>0000000000000000000000000000020a</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_resource_path_desc"
          labelKey="$$repository/etr_resource_path"
          description="The path of the resource in the repository." label="Resource Path"
          typeCode="string" name="ResourcePath"
          inheritedFrom="00000000000000000000000000000001f5">
          <ns1:identifier>00000000000000000000000000000205</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_resource_id_desc"
          labelKey="$$repository/etr_resource_id"
          description="The identifier of the subscribable resource." label="Resource ID"
          typeCode="string" name="ResourceID"
          inheritedFrom="00000000000000000000000000000001f5">
          <ns1:identifier>00000000000000000000000000000001f6</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_action_type_desc"
          labelKey="$$repository/etr_action_type"
          description="The type of the user action, for example, new file created,
            new version of the file created and so on."
          label="Action Type" typeCode="string" name="ActionType"
          inheritedFrom="00000000000000000000000000000001f5">
          <ns1:identifier>00000000000000000000000000000204</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_resource_http_url_desc"
          labelKey="$$repository/etr_resource_http_url"
          description="The HTTP URL of the resource." label="Resource HTTP"
          typeCode="string" name="ResourceHttpUrl"
          inheritedFrom="00000000000000000000000000000001f5">
          <ns1:identifier>00000000000000000000000000000209</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_resource_name_desc"
          labelKey="$$repository/etr_resource_name"
          description="The name of the resource." label="Resource Name" typeCode="string"
          name="ResourceName" inheritedFrom="00000000000000000000000000000001f5">
          <ns1:identifier>00000000000000000000000000000206</ns1:identifier>
        </ns1:property>
      </ns1:eventType>
      <ns2:eventType domainName="Repository" typeName="FileEvent"
          xmlns:ns2="http://xml.spss.com/notification">
        <ns2:identifier>00000000000000000000000000000001ff</ns2:identifier>
        <ns2:property descriptionKey="$$repository/etr_mime_type_desc"
          labelKey="$$repository/etr_mime_type"
          description="The MIME type of the resource." label="MIME type" typeCode="string"
```

```
        name="MimeType">
        <ns2:identifier>0000000000000000000000000000000001f7</ns2:identifier>
      </ns2:property>
      <ns2:property descriptionKey="$$repository/etr_attachments_desc"
        labelKey="$$repository/etr_attachments"
        description="If true the content of the resource should be sent as a binary attachment."
        label="Attachments" typeCode="boolean" name="Attachments">
        <ns2:identifier>0000000000000000000000000000000001f8</ns2:identifier>
      </ns2:property>
      <ns2:property descriptionKey="$$repository/etr_path_ids_desc"
        labelKey="$$repository/etr_path_ids_name"
        description="The path identifiers of the resource separated by colons."
        label="Path Identifiers" typeCode="string" name="PathIdentifiers"
        inheritedFrom="0000000000000000000000000000000001f5">
        <ns2:identifier>0000000000000000000000000000000000207</ns2:identifier>
      </ns2:property>
      <ns2:property descriptionKey="$$repository/etr_resource_spsscr_desc"
        labelKey="$$repository/etr_resource_spsscr_url"
        description="The SPSSCR URL of the resource." label="SPSSCR URL"
        typeCode="string" name="ResourceSpssCrUrl"
        inheritedFrom="0000000000000000000000000000000001f5">
        <ns2:identifier>000000000000000000000000000000000020a</ns2:identifier>
      </ns2:property>
      <ns2:property descriptionKey="$$repository/etr_resource_path_desc"
        labelKey="$$repository/etr_resource_path"
        description="The path of the resource in the repository." label="Resource Path"
        typeCode="string" name="ResourcePath"
        inheritedFrom="0000000000000000000000000000000001f5">
        <ns2:identifier>0000000000000000000000000000000000205</ns2:identifier>
      </ns2:property>
      <ns2:property descriptionKey="$$repository/etr_resource_id_desc"
        labelKey="$$repository/etr_resource_id"
        description="The identifier of the subscribable resource." label="Resource ID"
        typeCode="string" name="ResourceID"
        inheritedFrom="0000000000000000000000000000000001f5">
        <ns2:identifier>0000000000000000000000000000000001f6</ns2:identifier>
      </ns2:property>
      <ns2:property descriptionKey="$$repository/etr_action_type_desc"
        labelKey="$$repository/etr_action_type"
        description="The type of the user action, for example, new file created,
          new version of the file created and so on."
        label="Action Type" typeCode="string" name="ActionType"
        inheritedFrom="0000000000000000000000000000000001f5">
        <ns2:identifier>0000000000000000000000000000000000204</ns2:identifier>
      </ns2:property>
      <ns2:property descriptionKey="$$repository/etr_resource_http_url_desc"
        labelKey="$$repository/etr_resource_http_url"
        description="The HTTP URL of the resource." label="Resource HTTP"
        typeCode="string" name="ResourceHttpUrl"
        inheritedFrom="0000000000000000000000000000000001f5">
        <ns2:identifier>0000000000000000000000000000000000209</ns2:identifier>
      </ns2:property>
      <ns2:property descriptionKey="$$repository/etr_resource_name_desc"
        labelKey="$$repository/etr_resource_name"
        description="The name of the resource." label="Resource Name" typeCode="string"
        name="ResourceName" inheritedFrom="0000000000000000000000000000000001f5">
        <ns2:identifier>0000000000000000000000000000000000206</ns2:identifier>
      </ns2:property>
    </ns2:eventType>
  </getDerivedEventTypesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getDistributionList operation

Retrieves the distribution list corresponding to the specified identifier.

Input fields

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

Table 47. Fields for getDistributionList		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for a distribution list.

Return information

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

Table 48. Return Value	
Type	**Description**
distributionList	Represents a logical grouping of the individual subscribers. For internal use only. This is reserved for future use.

Java example

The information available in the `DistributionList` object returned by the `getDistributionList` operation includes the following:

- The list name
- The identifier for the list

The following example uses the `getName` and `getIdentifier` methods to access the distribution list characteristics.

```
String listIdentifier = "0a0a4aac0161f10f0000010d8e63c979c698";
DistributionList dList = stub.getDistributionList(listIdentifier);
System.out.println("Identifier " + dList.getIdentifier() +
    " corresponds to distribution list " + dList.getName());
```

SOAP request example

Client invocation of the `getDistributionList` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <getDistributionList xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c698</identifier>
    </getDistributionList>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getDistributionList` 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>
    <getDistributionListResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:distributionList enabled="true" name="analysts"
          xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c698</ns1:identifier>
      </ns1:distributionList>
    </getDistributionListResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getDistributionLists operation

Retrieves all distribution lists available in the system.

Return information

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

Table 49. Return Value	
Type	Description
distributionList[]	Represents a logical grouping of the individual subscribers. For internal use only. This is reserved for future use.

Java example

The information available in the `DistributionList` objects returned by the `getDistributionLists` operation includes the following:

- The list name
- The identifier for the list

The following example iterates through the returned objects, using the `getName` and `getIdentifier` methods to access the distribution list characteristics.

```
DistributionList[] dList = stub.getDistributionLists();
for (int j = 0; j < dList.length; j++) {
    System.out.println("Distribution list " + dList[j].getName() +
        " has an identifier of " dList[j].getIdentifier() + ".");
}
```

SOAP request example

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

SOAP response example

The server responds to a `getDistributionLists` 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>
    <getDistributionListsResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:distributionList enabled="true" name="managers"
          xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c68e</ns1:identifier>
      </ns1:distributionList>
      <ns2:distributionList enabled="true" name="analysts"
          xmlns:ns2="http://xml.spss.com/notification">
        <ns2:identifier>0a0a4aac0161f10f0000010d8e63c979c698</ns2:identifier>
      </ns2:distributionList>
    </getDistributionListsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getEventType operation

Returns the event type for a specified event type identifier.

Input fields

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

Table 50. Fields for getEventType

Field	Type/Valid Values	Description
identifier	string	An identifier for an event type.

Return information

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

Table 51. Return Value

Type	Description
eventType	Identifies a type of the event within its notification domain .

Java example

The information available in the `EventType` object returned by the `getEventType` operation includes the following:

- Domain name
- Event type name
- Event type properties
- Identifier for the event type

The following example uses accessor methods to retrieve the individual event type characteristics.

```
String typeIdentifier = "0a0a483800a83a1300000102652a7356800d";
EventType eventType = stub.getEventType(typeIdentifier);
System.out.println("Identifier: " + eventType.getIdentifier());
System.out.println("Domain: " + eventType.getDomainName());
System.out.println("Type name: " + eventType.getTypeName());
```

SOAP request example

Client invocation of the `getEventType` 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>
    <getEventType xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a483800a83a1300000102652a7356800d</identifier>
    </getEventType>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getEventType` 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>
    <getEventTypeResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:eventType domainName="PRMS" typeName="Completion"
          xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a483800a83a1300000102652a7356800d</ns1:identifier>
        <ns1:property descriptionKey="$$prms/etr_job_success_desc"
            labelKey="$$prms/etr_job_success"
            description="The success or failure of the job." label="Job Success"
            typeCode="boolean" name="JobSuccess">
          <ns1:identifier>0a0a483800a83a1300000102652a73568011</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$prms/etr_job_status_url_desc"
            labelKey="$$prms/etr_job_status_url" description="The job status URL."
            label="Job Status URL" typeCode="string" name="JobStatusURL">
          <ns1:identifier>0000000000000000000000000000000000269</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$prms/etr_job_start_desc"
            labelKey="$$prms/etr_job_start" description="The start of the job."
            label="Job Start" typeCode="timestamp" name="JobStart">
          <ns1:identifier>0000000000000000000000000000000000267</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$prms/etr_job_id_desc" labelKey="$$prms/etr_job_id"
            description="The job identifier." label="Job ID" typeCode="string" name="JobID">
          <ns1:identifier>0a0a483800a83a1300000102652a7356800f</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$prms/etr_job_end_desc" labelKey="$$prms/etr_job_end"
            description="The end of the job." label="Job End" typeCode="timestamp"
            name="JobEnd">
          <ns1:identifier>0000000000000000000000000000000000268</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$prms/etr_job_name_desc"
            labelKey="$$prms/etr_job_name" description="The name of the job."
            label="Job Name" typeCode="string" name="JobName">
          <ns1:identifier>0000000000000000000000000000000000266</ns1:identifier>
        </ns1:property>
      </ns1:eventType>
    </getEventTypeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getEventTypes operation

Returns all event types for a specified notification domain.

Input fields

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

Table 52. Fields for getEventTypes		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for a notification domain.

Return information

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

Table 53. Return Value	
Type	**Description**
eventType[]	Identifies a type of the event within its notification domain .

Java example

The information available in the `EventType` objects returned by the `getEventTypes` operation includes the following:

- Domain name
- Event type name
- Event type properties
- Identifier for the event type

The following example iterates through the returned objects, using accessor methods to retrieve the individual event type characteristics.

```
String domainIdentifier = "0a0a483800a83a1300000102652a7356800a";
EventType[] eventType = stub.getEventTypes(domainIdentifier);
for (int j = 0; j < eventType.length; j++) {
    System.out.println("Identifier: " + eventType[j].getIdentifier());
    System.out.println("Domain: " + eventType[j].getDomainName());
    System.out.println("Type name: " + eventType[j].getTypeName());
    PropertyValue[] propValue = eventType[j].getPropertyValue();
    for (int k = 0; k < propValue.length; k++) {
        System.out.println("Property: " + propValue[k].getName() +
            " (type = " + propValue[k].getTypeCode() + ") = " +
            propValue[k].getDescription);
    }
    System.out.println();
}
```

SOAP request example

Client invocation of the `getEventTypes` 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>
    <getEventTypes xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a483800a83a1300000102652a7356800a</identifier>
    </getEventTypes>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getEventTypes` 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>
    <getEventTypesResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:eventType domainName="PRMS" typeName="JobStepCompletion"
          xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>000000000000000000000000000000000259</ns1:identifier>
        <ns1:property descriptionKey="$$prms/etr_job_step_artifacts_desc"
            labelKey="$$prms/etr_job_step_artifacts"
            description="The array of the URLs for the artifacts generated by the job step."
            label="Job Step Artifacts" typeCode="array" name="JobStepArtifacts">
          <ns1:identifier>00000000000000000000000000000000026a</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$prms/etr_job_step_success_desc"
            labelKey="$$prms/etr_job_step_success"
            description="The job step success or failure." label="Job Step Success"
            typeCode="boolean" name="JobStepSuccess">
          <ns1:identifier>00000000000000000000000000000000025b</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$prms/etr_job_step_start_desc"
            labelKey="$$prms/etr_job_step_start" description="The start of a job step."
            label="Job Step Start" typeCode="timestamp" name="JobStepStart">
          <ns1:identifier>000000000000000000000000000000000262</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$prms/etr_job_step_job_name_desc"
            labelKey="$$prms/etr_job_step_job_name"
            description="The job name for this job step." label="Job Name" typeCode="string"
            name="JobName">
          <ns1:identifier>000000000000000000000000000000000264</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$prms/etr_job_step_status_url_desc"
```

```
                labelKey="$$prms/etr_job_step_status_url"
                description="The URL for the job step status." label="Job Step Status URL"
                typeCode="string" name="JobStepStatusURL">
             <ns1:identifier>0000000000000000000000000000000000000265</ns1:identifier>
          </ns1:property>
          <ns1:property descriptionKey="$$prms/etr_job_step_id_desc"
                labelKey="$$prms/etr_job_step_id" description="The job step identifier."
                label="Job Step ID" typeCode="string" name="JobStepID">
             <ns1:identifier>000000000000000000000000000000000000025a</ns1:identifier>
          </ns1:property>
          <ns1:property descriptionKey="$$prms/etr_job_step_attachments_desc"
                labelKey="$$prms/etr_job_step_attachments"
                description="If true the generated artifacts published to the repository should
                  be sent as binary attachments."
                label="Attachments" typeCode="boolean" name="Attachments">
             <ns1:identifier>0000000000000000000000000000000000000260</ns1:identifier>
          </ns1:property>
          <ns1:property descriptionKey="$$prms/etr_job_step_name_desc"
                labelKey="$$prms/etr_job_step_name" description="The name of the job step."
                label="Job Step Name" typeCode="string" name="JobStepName">
             <ns1:identifier>0000000000000000000000000000000000000261</ns1:identifier>
          </ns1:property>
          <ns1:property descriptionKey="$$prms/etr_job_step_end_desc"
                labelKey="$$prms/etr_job_step_end" description="The end of the job step."
                label="Job End" typeCode="timestamp" name="JobStepEnd">
             <ns1:identifier>0000000000000000000000000000000000000263</ns1:identifier>
          </ns1:property>
       </ns1:eventType>
       <ns2:eventType domainName="PRMS" typeName="Completion"
             xmlns:ns2="http://xml.spss.com/notification">
          <ns2:identifier>0a0a483800a83a1300000102652a7356800d</ns2:identifier>
          <ns2:property descriptionKey="$$prms/etr_job_success_desc"
                labelKey="$$prms/etr_job_success"
                description="The success or failure of the job." label="Job Success"
                typeCode="boolean" name="JobSuccess">
             <ns2:identifier>0a0a483800a83a1300000102652a73568011</ns2:identifier>
          </ns2:property>
          <ns2:property descriptionKey="$$prms/etr_job_status_url_desc"
                labelKey="$$prms/etr_job_status_url" description="The job status URL."
                label="Job Status URL" typeCode="string" name="JobStatusURL">
             <ns2:identifier>0000000000000000000000000000000000000269</ns2:identifier>
          </ns2:property>
          <ns2:property descriptionKey="$$prms/etr_job_start_desc"
                labelKey="$$prms/etr_job_start" description="The start of the job."
                label="Job Start" typeCode="timestamp" name="JobStart">
             <ns2:identifier>0000000000000000000000000000000000000267</ns2:identifier>
          </ns2:property>
          <ns2:property descriptionKey="$$prms/etr_job_id_desc" labelKey="$$prms/etr_job_id"
                description="The job identifier." label="Job ID" typeCode="string" name="JobID">
             <ns2:identifier>0a0a483800a83a1300000102652a7356800f</ns2:identifier>
          </ns2:property>
          <ns2:property descriptionKey="$$prms/etr_job_end_desc" labelKey="$$prms/etr_job_end"
                description="The end of the job." label="Job End" typeCode="timestamp"
                name="JobEnd">
             <ns2:identifier>0000000000000000000000000000000000000268</ns2:identifier>
          </ns2:property>
          <ns2:property descriptionKey="$$prms/etr_job_name_desc"
                labelKey="$$prms/etr_job_name" description="The name of the job."
                label="Job Name" typeCode="string" name="JobName">
             <ns2:identifier>0000000000000000000000000000000000000266</ns2:identifier>
          </ns2:property>
       </ns2:eventType>
      </getEventTypesResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

The getNotificationDomain operation

Retrieves the notification domain corresponding to the specified identifier.

Input fields

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

Table 54. Fields for getNotificationDomain		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for a notification domain.

Return information

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

Table 55. Return Value	
Type	**Description**
notificationDomain	Identifies the particular vertical industry domain or application in which the event type is defined.

Java example

The information available in the `NotificationDomain` object returned by the `getNotificationDomain` operation includes the following:

- The domain name
- The identifier for the domain

The following example uses the `getName` and `getIdentifier` methods to access the notification domain characteristics.

```
String domainIdentifier = "0a0a483800a83a1300000102652a7356800a";
NotificationDomain nDomain = stub.getNotificationDomain(domainIdentifier);
System.out.println("Identifier " + nDomain.getIdentifier() +
    " corresponds to notification domain " + nDomain.getName());
```

SOAP request example

Client invocation of the `getNotificationDomain` 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>
    <getNotificationDomain xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a483800a83a1300000102652a7356800a</identifier>
    </getNotificationDomain>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getNotificationDomain` 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>
    <getNotificationDomainResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:notificationDomain name="PRMS" xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a483800a83a1300000102652a7356800a</ns1:identifier>
      </ns1:notificationDomain>
    </getNotificationDomainResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getNotificationDomains operation

Retrieves all notification domains available in the system.

Return information

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

Table 56. Return Value

Type	Description
notificationDomain[]	Identifies the particular vertical industry domain or application in which the event type is defined.

Java example

The information available in the `NotificationDomain` objects returned by the `getNotificationDomains` operation includes the following:

- The domain name
- The identifier for the domain

The following example iterates through the returned objects, using the `getName` and `getIdentifier` methods to access the notification domain characteristics.

```
NotificationDomain[] nDomain = stub.getNotificationDomains();
for (int j = 0; j < nDomain.length; j++) {
    System.out.println("Notification domain " + nDomain[j].getName() +
        " has an identifier of " nDomain[j].getIdentifier() + ".");
}
```

SOAP request example

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

SOAP response example

The server responds to a `getNotificationDomains` 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>
    <getNotificationDomainsResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:notificationDomain name="Repository" xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>00000000000000000000000000000001f4</ns1:identifier>
      </ns1:notificationDomain>
      <ns2:notificationDomain name="PRMS" xmlns:ns2="http://xml.spss.com/notification">
        <ns2:identifier>0a0a483800a83a1300000102652a7356800a</ns2:identifier>
      </ns2:notificationDomain>
    </getNotificationDomainsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getObjects operation

Retrieves notification objects from the repository that match a specified criterion. Available criteria limit the retrieval to one of the following object types:

- Distribution lists
- Event types
- Notification providers
- Subscribables
- Subscribers
- Subscriber specifiers
- Subscriptions
- Subscription selectors

To limit the returned objects to a subset of all available objects matching the criterion type, define values for specific criterion characteristics. For example, the criterion could limit the returned set to all multicasted subscriptions or to a subscriber having a specific principal ID.

Input fields

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

Table 57. Fields for getObjects		
Field	**Type/Valid Values**	**Description**
querySpecification	querySpecification	A query specification for the notification service.

Return information

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

Table 58. Return Value	
Type	**Description**
queryResult	Results of the query for the notification service.

Java example

To retrieve notification objects from the repository:

1. Create a criterion object for the type of object to be retrieved.
2. Define specific properties of the criterion.
3. Assign the criterion object to a `QuerySpecification` object using the appropriate set method.
4. Supply the `getObjects` operation with the query specification.

The following example retrieves all repository items to which a subscriber can subscribe, sending the item identifiers and types to the console.

```
SubscribableCriterion subscribableCrit = new SubscribableCriterion();

QuerySpecification querySpec = new QuerySpecification();
querySpec.setSubscribableCriterion(subscribableCrit);
QueryResult result = stub.getObjects(querySpec);

IdentificationSpecifier[] idSpecifier = result.getIdentificationSpecifier();
for (int j = 0; j < idSpecifier.length; j++) {
    System.out.println("Identifier: " + idSpecifier[j].getIdentifier());
    System.out.println("Type: " + idSpecifier[j].getObjectType());
```

```
    System.out.println();
    }
```

SOAP request example

Client invocation of the `getObjects` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <getObjects xmlns="http://xml.spss.com/notification/remote">
      <ns2:querySpecification xmlns:ns2="http://xml.spss.com/notification">
        <ns2:subscribableCriterion/>
      </ns2:querySpecification>
    </getObjects>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getObjects` 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>
    <getObjectsResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:queryResult xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identificationSpecifier identifier="0a0a4aac43d009f10000010dd13e0acd8013"
            objectType="HierarchicalContent.Folder"/>
        <ns1:identificationSpecifier identifier="0a0a4aac43d009f10000010dd13e0acd8292"
            objectType="HierarchicalContent.File"/>
        <ns1:identificationSpecifier identifier="0a0a4aac43d009f10000010dd13e0acd8534"
            objectType="ProcessManagement.EventCluster"/>
        <ns1:identificationSpecifier identifier="0a0a4aac43d009f10000010dd13e0acd8668"
            objectType="HierarchicalContent.File"/>
        <ns1:identificationSpecifier identifier="0a0a4aacfe9747c10000010dd18630f58823"
            objectType="ProcessManagement.WorkEvent"/>
        <ns1:identificationSpecifier identifier="0a0a4aacfe9747c10000010dd18630f58934"
            objectType="HierarchicalContent.File"/>
        <ns1:identificationSpecifier identifier="0a0a4aacfe9747c10000010dd18630f58a54"
            objectType="ProcessManagement.WorkEvent"/>
      </ns1:queryResult>
    </getObjectsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getParentEventType operation

Returns the event type (if any) from which a specified event type is derived.

Input fields

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

Table 59. Fields for getParentEventType		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for an event type.

Return information

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

Table 60. Return Value	
Type	**Description**
eventType	Identifies a type of the event within its notification domain .

Java example

The information available in the `EventType` object returned by the `getParentEventType` operation includes the following:

- Domain name
- Event type name
- Event type properties
- Identifier for the event type

The following example uses accessor methods to retrieve the individual event type characteristics.

```
String typeIdentifier = "00000000000000000000000000000001ff";
EventType eventType = stub.getParentEventType(typeIdentifier);
System.out.println("Identifier: " + eventType.getIdentifier());
System.out.println("Domain: " + eventType.getDomainName());
System.out.println("Type name: " + eventType.getTypeName());
```

SOAP request example

Client invocation of the `getParentEventType` 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>
    <getParentEventType xmlns="http://xml.spss.com/notification/remote">
      <identifier>00000000000000000000000000000001ff</identifier>
    </getParentEventType>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getParentEventType` 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>
    <getParentEventTypeResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:eventType domainName="Repository" typeName="ResourceEvent"
          xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>00000000000000000000000000000001f5</ns1:identifier>
        <ns1:property descriptionKey="$$repository/etr_path_ids_desc"
            labelKey="$$repository/etr_path_ids_name"
            description="The path identifiers of the resource separated by colons."
            label="Path Identifiers" typeCode="string" name="PathIdentifiers">
          <ns1:identifier>00000000000000000000000000000000207</ns1:identifier>
        </ns1:property>
        <ns1:property descriptionKey="$$repository/etr_resource_spsscr_desc"
```

```
        labelKey="$$repository/etr_resource_spsscr_url"
        description="The SPSSCR URL of the resource." label="SPSSCR URL"
        typeCode="string" name="ResourceSpssCrUrl">
      <ns1:identifier>000000000000000000000000000000000020a</ns1:identifier>
    </ns1:property>
    <ns1:property descriptionKey="$$repository/etr_resource_path_desc"
        labelKey="$$repository/etr_resource_path"
        description="The path of the resource in the repository." label="Resource Path"
        typeCode="string" name="ResourcePath">
      <ns1:identifier>0000000000000000000000000000000000000205</ns1:identifier>
    </ns1:property>
    <ns1:property descriptionKey="$$repository/etr_resource_id_desc"
        labelKey="$$repository/etr_resource_id"
        description="The identifier of the subscribable resource." label="Resource ID"
        typeCode="string" name="ResourceID">
      <ns1:identifier>00000000000000000000000000000000000001f6</ns1:identifier>
    </ns1:property>
    <ns1:property descriptionKey="$$repository/etr_action_type_desc"
        labelKey="$$repository/etr_action_type"
        description="The type of the user action, for example, new file created,
            new version of the file created and so on."
        label="Action Type" typeCode="string" name="ActionType">
      <ns1:identifier>0000000000000000000000000000000000000204</ns1:identifier>
    </ns1:property>
    <ns1:property descriptionKey="$$repository/etr_resource_http_url_desc"
        labelKey="$$repository/etr_resource_http_url"
        description="The HTTP URL of the resource." label="Resource HTTP"
        typeCode="string" name="ResourceHttpUrl">
      <ns1:identifier>0000000000000000000000000000000000000209</ns1:identifier>
    </ns1:property>
    <ns1:property descriptionKey="$$repository/etr_resource_name_desc"
        labelKey="$$repository/etr_resource_name"
        description="The name of the resource." label="Resource Name" typeCode="string"
        name="ResourceName">
      <ns1:identifier>0000000000000000000000000000000000000206</ns1:identifier>
    </ns1:property>
    </ns1:eventType>
    </getParentEventTypeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getSubscriber operation

Retrieves the subscriber corresponding to the specified identifier.

Input fields

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

Table 61. Fields for getSubscriber		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for a subscriber.

Return information

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

Table 62. Return Value	
Type	**Description**
subscriber	Represents an individual subscriber.

Java example

The information available in the `Subscriber` object returned by the `getSubscriber` operation includes the following:

- Principal ID for the subscriber
- Subscriber identifier

The following example uses the `getPrincipalID` and `getIdentifier` methods to access the subscriber characteristics.

```
String subscriberIdentifier = "0a0a4aac0161f10f0000010d8e63c979c09f";
Subscriber subscriber = stub.getSubscriber(subscriberIdentifier);
```

```
System.out.println("Identifier " + subscriber.getIdentifier() +
    " corresponds to subscriber " + subscriber.getName());
```

SOAP request example

Client invocation of the `getSubscriber` 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>
    <getSubscriber xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c09f</identifier>
    </getSubscriber>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getSubscriber` 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>
    <getSubscriberResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:subscriber enabled="true" principalID="//u/smtp/jjones@spss.com"
          xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c09f</ns1:identifier>
      </ns1:subscriber>
    </getSubscriberResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getSubscribers operation

Retrieves all subscribers available in the system.

Return information

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

Table 63. Return Value	
Type	**Description**
subscriber[]	Represents an individual subscriber.

Java example

The information available in the `Subscriber` objects returned by the `getSubscribers` operation includes the following:

- Principal ID for the subscriber
- Subscriber identifier

The following example iterates through the returned objects, using the `getPrincipalID` and `getIdentifier` methods to access the subscriber characteristics.

```
Subscriber[] subscriber = stub.getSubscribers();
for (int j = 0; j < subscriber.length; j++) {
   System.out.println("Subscriber " + subscriber[j].getPrincipalID() +
      " has an identifier of " subscriber[j].getIdentifier() + ".");
}
```

SOAP request example

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

SOAP response example

The server responds to a `getSubscribers` 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>
    <getSubscribersResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:subscriber enabled="true" principalID="//u/smtp/kkreuter@spss.com"
         xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a4aac0161f10f0000010d8e63c9798f11</ns1:identifier>
      </ns1:subscriber>
      <ns2:subscriber enabled="true" principalID="//uNative//admin"
         xmlns:ns2="http://xml.spss.com/notification">
        <ns2:identifier>0a0a4aac0161f10f0000010d8e63c9799d9e</ns2:identifier>
      </ns2:subscriber>
      <ns3:subscriber enabled="true" principalID="//u/smtp/jjones@spss.com"
         xmlns:ns3="http://xml.spss.com/notification">
        <ns3:identifier>0a0a4aac0161f10f0000010d8e63c979c09f</ns3:identifier>
      </ns3:subscriber>
      <ns4:subscriber enabled="true" principalID="//u/smtp/bbrever@yahoo.com"
         xmlns:ns4="http://xml.spss.com/notification">
        <ns4:identifier>0a0a4aac0161f10f0000010d8e63c979c0d6</ns4:identifier>
      </ns4:subscriber>
    </getSubscribersResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getSubscription operation

Returns the subscription for a specified subscription identifier.

Input fields

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

Table 64. Fields for getSubscription		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for a subscription.

Return information

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

Table 65. Return Value

Type	Description
subscription	Represents a single subscription.

Java example

The information available in the `Subscription` object returned by the `getSubscription` operation includes the following:

- Principal ID of the subscriber
- Boolean indicator of whether or not the subscription is multicasted
- Boolean indicator of whether or not the subscription is scheduled
- Subscription selector identifier for the subscription
- Values for the subscription properties
- Subscription identifier

The following example uses a variety of accessor methods to retrieve the individual subscription characteristics.

```
String subscriptionIdentifier = "0a0a4aac0161f10f0000010d8e63c979c098";
Subscription subscription = stub.getSubscription(subscriptionIdentifier);
System.out.println("Identifier: " + subscription.getIdentifier());
System.out.println("Principal ID: " + subscription.getPrincipalID());
if (subscription.isMulticasted()) {
   System.out.println("Multicasted");
}
if (subscription.isScheduled()) {
   System.out.println("Scheduled");
} else {
   System.out.println("Not scheduled");
}
System.out.println("Selector Identifier: " +
   subscription.getSubscriptionSelectorIdentifier());
PropertyValue[] propValue = subscription[j].getPropertyValue();
for (int k = 0; k < propValue.length; k++) {
   System.out.println("Property " + propValue[k].getName() +
   " (type = " + propValue[k].getTypeCode() +
   " ) has a value of" + propValue[k].getValue() + ".");
}
System.out.println();
```

SOAP request example

Client invocation of the `getSubscription` 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>
    <getSubscription xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c098</identifier>
    </getSubscription>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getSubscription` 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>
    <getSubscriptionResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:subscription subscriptionSelectorIdentifier="0a0a483800a83a1300000102652a73568013"
          enabled="true" scheduled="false" multicasted="true" principalID="//uNative//admin"
          xmlns:ns1="http://xml.spss.com/notification">
      <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c098</ns1:identifier>
      <ns1:propertyValue typeCode="boolean" name="JobSuccess$0">
        <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c09a</ns1:identifier>
        <ns1:value>true</ns1:value>
      </ns1:propertyValue>
      <ns1:propertyValue typeCode="string" name="JobID$0">
        <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c099</ns1:identifier>
        <ns1:value>0a0a4aac0161f10f0000010d8e63c9798ebd</ns1:value>
      </ns1:propertyValue>
      </ns1:subscription>
    </getSubscriptionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getSubscriptions operation

Returns subscriptions for a specified subscriber.

Input fields

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

Table 66. Fields for getSubscriptions		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for a subscriber.

Return information

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

Table 67. Return Value	
Type	**Description**
subscription[]	Represents a single subscription.

Java example

The information available in the `Subscription` objects returned by the `getSubscriptions` operation includes the following:

- Principal ID of the subscriber
- Boolean indicator of whether or not the subscription is multicasted
- Boolean indicator of whether or not the subscription is scheduled
- Subscription selector identifier for the subscription
- Values for the subscription properties
- Subscription identifier

The following example iterates through the returned objects, using accessor methods to retrieve the individual subscription characteristics.

```
String subscriberIdentifier = "0a0a4aac0161f10f0000010d8e63c979c09f";
Subscription[] subscription = stub.getSubscriptions(subscriberIdentifier);
```

```
    for (int j = 0; j < subscription.length; j++) {
        System.out.println("Identifier: " + subscription[j].getIdentifier());
        System.out.println("Principal ID: " + subscription[j].getPrincipalID());
        if (subscription[j].isMulticasted()) {
            System.out.println("Multicasted");
        }
        if (subscription[j].isScheduled()) {
            System.out.println("Scheduled");
        } else {
            System.out.println("Not scheduled");
        }
        System.out.println("Selector Identifier: " +
            subscription[j].getSubscriptionSelectorIdentifier());
        PropertyValue[] propValue = subscription[j].getPropertyValue();
        for (int k = 0; k < propValue.length; k++) {
            System.out.println("Property " + propValue[k].getName() +
                " (type = " + propValue[k].getTypeCode() +
                ") has a value of" + propValue[k].getValue() + ".");
        }
        System.out.println();
    }
```

SOAP request example

Client invocation of the `getSubscriptions` 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>
    <getSubscriptions xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a4aac0161f10f0000010d8e63c979c09f</identifier>
    </getSubscriptions>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getSubscriptions` 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>
    <getSubscriptionsResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:subscription subscriptionSelectorIdentifier="0a0a483800a83a1300000102652a73568013"
          enabled="true" scheduled="false" multicasted="true" principalID="//uNative//validUser"
          xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c098</ns1:identifier>
        <ns1:propertyValue typeCode="boolean" name="JobSuccess$0">
          <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c09a</ns1:identifier>
          <ns1:value>true</ns1:value>
        </ns1:propertyValue>
        <ns1:propertyValue typeCode="string" name="JobID$0">
          <ns1:identifier>0a0a4aac0161f10f0000010d8e63c979c099</ns1:identifier>
          <ns1:value>0a0a4aac0161f10f0000010d8e63c9798ebd</ns1:value>
        </ns1:propertyValue>
      </ns1:subscription>
    </getSubscriptionsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getSubscriptionSelector operation

Retrieves the subscription selector for a specified selector identifier.

Input fields

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

Table 68. Fields for getSubscriptionSelector		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for a subscription selector.

Return information

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

Table 69. Return Value	
Type	**Description**
subscriptionSelector	Defines a subscription selector for the given event type.

Java example

The information available in the `SubscriptionSelector` object returned by the `getSubscriptionSelector` operation includes the following:

- Selector name
- Filter expression
- Compiled filter expression
- Selector identifier

The following example uses various accessor methods to retrieve the individual subscription selector characteristics.

```
String selectorIdentifier = "0a0a483800a83a1300000102652a73568013";
SubscriptionSelector selector = stub.getSubscriptionSelector(selectorIdentifier);
System.out.println("Identifier: " + selector.getIdentifier());
System.out.println("Name: " + selector.getName());
System.out.println("Filter expression: " + selector.getFilterExpression());
System.out.println("Compiled filter expression: " +
    selector.getCompiledFilterExpression());
System.out.println();
```

SOAP request example

Client invocation of the `getSubscriptionSelector` 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>
    <getSubscriptionSelector xmlns="http://xml.spss.com/notification/remote">
      <identifier>0a0a483800a83a1300000102652a73568013</identifier>
    </getSubscriptionSelector>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getSubscriptionSelector` 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>
    <getSubscriptionSelectorResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:subscriptionSelector
          filterExpression="JobID=='JobID' &amp;&amp; JobSuccess==true"
          compiledFilterExpression="JobID$0=='${JobID}' &amp;&amp; JobSuccess$0==${JobSuccess}"
          name="prms_jobid_completion" xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>0a0a483800a83a1300000102652a73568013</ns1:identifier>
      </ns1:subscriptionSelector>
    </getSubscriptionSelectorResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The getSubscriptionSelectors operation

Retrieves all available subscription selectors for a specified event type. In no event type is specified, the operation returns all subscription selectors in the system.

Input fields

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

Table 70. Fields for getSubscriptionSelectors		
Field	**Type/Valid Values**	**Description**
identifier	string	An identifier for an event type.

Return information

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

Table 71. Return Value	
Type	**Description**
subscriptionSelector[]	Defines a subscription selector for the given event type.

Java example

The information available in the `SubscriptionSelector` objects returned by the `getSubscriptionSelectors` operation includes the following:

- Selector name
- Filter expression
- Compiled filter expression
- Selector identifier

The following example iterates through the returned objects, using accessor methods to retrieve the individual subscription selector characteristics.

```
SubscriptionSelector[] selector = stub.getSubscriptionSelectors();
for (int j = 0; j < selector.length; j++) {
    System.out.println("Identifier: " + selector[j].getIdentifier());
    System.out.println("Name: " + selector[j].getName());
    System.out.println("Filter expression: " + selector[j].getFilterExpression());
    System.out.println("Compiled filter expression: " +
        selector[j].getCompiledFilterExpression());
    System.out.println();
}
```

SOAP request example

Client invocation of the `getSubscriptionSelectors` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <getSubscriptionSelectors xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `getSubscriptionSelectors` 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>
    <getSubscriptionSelectorsResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:subscriptionSelector
          filterExpression="ResourceID rlike '${ResourceID}' &amp;&amp; Attachments==false
          &amp;&amp; MimeType like '${MimeType}'"
          compiledFilterExpression="ResourceID$0 rlike '${ResourceID}'
          &amp;&amp; Attachments$0==${Attachments} &amp;&amp; MimeType$0 like '${MimeType}'"
          name="selector_folder_link" xmlns:ns1="http://xml.spss.com/notification">
        <ns1:identifier>00000000000000000000000000000001fb</ns1:identifier>
      </ns1:subscriptionSelector>
      <ns2:subscriptionSelector
          filterExpression="ResourceID rlike '${ResourceID}' &amp;&amp; Attachments==true
          &amp;&amp; MimeType like '${MimeType}'"
          compiledFilterExpression="ResourceID$0 rlike '${ResourceID}'
          &amp;&amp; Attachments$0==${Attachments} &amp;&amp; MimeType$0 like '${MimeType}'"
          name="selector_folder_attachment" xmlns:ns2="http://xml.spss.com/notification">
        <ns2:identifier>00000000000000000000000000000001fd</ns2:identifier>
      </ns2:subscriptionSelector>
      <ns3:subscriptionSelector
          filterExpression="ResourceID=='${ResourceID}' &amp;&amp; Attachments==false"
          compiledFilterExpression="ResourceID$0=='${ResourceID}'
          &amp;&amp; Attachments$0==${Attachments}"
          name="selector_file_link" xmlns:ns3="http://xml.spss.com/notification">
        <ns3:identifier>00000000000000000000000000000000200</ns3:identifier>
      </ns3:subscriptionSelector>
      <ns4:subscriptionSelector
          filterExpression="ResourceID=='${ResourceID}' &amp;&amp; Attachments==true"
          compiledFilterExpression="ResourceID$0=='${ResourceID}'
          &amp;&amp; Attachments$0==${Attachments}"
          name="selector_file_attachment" xmlns:ns4="http://xml.spss.com/notification">
        <ns4:identifier>00000000000000000000000000000000202</ns4:identifier>
      </ns4:subscriptionSelector>
      <ns5:subscriptionSelector filterExpression="ResourceID rlike '${ResourceID}'"
          compiledFilterExpression="ResourceID$0 rlike '${ResourceID}'" name="selector_folder"
          xmlns:ns5="http://xml.spss.com/notification">
        <ns5:identifier>00000000000000000000000000000000208</ns5:identifier>
      </ns5:subscriptionSelector>
      <ns6:subscriptionSelector
          filterExpression="JobStepID=='JobStepID' &amp;&amp; JobStepSuccess==true
          &amp;&amp; Attachments=true"
          compiledFilterExpression="JobStepID$0=='${JobStepID}'
          &amp;&amp; JobStepSuccess$0==${JobStepSuccess} &amp;&amp; Attachments$0==${Attachments}"
          name="prms_jobstep_completion_success" xmlns:ns6="http://xml.spss.com/notification">
        <ns6:identifier>0000000000000000000000000000000025c</ns6:identifier>
      </ns6:subscriptionSelector>
      <ns7:subscriptionSelector
          filterExpression="JobStepID=='JobStepID' &amp;&amp; JobStepSuccess==false
          &amp;&amp; Attachments=true"
          compiledFilterExpression="JobStepID$0=='${JobStepID}'
          &amp;&amp; JobStepSuccess$0==${JobStepSuccess} &amp;&amp; Attachments$0==${Attachments}"
          name="prms_jobstep_completion_failure" xmlns:ns7="http://xml.spss.com/notification">
        <ns7:identifier>0000000000000000000000000000000025e</ns7:identifier>
      </ns7:subscriptionSelector>
      <ns8:subscriptionSelector
          filterExpression="JobID=='JobID' &amp;&amp; JobSuccess==true"
          compiledFilterExpression="JobID$0=='${JobID}' &amp;&amp; JobSuccess$0==${JobSuccess}"
          name="prms_jobid_completion" xmlns:ns8="http://xml.spss.com/notification">
        <ns8:identifier>0a0a483800a83a1300000102652a73568013</ns8:identifier>
      </ns8:subscriptionSelector>
      <ns9:subscriptionSelector filterExpression="JobSuccess==true"
          compiledFilterExpression="JobSuccess$0==${JobSuccess}" name="prms_completion"
          xmlns:ns9="http://xml.spss.com/notification">
        <ns9:identifier>0a0a483800a83a1300000102652a73568015</ns9:identifier>
```

```
        </ns9:subscriptionSelector>
      </getSubscriptionSelectorsResponse>
    </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 72. 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/notification/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/notification/remote">
      <version>4.20.000</version>
    </getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The removeSubscriberFromDistributionList operation

Removes a subscriber from a specified distribution list.

Input fields

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

Table 73. Fields for removeSubscriberFromDistributionList		
Field	**Type/Valid Values**	**Description**
subscriberIdentifier	string	An identifier for a subscriber.
distributionListIdentifier	string	Identifier for the distribution list.

Java example

To remove a subscriber from a distribution list, supply the `removeSubscriberFromDistributionList` operation with identifiers for the subscriber and the distribution list.

```
String distributionListIdentifier = "0a0a4aac0161f10f0000010d8e63c979c698";
String subscriberIdentifier = "0a0a4aac0161f10f0000010d8e63c979c09f";
stub.removeSubscriberFromDistributionList(subscriberIdentifier, distributionListIdentifier);
```

SOAP request example

Client invocation of the `removeSubscriberFromDistributionList` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <removeSubscriberFromDistributionList xmlns="http://xml.spss.com/notification/remote">
      <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c09f</subscriberIdentifier>
      <distributionListIdentifier>0a0a4aac0161f10f0000010d8e63c979c698</distributionListIdentifier>
    </removeSubscriberFromDistributionList>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `removeSubscriberFromDistributionList` 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>
    <removeSubscriberFromDistributionListResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The removeSubscribersFromDistributionList operation

Removes one or more subscribers from a specified distribution list.

Input fields

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

Table 74. Fields for removeSubscribersFromDistributionList

Field	Type/Valid Values	Description
subscriberIdentifier	string[]	Identifiers for subscribers.
distributionListIdentifier	string	Identifier for the distribution list.

Java example

To remove one or more subscribers from a distribution list, supply the `removeSubscriberFromDistributionList` operation with identifiers for the subscribers and the distribution list.

```
String distributionListIdentifier = "0a0a4aacfe9747c10000010dd18630f581cb";
String subscriberIdentifier = "0a0a4aacfe9747c10000010dd18630f581f5";
stub.removeSubscriberFromDistributionList(subscriberIdentifier, distributionListIdentifier);
```

SOAP request example

Client invocation of the `removeSubscribersFromDistributionList` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <removeSubscribersFromDistributionList xmlns="http://xml.spss.com/notification/remote">
      <subscriberIdentifier>0a0a4aacfe9747c10000010dd18630f581f5</subscriberIdentifier>
      <distributionListIdentifier>0a0a4aacfe9747c10000010dd18630f581cb</distributionListIdentifier>
    </removeSubscribersFromDistributionList>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `removeSubscribersFromDistributionList` 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>
    <removeSubscribersFromDistributionListResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The setSubscriber operation

Adds a delivery device for a subscriber to the system. If the specified subscriber does not currently exist, the operation creates one.

Input fields

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

Table 75. Fields for setSubscriber		
Field	**Type/Valid Values**	**Description**
principal	string	Principal ID for the subscriber.
address	string	Fully qualified recipient address for the specified protocol.
protocolType	protocolType	Enumerates delivery protocols.

Java example

To add a delivery device for a subscriber, first determine the principal ID for the subscriber. Available principal IDs can be accessed using the Directory Information Service. Supply the `setSubscriber` operation with the principal ID, the address associated with the ID, and the protocol type.

```
String principalID = "//uNative//validUser";
String address = "pesUser@company.com";
stub.setSubscriber(principalID, address, ProtocolType.SMTP);
```

SOAP request example

Client invocation of the `setSubscriber` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <setSubscriber xmlns="http://xml.spss.com/notification/remote">
      <principal>//uNative//validUser</principal>
      <address>pesUser@company.com</address>
      <protocolType xmlns="http://xml.spss.com/notification">smtp</protocolType>
    </setSubscriber>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `setSubscriber` 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>
    <setSubscriberResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The subscribe operation

Adds a subscriber to a subscription. Both the subscriber and the subscription must exist in the system.

Input fields

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

Table 76. Fields for subscribe		
Field	Type/Valid Values	Description
subscriberIdentifier	string	An identifier for a subscriber.
subscriptionIdentifier	string	An identifier for a subscription.

Java example

To add a subscriber to a subscription, supply the `subscribe` operation with strings corresponding to the identifiers for both items.

```
String subscriberIdentifier = "0a0a4aac0161f10f0000010d8e63c979c09f";
String subscriptionIdentifier = "0a0a4aac0161f10f0000010d8e63c979c098";
stub.subscribe(subscriberIdentifier, subscriptionIdentifier);
```

SOAP request example

Client invocation of the `subscribe` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <subscribe xmlns="http://xml.spss.com/notification/remote">
      <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c09f</subscriberIdentifier>
      <subscriptionIdentifier>0a0a4aac0161f10f0000010d8e63c979c098</subscriptionIdentifier>
    </subscribe>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `subscribe` 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>
    <subscribeResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The unsubscribe operation

Removes a subscriber from a subscription.

Input fields

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

Table 77. Fields for unsubscribe		
Field	**Type/Valid Values**	**Description**
subscriberIdentifier	string	An identifier for a subscriber.
subscriptionIdentifier	string	An identifier for a subscription.

Java example

To remove a subscriber from a subscription, supply the `unsubscribe` operation with strings corresponding to the identifiers for both items.

```
String subscriberIdentifier = "0a0a4aac0161f10f0000010d8e63c979c09f";
String subscriptionIdentifier = "0a0a4aac0161f10f0000010d8e63c979c098";
stub.unsubscribe(subscriberIdentifier, subscriptionIdentifier);
```

SOAP request example

Client invocation of the `unsubscribe` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <unsubscribe xmlns="http://xml.spss.com/notification/remote">
      <subscriberIdentifier>0a0a4aac0161f10f0000010d8e63c979c09f</subscriberIdentifier>
      <subscriptionIdentifier>0a0a4aac0161f10f0000010d8e63c979c098</subscriptionIdentifier>
    </unsubscribe>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `unsubscribe` 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>
    <unsubscribeResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The unsubscribeAll operation

Removes all subscribers from a subscription.

Input fields

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

Table 78. Fields for unsubscribeAll		
Field	**Type/Valid Values**	**Description**
subscriptionIdentifier	string	An identifier for a subscription.

Java example

To remove all subscribers from a subscription, supply the `unsubscribeAll` operation with the identifier for the subscription.

```
String subscriptionIdentifier = "0a0a4aac0161f10f0000010d8e63c979c098";
stub.unsubscribeAll(subscriptionIdentifier);
```

SOAP request example

Client invocation of the `unsubscribeAll` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <unsubscribeAll xmlns="http://xml.spss.com/notification/remote">
      <subscriptionIdentifier>0a0a4aac0161f10f0000010d8e63c979c098</subscriptionIdentifier>
    </unsubscribeAll>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `unsubscribeAll` 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>
    <unsubscribeAllResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateContentFormatter operation

Updates the characteristics of an existing content formatter.

Input fields

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

Table 79. Fields for updateContentFormatter		
Field	**Type/Valid Values**	**Description**
contentFormatter	contentFormatter	Handles formatting of the notification message for the appropriate transport protocol.

Java example

To update a content formatter:

1. Call the `getContentFormatter` operation with a content formatter identifier to return the formatter to be updated. Alternatively, if the identifier is not known, the `getContentFormatters` operation could be used to return a vector of all `ContentFormatter` objects in the system, from which the target content formatter could be selected.
2. Set updated values for the formatter as needed.
3. Supply the `updateContentFormatter` operation with the revised content formatter.

The following code updates the template name for an existing content formatter.

```
String formatterIdentifier = "0a0a4aacd90178eb0000010dea8eaedf802a";
ContentFormatter formatter = stub.getContentFormatter(formatterIdentifier);
formatter.setTemplateName("job_failure_new.xml");
stub.updateContentFormatter(formatter);
```

SOAP request example

Client invocation of the `updateContentFormatter` 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>
    <updateContentFormatter xmlns="http://xml.spss.com/notification/remote">
      <contentFormatter xmlns="http://xml.spss.com/notification" templateName="job_failure_new.xml"
          protocolType="smtp">
        <identifier>0a0a4aacd90178eb0000010dea8eaedf802a</identifier>
      </contentFormatter>
    </updateContentFormatter>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updateContentFormatter` 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>
    <updateContentFormatterResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateDeliveryDevice operation

Updates the characteristics of an existing delivery device.

Input fields

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

Table 80. Fields for updateDeliveryDevice		
Field	Type/Valid Values	Description
deliveryDevice	deliveryDevice	Represents a device that can receive notifications.

Java example

To update a delivery device:

1. Call the `getDeliveryDevice` operation with a delivery device identifier to return the device to be updated. Alternatively, if the identifier is not known, the `getDeliveryDevices` operation could be used to return a vector of all `DeliveryDevice` objects in the system, from which the target delivery device could be selected.

2. Set updated values for the delivery device as needed.

3. Supply the `updateDeliveryDevice` operation with the revised delivery device.

The following code updates the address for an existing delivery device.

```
String deviceIdentifier = "0a0a4aac0161f10f0000010d8e63c979c0a0";
DeliveryDevice dDevice = stub.getDeliveryDevice(deviceIdentifier);
dDevice.setAddress("jjones@yahoo.com");
stub.updateDeliveryDevice(dDevice);
```

SOAP request example

Client invocation of the `updateDeliveryDevice` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <soapenv:Body>
      <updateDeliveryDevice xmlns="http://xml.spss.com/notification/remote">
        <ns2:deliveryDevice protocolType="smtp" address="jjones@yahoo.com"
            xmlns:ns2="http://xml.spss.com/notification">
          <ns2:identifier>0a0a4aac0161f10f0000010d8e63c979c0a0</ns2:identifier>
        </ns2:deliveryDevice>
      </updateDeliveryDevice>
    </soapenv:Body>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updateDeliveryDevice` 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>
    <updateDeliveryDeviceResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateDistributionList operation

Updates the characteristics of an existing distribution list.

Input fields

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

Table 81. Fields for updateDistributionList		
Field	**Type/Valid Values**	**Description**
distributionList	distributionList	Represents a logical grouping of the individual subscribers. For internal use only. This is reserved for future use.

Java example

To update a distribution list:

1. Call the `getDistributionList` operation with a distribution list identifier to return the list to be updated. Alternatively, if the identifier is not known, the `getDistributionLists` operation could be used to return a vector of all `DistributionList` objects in the system, from which the target distribution list could be selected.

2. Set updated values for the list as needed.

3. Supply the `updateSubscriber` operation with the revised distribution list.

The following code updates the name for an existing distribution list.

```
String subscriberIdentifier = "0a0a4aacfe9747c10000010dd18630f5804e";
Subscriber subscriber = stub.getSubscriber(subscriberIdentifier);
subscriber.setPrincipalID("//u/smtp/jjones@gmail.com");
stub.updateSubscriber(subscriber);
```

SOAP request example

Client invocation of the `updateDistributionList` 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>
    <updateDistributionList xmlns="http://xml.spss.com/notification/remote">
      <distributionList xmlns="http://xml.spss.com/notification" name="Analysts">
        <identifier>0a0a4aacfe9747c10000010dd18630f581cb</identifier>
      </distributionList>
    </updateDistributionList>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updateDistributionList` 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>
      <updateDistributionListResponse xmlns="http://xml.spss.com/notification/remote"/>
   </soapenv:Body>
</soapenv:Envelope>
```

The updateEventType operation

Updates the characteristics of an existing event type.

Input fields

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

Table 82. Fields for updateEventType		
Field	**Type/Valid Values**	**Description**
eventType	eventType	Identifies a type of the event within its notification domain .

Java example

To update a content formatter:

1. Call the getEventType operation with an event type identifier to return the type to be updated. Alternatively, if the identifier is not known, the getEventTypes operation could be used to return a vector of all EventType objects in the system, from which the target content type could be selected.
2. Set updated values for the event type as needed.
3. Supply the updateEventType operation with the revised event type.

The following code adds a new property to an existing event type.

```
String eventIdentifier = "0a0a4aacfe9747c10000010dd18630f59dc9";
EventType eventType = stub.getEventType(eventIdentifier);
Property property = new Property();
property.setName("JobName");
property.setTypeCode(TypeCode.STRING);
eventType.setProperty(property);
stub.updateEventType(eventType);
```

SOAP request example

Client invocation of the updateEventType 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>
    <updateEventType xmlns="http://xml.spss.com/notification/remote">
      <eventType xmlns="http://xml.spss.com/notification" domainName="MyDomain" typeName="Completion">
        <identifier>0a0a4aacfe9747c10000010dd18630f59dc9</identifier>
        <property typeCode="string" name="JobID"/>
        <property typeCode="boolean" name="Success"/>
        <property typeCode="string" name="JobName"/>
      </eventType>
    </updateEventType>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updateEventType` 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>
    <updateEventTypeResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateNotificationDomain operation

Updates the characteristics of an existing notification domain.

Input fields

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

Table 83. Fields for updateNotificationDomain		
Field	**Type/Valid Values**	**Description**
notificationDomain	notificationDomain	Identifies the particular vertical industry domain or application in which the event type is defined.

Java example

To update a notification domain:

1. Call the `getNotificationDomain` operation with a domain identifier to return the notification domain to be updated. Alternatively, if the identifier is not known, the `getNotificationDomains` operation could be used to return a vector of all `NotificationDomain` objects in the system, from which the target domain could be selected.

2. Set updated values for the notification domain as needed.

3. Supply the `updateNotificationDomain` operation with the revised notification domain.

The following code updates the name for an existing notification domain.

```
String domainIdentifier = "0a0a4aac0161f10f0000010d8e63c979e87c";
NotificationDomain nDomain = stub.getNotificationDomain(domainIdentifier);
nDomain.setName("Search");
stub.updateNotificationDomain(nDomain);
```

SOAP request example

Client invocation of the `updateNotificationDomain` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <updateNotificationDomain xmlns="http://xml.spss.com/notification/remote">
```

```
        <notificationDomain xmlns="http://xml.spss.com/notification" name="Search">
            <identifier>0a0a4aac0161f10f0000010d8e63c979e87c</identifier>
        </notificationDomain>
    </updateNotificationDomain>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updateNotificationDomain` 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>
    <updateNotificationDomainResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateSubscriber operation

Updates the characteristics of an existing subscriber.

Input fields

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

Table 84. Fields for updateSubscriber		
Field	**Type/Valid Values**	**Description**
subscriber	subscriber	Represents an individual subscriber.

Java example

To update a subscriber:

1. Call the `getSubscriber` operation with a subscriber identifier to return the subscriber to be updated. Alternatively, if the identifier is not known, the `getSubscribers` operation could be used to return a vector of all `Subscriber` objects in the system, from which the target subscriber could be selected.

2. Set updated values for the subscriber as needed.

3. Supply the `updateSubscriber` operation with the revised subscriber.

The following code updates the principal ID for an existing subscriber.

```
String subscriberIdentifier = "0a0a4aacfe9747c10000010dd18630f5804e";
Subscriber subscriber = stub.getSubscriber(subscriberIdentifier);
subscriber.setPrincipalID("//u/smtp/jjones@gmail.com");
stub.updateSubscriber(subscriber);
```

SOAP request example

Client invocation of the `updateSubscriber` 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>
        <updateSubscriber xmlns="http://xml.spss.com/notification/remote">
          <subscriber xmlns="http://xml.spss.com/notification" principalID="//u/smtp/jjones@gmail.com">
            <identifier>0a0a4aacfe9747c10000010dd18630f5804e</identifier>
          </subscriber>
        </updateSubscriber>
      </soapenv:Body>
  </soapenv:Envelope>
```

SOAP response example

The server responds to a `updateSubscriber` 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>
    <updateSubscriberResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateSubscription operation

Updates the characteristics of an existing subscription.

Input fields

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

Table 85. Fields for updateSubscription

Field	Type/Valid Values	Description
subscription	subscription	Represents a single subscription.

Java example

To update a subscription:

1. Call the `getSubscription` operation with a subscription identifier to return the subscription to be updated. Alternatively, if the identifier is not known, the `getSubscriptions` operation could be used to return a vector of all `Subscription` objects in the system, from which the target subscription could be selected.
2. Set updated values for the subscription as needed.
3. Supply the `updateSubscription` operation with the revised subscription.

The following code updates the value for the first property of an existing subscription.

```
String subscriptionIdentifier = "0a0a4aac00072ffb000001094fa9f57ecace";
Subscription subscription = stub.getSubscription(subscriptionIdentifier);
PropertyValue propValue = new PropertyValue;
Value value = new Value();
value.setContent(false);
propValue.setValue(value);
subscription.setPropertyValue(0, propValue);
stub.updateSubscription(subscription);
```

SOAP request example

Client invocation of the `updateSubscription` 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>
      <updateSubscription xmlns="http://xml.spss.com/notification/remote">
        <subscription xmlns="http://xml.spss.com/notification">
          <identifier>0a0a4aac00072ffb000001094fa9f57ecace</identifier>
          <propertyValue typeCode="boolean" name="JobSuccess$0">
            <identifier>0a0a4aac00072ffb000001094fa9f57ecad0</identifier>
            <value>false</value>
          </propertyValue>
          <propertyValue typeCode="string" name="JobID$0">
            <identifier>0a0a4aac00072ffb000001094fa9f57ecacf</identifier>
            <value>0a0a4aac00072ffb000001094fa9f57eca9d</value>
          </propertyValue>
        </subscription>
      </updateSubscription>
    </soapenv:Body>
  </soapenv:Envelope>
```

SOAP response example

The server responds to a `updateSubscription` 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>
    <updateSubscriptionResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateSubscriptions operation

Updates the characteristics of one or more existing subscriptions.

Input fields

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

Table 86. Fields for updateSubscriptions

Field	Type/Valid Values	Description
subscription	subscription[]	Represents a single subscription.

Java example

To update one or more subscriptions:

1. Call the `getSubscription` operation for each subscription identifier to return each subscription to be updated. Alternatively, if the identifier is not known, the `getSubscriptions` operation could be used to return a vector of all `Subscription` objects in the system, from which the target subscriptions could be selected.

2. Set updated values for the subscriptions as needed.

3. Supply the `updateSubscriptions` operation with an array containing the revised subscriptions.

The following code updates the value for the first property, the resource ID, of an existing subscription.

```
String subscriptionIdentifier = "0a0a4aac0161f10f0000010d8e63c979c057";
Subscription subscription = stub.getSubscription(subscriptionIdentifier);
PropertyValue propValue = new PropertyValue;
Value value = new Value();
value.setContent("0a0a4aac0161f10f0000010d8e63c9799c24");
propValue.setValue(value);
subscription.setPropertyValue(0, propValue);
stub.updateSubscriptions(subscription);
```

SOAP request example

Client invocation of the `updateSubscriptions` 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>
        <wsse:Username>validUser</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </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>
    <updateSubscriptions xmlns="http://xml.spss.com/notification/remote">
      <ns2:subscription subscriptionSelectorIdentifier="0000000000000000000000000000000000000200"
          enabled="true" scheduled="false" multicasted="false" principalID="//uNative//validUser"
          xmlns:ns2="http://xml.spss.com/notification">
      <ns2:identifier>0a0a4aac0161f10f0000010d8e63c979c057</ns2:identifier>
      <ns2:propertyValue typeCode="string" name="ResourceID$0">
        <ns2:identifier>0a0a4aac0161f10f0000010d8e63c979c058</ns2:identifier>
        <ns2:value>0a0a4aac0161f10f0000010d8e63c9799c24</ns2:value>
      </ns2:propertyValue>
      <ns2:propertyValue typeCode="boolean" name="Attachments$0">
        <ns2:identifier>0a0a4aac0161f10f0000010d8e63c979c059</ns2:identifier>
        <ns2:value>false</ns2:value>
      </ns2:propertyValue>
      </ns2:subscription>
    </updateSubscriptions>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updateSubscriptions` 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>
    <updateSubscriptionsResponse xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The updateSubscriptionSelector operation

Updates the characteristics of an existing subscription selector.

Input fields

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

Table 87. Fields for updateSubscriptionSelector		
Field	**Type/Valid Values**	**Description**
subscriptionSelector	subscriptionSelector	Defines a subscription selector for the given event type.

Java example

To update a subscription selector:

1. Call the `getSubscriptionSelector` operation with a subscription selector identifier to return the selector to be updated. Alternatively, if the identifier is not known, the `getSubscriptionSelectors` operation could be used to return a vector of all `SubscriptionSelector` objects in the system, from which the target subscription selector could be selected.

2. Set updated values for the selector as needed.
3. Supply the `updateSubscriptionSelector` operation with the revised selector.

The following code updates the name for an existing subscription selector.

```
String selectorIdentifier = "0a0a483800a83a1300000102652a73568013";
SubscriptionSelector selector = stub.getSubscriptionSelector(selectorIdentifier);
selector.setName("new_name");
stub.updateSubscriptionSelector(selector);
```

SOAP request example

Client invocation of the `updateSubscriptionSelector` 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>
    <updateSubscriptionSelector xmlns="http://xml.spss.com/notification/remote">
     <subscriptionSelector xmlns="http://xml.spss.com/notification"
        filterExpression="JobID=='JobID' &amp;&amp; JobSuccess==true"
        compiledFilterExpression="JobID$0=='${JobID}' &amp;&amp; JobSuccess$0==${JobSuccess}"
        name="new_name">
      <identifier>0a0a483800a83a1300000102652a73568013</identifier>
     </subscriptionSelector>
    </updateSubscriptionSelector>
 </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a `updateSubscriptionSelector` 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>
    <updateSubscriptionSelectorResponse xmlns="http://xml.spss.com/notification/remote"/>
 </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 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" on page 95 for more information.
2. Configure the web service endpoint. See the topic "Configuring the web service endpoint" on page 96 for more information.
3. Programmatically configure the necessary endpoint behaviors. See the topic "Configuring endpoint behaviors" on page 97 for more information.
4. Exercise the web service. See the topic "Exercising the service" on page 97 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 88. 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 89. 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 90. Attributes of `wsse:UsernameToken`

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

Table 91. 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 92. 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 92. 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 93. 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.

Table 93. HTTP headers (continued)	
HTTP header	**Description**
`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.

Appendix A. Deprecated features

If you are migrating from an earlier release of IBM SPSS Collaboration and Deployment Services, you should be aware of the various features that have been deprecated since the last version.

If a feature is deprecated, IBM Corp. might remove this capability in a subsequent release of the product. Future investment will be focused on the strategic function listed under the recommended migration action. Typically, a feature is not deprecated unless an equivalent alternative is provided.

No features have been deprecated in this release. For reference purposes, the following table indicates features that were deprecated in recent previous versions of the product. Where possible, the table also indicates the recommended migration action.

Table 94. Features deprecated in previous versions

Deprecation	Recommended migration action
Security Provider: Active Directory with local override, which supports extended groups and allowed users	Use the standard Active Directory security provider with any necessary groups added
IBM SPSS Collaboration and Deployment Services Enterprise View	Use the Analytic Data View feature
IBM SPSS Collaboration and Deployment Services Enterprise View Driver	Use the Analytic Data View feature
Scenario files	Scenario files (.scn) are no longer supported. Enterprise View source nodes cannot be modified in Deployment Manager. Old scenario files can be modified in IBM SPSS Modeler client and resaved as stream files. Also, scoring configurations that used a scenario file must be deleted and recreated based on a stream file.
Web Install for IBM SPSS Deployment Manager	Use the standalone installer
BIRT Report Designer for IBM SPSS	None
BIRT Report Designer for IBM SPSS viewer	None
IBM SPSS Collaboration and Deployment Services Portlet	Use the IBM SPSS Collaboration and Deployment Services Deployment Portal directly, or use the web services APIs
IBM SPSS Collaboration and Deployment Services Web Part	Use the IBM SPSS Collaboration and Deployment Services Deployment Portal directly, or use the web services APIs
Scoring Service V1 API	Scoring Service V2 API
Scheduling Server Service	None
Reporting Service	None
Authentication Service `login` operation	Authentication Service `doLogin` operation
Search Service `search` operation	Search Service `search2.5` operation
SPSS AXIS/Castor web services client jar	Use the tools provided with the Java Runtime Environment, Integrated Development Environment, or Eclipse Web Tools Platform (WTP)

Table 94. Features deprecated in previous versions (continued)	
Deprecation	**Recommended migration action**
`clemrtl_setLogFile()` API function	None

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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, MD-NC119
Armonk, NY 10504-1785
US

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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide 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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

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.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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-IBMproducts. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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 actual people or business enterprises 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.

Privacy policy considerations

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

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

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

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

Trademarks

IBM, the IBM logo, and ibm.com 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

This glossary includes terms and definitions for IBM SPSS Collaboration and Deployment Services.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology (opens in new window).

A

access control list (ACL)
> In computer security, a list associated with an object that identifies all the subjects that can access the object and their access rights.

ACL
> See access control list.

action
> A permission for an aspect of system functionality. For example, the ability to set up notifications is defined as an action. Actions are grouped and assigned to users through roles. See also role.

Active Directory (AD)
> A hierarchical directory service that enables centralized, secure management of an entire network, which is a central component of the Microsoft Windows platform.

AD
> See Active Directory.

allowed user
> A subset of the users defined in a remote directory, such as SiteMinder or Windows Active Directory, that are allowed access to SPSS Predictive Enterprise Services. Allowed users are defined when only a few users in a remote directory need access to the application.

API
> See application programming interface.

appender
> A component that receives logging requests from a logger and writes log statements to a specified file or console. See also logger.

application programming interface (API)
> An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

B

batch file
> A file that contains instructions that are processed sequentially, as a unit.

binary large object (BLOB)
> A data type whose value is a sequence of bytes that can range in size from 0 bytes to 2 gigabytes less 1 byte. This sequence does not have an associated code page and character set. BLOBs can contain, for example, image, audio, or video data.

BLOB
> See binary large object.

break group
A set of rows of returned data that are grouped according to a common column value. For example, in a column of states, the rows of data for each state are grouped together.

burst report
A report that generates multiple output files during a single run by using multiple input parameters taken from break groups in the report.

C

cascading permission
A permission of a parent folder in the content repository that has been propagated to its child objects.

character large object (CLOB)
A data type whose value is a sequence of characters (single byte, multibyte, or both) that can range in size from 0 bytes to 2 gigabytes less 1 byte. In general, the CLOB data type is used whenever a character string might exceed the limits of the VARCHAR data type.

CLOB
See character large object.

common warehouse metamodel (CWM)
A metamodel written to be a common standard by the Object Management Group (OMG).

content repository
A centralized location for storing analytical assets, such as models and data. Content repository includes facilities for security and access control, content management, and process automation.

context data
Input data that is passed with a scoring request in real time. For example, when a score is requested for a customer based on credit rating and geocode, the credit score and geocode will be the context data for the request.

credential
Information acquired during authentication that describes a user, group associations, or other security-related identity attributes, and that is used to perform services such as authorization, auditing, or delegation. For example, a user ID and password are credentials that allow access to network and system resources.

CWM
See common warehouse metamodel.

D

data warehouse
A subject-oriented collection of data that is used to support strategic decision making. The warehouse is the central point of data integration for business intelligence. It is the source of data for data marts within an enterprise and delivers a common view of enterprise data.

distinguished name (DN)
The name that uniquely identifies an entry in a directory. A distinguished name is made up of attribute:value pairs, separated by commas. For example, CN=person name and C=country or region.

DN
See distinguished name.

Document Object Model (DOM)
A system in which a structured document, for example an XML file, is viewed as a tree of objects that can be programmatically accessed and updated. See also Simple API for XML.

document type definition (DTD)
The rules that specify the structure for a particular class of SGML or XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each element, attribute, and notation can be used within the particular class of documents.

DOM
See Document Object Model.

dormant schedule
A schedule associated with a deleted or unlabeled version of a job. A dormant schedule cannot be used until it is associated with a valid labeled job version.

DTD
See document type definition.

E

EAR
See enterprise archive.

enterprise archive (EAR)
A specialized type of JAR file, defined by the Java EE standard, used to deploy Java EE applications to Java EE application servers. An EAR file contains EJB components, a deployment descriptor, and web archive (WAR) files for individual web applications. See also Java archive, web archive.

execution server
A server that enables analytical processing of resources stored in the repository. For example, to execute an IBM SPSS Statistics syntax in an IBM SPSS Collaboration and Deployment Services job, an IBM SPSS Statistics execution server must be designated.

export
The process of storing objects and metadata from the content repository to an external file.

extended group
A locally-defined group of remote users. Extended groups are defined when groups in the remote directory are not fine-grained enough.

Extensible Markup Language (XML)
A standard metalanguage for defining markup languages that is based on Standard Generalized Markup Language (SGML).

Extensible Stylesheet Language (XSL)
A language for specifying style sheets for XML documents. Extensible Stylesheet Language Transformation (XSLT) is used with XSL to describe how an XML document is transformed into another document.

F

field content assist
A feature that provides predefined system and variable values for entry fields.

G

general job step
A method for running native operating system commands and executable programs on a host or a remote process server. General jobs have access to files stored within the repository and on the file system and can be used to control the input/output of analytical processing.

I

import
The process of adding objects and metadata defined in an external file generated by export, to the content repository.

iterative consumer reporting job step
A job step that is passed a set of input values generated by a preceding iterative producer reporting job step. The report in iterative consumer job step is executed for each tuple in the received data set.

iterative producer reporting job step
A job step that generates a set of values passed as input parameters to a following iterative consumer job step.

J

JAAS
See Java Authentication and Authorization Service.

JAR
See Java archive.

Java archive (JAR)
A compressed file format for storing all of the resources that are required to install and run a Java program in a single file. See also enterprise archive, web archive.

Java Authentication and Authorization Service (JAAS)
In Java EE technology, a standard API for performing security-based operations. Through JAAS, services can authenticate and authorize users while enabling the applications to remain independent from underlying technologies.

Java Generic Security Services (JGSS)
A specification that provides Java programs access to the services that include the signing and sealing of messages and a generic authentication mechanism.

Java Naming and Directory Interface (JNDI)
An extension to the Java platform that provides a standard interface for heterogeneous naming and directory services.

JGSS
See Java Generic Security Services.

JNDI
See Java Naming and Directory Interface.

job
A mechanism for automating analytical processing. A job consists of job steps, executed sequentially or conditionally. Input parameters can be defined for a job. A job can be run on demand or triggered by time-based or message-based schedules, with records of job execution stored as job history.

job step
A discrete unit of processing in a job. Depending on the type, job steps can be run on the content repository host or specially defined execution or remote process servers. Objects stored in the repository or the file system can provide input for a job step, and job step output can be stored in the repository or written to the file system.

K

KDC
See key distribution center.

Kerberos
A network authentication protocol that is based on symmetric key cryptography. Kerberos assigns a unique key, called a ticket, to each user who logs on to the network. The ticket is embedded in messages that are sent over the network. The receiver of a message uses the ticket to authenticate the sender.

key distribution center (KDC)
A network service that provides tickets and temporary session keys. The KDC maintains a database of principals (users and services) and their associated secret keys. It is composed of the authentication server and the ticket granting ticket server.

keystore
In security, a file or a hardware cryptographic card where identities and private keys are stored, for authentication and encryption purposes. Some keystores also contain trusted or public keys.

L

LDAP
See Lightweight Directory Access Protocol.

Lightweight Directory Access Protocol (LDAP)
An open protocol that uses TCP/IP to provide access to directories that support an X.500 model and that does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). For example, LDAP can be used to locate people, organizations, and other resources in an Internet or intranet directory.

lock
The process by which integrity of data is ensured by preventing more than one user from accessing or changing the same data or object at the same time.

logger
A component that prepares log statements to be written to console or log file. See also appender.

M

message-based schedule
A schedule that is used to trigger job execution by an event signalled by a Java Messaging Service (JMS) message. For example, when a job relies on the input from a third-party application, the application must send a JMS message when the input file is ready for processing.

metamodel
A model that defines the language for expressing a model.

meta-object
An instance of an XMI class as defined in the metamodel.

meta-object facility (MOF)
A generalized facility and repository for storing abstract information about concrete object systems; dealing mostly with construction, standardized by the Object Management Group (OMG).

MIME
See Multipurpose Internet Mail Extensions.

MOF
See meta-object facility.

Multipurpose Internet Mail Extensions (MIME)
An Internet standard that allows different forms of data, including video, audio, or binary data, to be attached to email without requiring translation into ASCII text.

N

notification
A mechanism that is used to generate email messages informing users of specific types of system events, such as changes to content repository objects and processing success and failure. Unlike subscriptions, notifications can be set up to send email to multiple users.

O

Object Management Group (OMG)
A non-profit consortium whose purpose is to promote object-oriented technology and the standardization of that technology. The Object Management Group was formed to help reduce the complexity, lower the costs, and hasten the introduction of new software applications.

ODS
See Output Delivery System.

OMG
See Object Management Group.

Output Delivery System (ODS)
A method of controlling the destination for output within SAS. ODS can route SAS output to a SAS data file, a text listing file, HTML files, and files optimized for high-resolution printing.

P

package
An installable unit of a software product. Software product packages are separately installable units that can operate independently from other packages of that software product.

principal
An entity that can communicate securely with another entity. A principal is identified by its associated security context, which defines its access rights.

R

remote process server
A remote system that is designated for running native operating system commands and executable programs.

repository content adapter
An optional software package that enables storing and processing content from other IBM SPSS applications, such as Statistics, Modeler, and Data Collection, as well as third parties.

repository database
A relational database that is used for storing content repository objects and metadata.

resource
A content repository object.

resource definition
A subset of content repository resources used to enable analytical processing, such as definitions of data sources, credentials, execution servers, and JMS message domains.

role
A set of permissions or access rights. See also action.

S

SAX
See Simple API for XML.

schedule
A content repository object that triggers job execution.

scoring configuration
A configuration that defines model-specific settings for generating real-time scores, such as input data, processing rules, outputs, logging, etc.

security provider
A system that performs user authentication. Users and groups can be defined locally (in which case, IBM SPSS Collaboration and Deployment Services itself is the security provider) or derived from a remote directory, such as Windows Active Directory or OpenLDAP.

service provider interface (SPI)
An API that supports replaceable components and can be implemented or extended by a third party.

SGML
See Standard Generalized Markup Language.

shell script
A program, or script, that is interpreted by the shell of an operating system.

Simple API for XML (SAX)
An event-driven, serial-access protocol for accessing XML documents, used. A Java-only API, SAX is used by most servlets and network programs to transmit and receive XML documents. See also Document Object Model.

single sign-on (SSO)
An authentication process in which a user can access more than one system or application by entering a single user ID and password.

SOAP
A lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP can be used to query and return information and invoke services across the Internet.

SPI
See service provider interface.

SSO
See single sign-on.

Standard Generalized Markup Language (SGML)
A standard metalanguage for defining markup languages that is based on the ISO 8879 standard. SGML focuses on structuring information rather than presenting information; it separates the structure and content from the presentation. It also facilitates the interchange of documents across an electronic medium.

stop word
A word that is commonly used, such as "the," "an," or "and," that is ignored by a search application.

subscription
Email notices and Really Simple Syndication (RSS) feeds that repository users create to receive when the state of an asset changes.

T

TGT
See ticket-granting ticket.

ticket-granting ticket (TGT)
A ticket that allows access to the ticket granting service on the key distribution center (KDC). Ticket granting tickets are passed to the principal by the KDC after the principal has completed a successful request. In a Windows 2000 environment, a user logs on to the network and the KDC will verify the principal's name and encrypted password and then send a ticket granting ticket to the user.

time-based schedule
A schedule that triggers job execution at a specified time or date. For example, a time-based schedule may run a job at 5:00 pm every Thursday.

U

Universally Unique Identifier (UUID)
The 128-bit numeric identifier that is used to ensure that two components do not have the same identifier.

UUID
See Universally Unique Identifier.

V

Velocity
A Java-based template engine that provides a simple and powerful template language to reference objects defined in Java code. Velocity is an open source package directed by the Apache Project.

W

W3C
See World Wide Web Consortium.

WAR
See web archive.

web archive (WAR)
A compressed file format, defined by the Java EE standard, for storing all the resources required to install and run a web application in a single file. See also enterprise archive, Java archive.

Web Services Description Language (WSDL)
An XML-based specification for describing networked services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.

World Wide Web Consortium (W3C)
An international industry consortium set up to develop common protocols to promote evolution and interoperability of the World Wide Web.

WSDL
See Web Services Description Language.

X

XMI
See XML Metadata Interchange.

XML
See Extensible Markup Language.

XML Metadata Interchange (XMI)
A model-driven XML integration framework for defining, interchanging, manipulating, and integrating XML data and objects. XMI-based standards are in use for integrating tools, repositories, applications, and data warehouses.

XSL
See Extensible Stylesheet Language.

Index