IBM SPSS Collaboration and Deployment Services 5 Subscription Manager Service Developer’s Guide

Preface

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

About IBM Business Analytics

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

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

Technical support

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

Contents

Appendices

Chapter

1

Introduction to web services

What are web services?

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

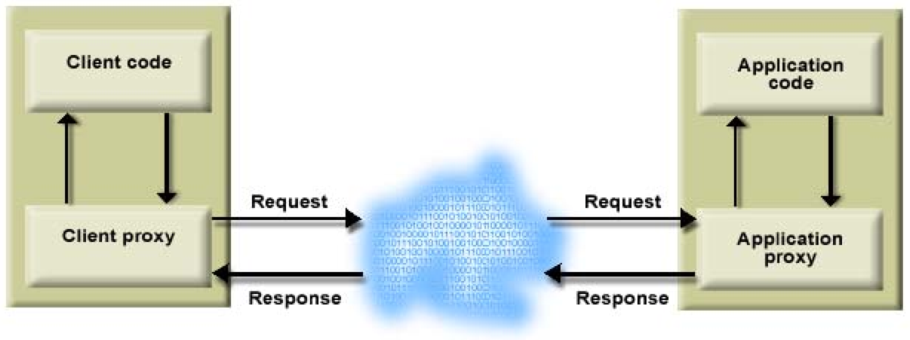
Advantages to implementing application functionality as web services include the following:

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

Web service system architecture

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

Figure 1-1
Web service architecture



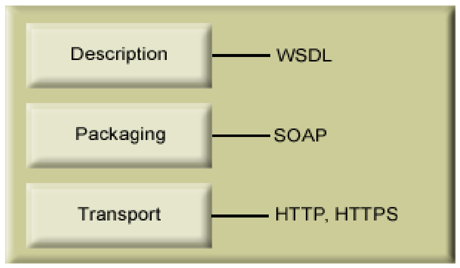
The client code supplies input to an operation offered by a proxy class. The proxy class generates a request containing a standardized representation of the input and sends it across the network to the application. A proxy class on the server receives the request and unmarshals the contents into objects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

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

Web service protocol stack

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

Figure 1-2
Web service protocol stack



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

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

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

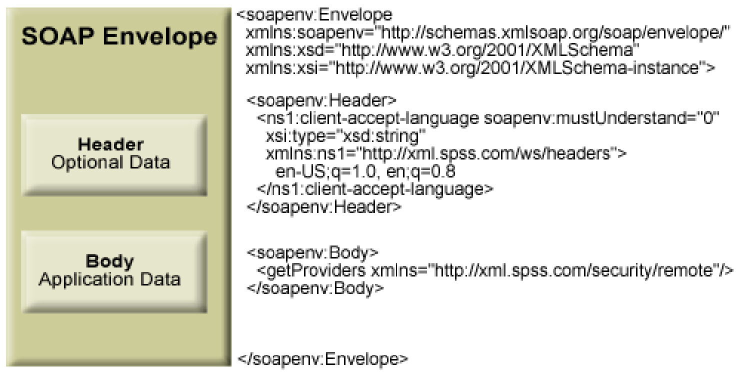
Simple Object Access Protocol

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

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

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

Figure 1-3
An example SOAP packet



```
<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>
    <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>
    <getProviders xmlns="http://xml.spss.com/security/remote"/>
  </soapenv:Body>

</soapenv:Envelope>
```

Web Service Description Language

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

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

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

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

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

Types

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

```
<wsdl:types>
 <schema targetNamespace="http://xml.spss.com/security/remote"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="getProviders">
   <complexType />
  </element>
  <element name="getProvidersResponse">
   <complexType>
    <sequence>
     <element name="providerInfo[unbounded]" type="tns1:providerInfo" />
    </sequence>
   </complexType>
  </element>
 </schema>
</wsdl:types>
```

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

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

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

Messages

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

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

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

Port types

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

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

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

Bindings

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

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

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

Services

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

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

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

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

Proxies

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

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

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

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

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

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

Chapter
2

Subscription Manager Service overview

The Subscription Manager Service allows a client to manage notification plug-ins, which augment the standard services with additional functionality. For instance, plug-ins can generate e-mail distribution lists from database queries. The service also includes operations for message template management.

Accessing the Subscription Manager Service

To access the functionality offered by the Subscription Manager 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>/notification/services/SubscriptionManager
```

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

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

```
http://cads_server:80/notification/services/SubscriptionManager?wsdl
```

Calling Subscription Manager 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/SubscriptionManager";
URL url = new URL("http", "cads_server", 80, context);
SubscriptionManagerService service = new SubscriptionManagerServiceLocator();
stub = service.getSubscriptionManager(url);
```

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

```
stub.findMessageTemplate(templateSpec);
```

Chapter
3

Subscription manager concepts

Notification providers

A notification provider is a plug-in to the general notification service that extends the standard functionality. For example, the generation of a list of notification recipients through an iterative process is handled by a recipient provider plug-in. The definition of a notification provider includes the specification of a type, an optional list of provider parameters, and optional dependency sets. The “Provider Types” table identifies notification providers available in Subscription Manager Service.

Table 3-1
Provider Types

Type	Interface
Reporting.ReportRecipientProvider	RecipientProvider
ProcessManagement.IterativeRecipientProvider	RecipientProvider
ProcessManagement.JobStepCompletionEventSplitter	EventSplitter NotificationProvider
HierarchicalContent.FolderEventSplitter	EventSplitter NotificationProvider
HierarchicalContent.FileEventSplitter	EventSplitter NotificationProvider
HierarchicalContent.FolderSubscriptionValidator	SubscriptionValidator NotificationProvider
HierarchicalContent.FileSubscriptionValidator	SubscriptionValidator NotificationProvider
SPSSNotification.DirectoryBusinessInfoProvider	BusinessInfoProvider

Dependency sets correspond to notification objects that depend on the provider. For example, recipient providers can be associated with a subscription. During processing of the notification event, the service invokes all providers associated with the matching subscription, collects generated recipients, and adds them to the list of notification recipients.

The Subscription Manager Service includes operations for creating, retrieving, updating, and deleting notification providers.

Message templates

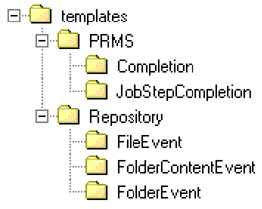
Message templates define the structure and content of notification messages. A template contains some fixed text and one or more content variables that are replaced with relevant information derived from the event triggering the notification. The template specification defines the protocol for the notification 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, the Subscription Manager Service 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 directory tree in the "IBM SPSS Collaboration and Deployment Services template directories" figure.

Figure 3-1
IBM SPSS Collaboration and Deployment Services template directories



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

The Subscription Manager Service includes operations for finding and applying message templates.

Notification message template structure

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

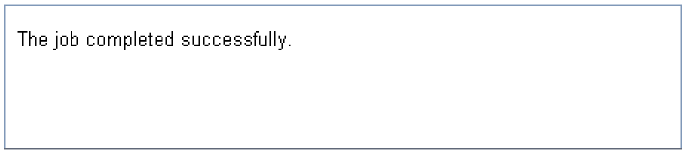
Velocity template structure

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 e-mail processor. The following sample template generates a simple, generic e-mail message indicating the success of the corresponding job.

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

The following figure displays the resulting e-mail.

Figure 3-2
Generic notification message



The job completed successfully.

For more information about Velocity templates, see the Apache Velocity project (*http://velocity.apache.org/*) documentation.

Message properties

E-mail 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 e-mail address assigned to the message. Default SMTP properties are listed under repository notification configuration options. If 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 3-2
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.
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 e-mail 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.

Message Property	Attribute	Event Property	Description
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>
```

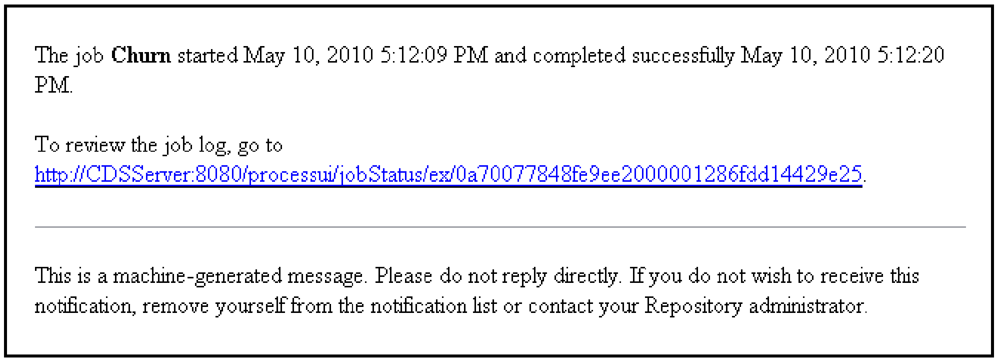
```
<p>The job <b>${JobName}</b> started ${JobStart} and #if($JobSuccess) completed successfully #else failed #end ${JobEnd}.

<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 wish to receive this notification, remove yourself fr
</body>
</html>
```

The following figure displays the resulting e-mail.

Figure 3-3
Message using customized content



The job **Churn** started May 10, 2010 5:12:09 PM and completed successfully May 10, 2010 5:12:20 PM.

To review the job log, go to
http://CDSServer:8080/processui/jobStatus/ex/0a70077848fe9ee2000001286fdd14429e25.

This is a machine-generated message. Please do not reply directly. If you do not wish to receive this notification, remove yourself from the notification list or contact your Repository administrator.

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® Collaboration and Deployment Services 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.

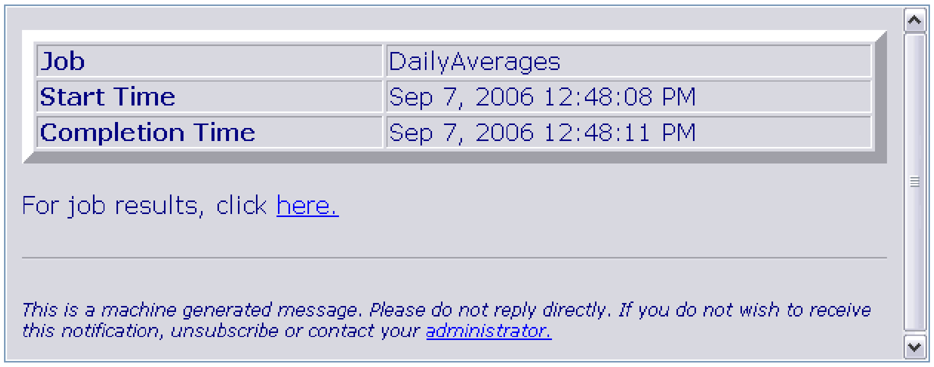
The following template generates a notification message for job step completion, formats content as a table, specifies background color for the message using an inline style for body, and defines a blue Verdana font for paragraphs using an internal style sheet. The message also includes a link to the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

The following figure displays the resulting e-mail.

Figure 3-4
Message using customized formatting



Job	DailyAverages
Start Time	Sep 7, 2006 12:48:08 PM
Completion Time	Sep 7, 2006 12:48:11 PM

For job results, click here.

This is a machine generated message. Please do not reply directly. If you do not wish to receive this notification, unsubscribe or contact your administrator.

Chapter

4

Operation reference

The addNotificationProvider operation

Creates a new instance of a notification provider and associates it with the specified notification elements.

Input fields

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

Table 4-1
Fields for addNotificationProvider

Field	Type/Valid Values	Description
notificationProvider	notificationProvider	A provider for the notification service.

Return information

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

Table 4-2
Return Value

Type	Description
string	The identifier for the new notification provider.

Java example

Adding a notification provider involves the following steps:

1. Create a NotificationProvider object.
2. Use the setType method to define the type for the provider. A list of valid types can be retrieved using the getProviderTypes method.
3. Define the parameters for the provider using the setParameters method.
4. Create a DependencySet object to identify any dependencies the provider has on other notification objects. For example, this object can specify the identifier of a subscription associated with the provider. Use the setDependencySet method to assign the dependencies to the provider.
5. Supply the addNotificationProvider operation with the provider object.

The following sample creates a recipient provider that generates a list of email addresses from a database column.

```
NotificationProvider notificationProvider = new NotificationProvider();
notificationProvider.setType("Reporting.ReportRecipientProvider");
```

```
String parameters = "Source>/employees.rptdesign>Version>LATEST>
  SourceVersion>0a0b32e0ba9a60b50000010eecc7c28b82de>
  DataSet>ds1>DataSource>dbserver>Credentials>/validUser>
  CredentialsVersion>0a0b32e0544c7e680000010f2ab3162c806c>EmailColumn>EMP_ID";
notificationProvider.setParameters(parameters);
String identifier = stub.addNotificationProvider(notificationProvider);
```

The applyMessageTemplate operation

Applies the notification message template to the specified subscriptions.

Input fields

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

Table 4-3
Fields for applyMessageTemplate

Field	Type/Valid Values	Description
messageTemplateSpecification	messageTemplateSpecification	A specification for the message template used to format a notification message.

Java example

Applying a message template involves the following steps:

1. Create a MessageTemplateSpecification object.
2. Define the protocol type for the message using the addContentFormatterProtocolType method. The Subscription Manager Service currently only supports the SMTP protocol.
3. Supply the addSubscriptionIdentifier method with a string corresponding to the identifier of the subscription associated with the message.
4. Create a MessageTemplateContent object.
5. Supply the setContent method with a string corresponding to the message content.
6. Add the content object to the specification using the addMessageTemplateContent method.
7. Supply the applyMessageTemplate operation with the specification object.

The following sample applies a custom template to a subscription.

```
// create a specification
MessageTemplateSpecification msgTemplateSpec = new MessageTemplateSpecification();
msgTemplateSpec.addContentFormatterProtocolType(ProtocolType.SMTP);
msgTemplateSpec.addSubscriptionIdentifier("0a0a4aacfe9747c10000010dd18630f58047");

MessageTemplateContent msgTemplateContent = new MessageTemplateContent()
String templateContent = "/mimeMessage/messageSubject=PASW Services:
  New content in ${ResourcePath}
  /mimeMessage/messageContent[text/html;charset=utf-8]=
  <html>
  <head>
  <meta http-equiv="Content-Type"
  content="text/html;charset=utf-8"/>
  </head>
  <body>
  <p>The #if( $ActionType=="FileVersionCreated" ) new version of #end
  file <b>${ChildName}</b> has been created in folder
  <b>${ResourcePath}</b>.</p>
  <p>To review the content of the file, go to <a href='${ChildHttpUrl}'>
  ${ChildHttpUrl}</a>. </br></p>
  <hr><p>This is a machine-generated message. Please do not reply directly.
  If you do not wish to receive this notification, unsubscribe or contact your
  administrator.</p>
  </body>
  </html>";
msgTemplateContent.setContent(templateContent);
msgTemplateSpec.addMessageTemplateContent(messageTemplateContent);
stub.applyMessageTemplate(messageTemplateSpecification);
```

SOAP request example

Client invocation of the applyMessageTemplate 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>
  <applyMessageTemplate xmlns="http://xml.spss.com/notification/remote">
   <ns2:messageTemplateSpecification templateProviderType="database"
   xmlns:ns2="http://xml.spss.com/notification">
    <ns2:messageTemplateContent>
     <ns2:content>
      /mimeMessage/messageSubject=PASW Services:
        New content in ${ResourcePath}
      /mimeMessage/messageContent[text/html;charset=utf-8]=
      &lt;html&gt;
      &lt;head&gt;
      &lt;meta http-equiv=&quot;Content-Type&quot;
```

```
            content=&quot;text/html;charset=utf-8&quot;/&gt;
          &lt;/head&gt;
          &lt;body&gt;
          &lt;p&gt;The #if( $ActionType==&quot;FileVersionCreated&quot; ) new version of #end
          file &lt;b&gt;${ChildName}&lt;/b&gt; has been created in folder
            &lt;b&gt;${ResourcePath}&lt;/b&gt;.&lt;/p&gt;

          &lt;p&gt;To review the content of the file, go to &lt;a href='${ChildHttpUrl}'&gt;
            ${ChildHttpUrl}&lt;/a&gt;. &lt;/br&gt;&lt;/p&gt;
          &lt;hr&gt;&lt;p&gt;This is a machine-generated message. Please do not reply directly.
            If you do not wish to receive this notification, unsubscribe or contact your
            administrator.&lt;/p&gt;
          &lt;/body&gt;
          &lt;/html&gt;
        </ns2:content>
      </ns2:messageTemplateContent>
      <ns2:subscriptionIdentifier>0a0a4aacfe9747c10000010dd18630f58047</ns2:subscriptionIdentifier>
      <ns2:contentFormatterProtocolType>smtp</ns2:contentFormatterProtocolType>
    </ns2:messageTemplateSpecification>
  </applyMessageTemplate>
 </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

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

The copy operation

Copies subscriptions associated with a source repository object to a new target object.

Input fields

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

Table 4-4
Fields for copy

Field	Type/Valid Values	Description
copySpecification	copySpecification	A copy specification for notification objects.

Java example

Copying a subscription involves the following steps:

1. Create a CopySpecification object.
2. Create a SubscriptionCopySpecification object.
3. Specify the identifier for the repository object having the subscription to be copied using the setSourceIdentifier method.
4. Specify the identifier for the repository object to which to copy the subscription using the setTargetIdentifier method.
5. Add the subscription specification object to the copy specification using the setSubscriptionCopySpecification method.
6. Supply the copy operation with the specification object.

The following sample copies the subscriptions for one object to another object.

```
CopySpecification spec = new CopySpecification();
SubscriptionCopySpecification subsSpec = new SubscriptionCopySpecification();
subsSpec.setSourceIdentifier("0a0a4a35ae5916610000010eec3387118177");
subsSpec.setTargetIdentifier("0a0a4a35c75d16910000010eec1efbfc8006");
spec.setSubscriptionCopySpecification(subsSpec);
stub.copy(spec);
```

SOAP response example

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

The deleteNotificationProvider operation

Removes a specified notification provider from the system.

Input fields

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

Table 4-5
Fields for deleteNotificationProvider

Field	Type/Valid Values	Description
identifier	string	A notification provider identifier.

Java example

To delete a notification provider, supply the deleteNotificationProvider operation with the identifier for the provider.

```
String providerIdentifier = "0a0b32e0cfc420760000010ec3b57caf8252";
stub.deleteNotificationProvider(providerIdentifier);
```

SOAP response example

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

The enumerateMessageTemplates operation

Returns an array of all notification message templates for a specified event type in a domain. Each uniquely named template for the given notification transport protocol will be returned as an element of the array. If the event type has no associated templates, the array will be empty.

Input fields

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

Table 4-6
Fields for enumerateMessageTemplates

Field	Type/Valid Values	Description
messageTemplateSpecification	messageTemplateSpecification	A specification for the message template used to format a notification message.

Return information

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

Table 4-7
Return Value

Type	Description
messageTemplate[]	The path to and actual content of a notification template.

Java example

To retrieve a set of notification message templates:

1. Create a MessageTemplateSpecification object.
2. Specify the domain of the templates using the setDomainName method.
3. Specify the event type using the setTypeName method.
4. Define the provider type using the setTemplateProviderType method.
5. Supply the enumerateMessageTemplates operation with the specification.

   The following code retrieves the message templates for the *Completion* type in the *PRMS* domain.

```
MessageTemplateSpecification messageTemplateSpecification =
  new MessageTemplateSpecification();
messageTemplateSpecification.setDomainName("PRMS");
messageTemplateSpecification.setTypeName("Completion");
messageTemplateSpecification.setTemplateProviderType(TemplateProviderType.FILE);
MessageTemplate[] messageTemplates =
  subscriptionManager.enumerateMessageTemplates(messageTemplateSpecification);
```

For any returned message template, the localizable content for all supported locales can be accessed using the getMessageTemplateContent method.

SOAP request example

Client invocation of the enumerateMessageTemplates 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>
    <enumerateMessageTemplates xmlns="http://xml.spss.com/notification/remote">
      <messageTemplateSpecification domainName="PRMS" typeName="Completion"
          templateProviderType="file" xmlns="http://xml.spss.com/notification">
      </messageTemplateSpecification>
    </enumerateMessageTemplates>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a enumerateMessageTemplates 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>
    <enumerateMessageTemplatesResponse xmlns="http://xml.spss.com/notification/remote">
      <ns1:messageTemplate templateName="job_completion.vm" domainName="PRMS" typeName="Completion"
        templateProviderType="file" xmlns:ns1="http://xml.spss.com/notification">
        <ns1:path>/PRMS/Completion/job_completion.vm</ns1:path>
        <ns1:messageTemplateContent>
          <ns1:content>#* A Velocity template to replace the XSL notification templates for job
            completion status (covers both success and failure) *# /mimeMessage/messageSubject=PASW
            Services: Job ${JobName}#if($JobSuccess) completed successfully
            #else failed #end /mimeMessage/messageContent[text/html;charset=utf-8]=
            &lt;html&gt; &lt;head&gt; &lt;meta http-equiv='Content-Type'
            content='text/html;charset=utf-8'/&gt; &lt;/head&gt; &lt;body&gt;
            &lt;p&gt;PASW Services job
            &lt;b&gt;${JobName}&lt;/b&gt; started ${JobStart} and #if($JobSuccess)
            completed successfully #else failed #end ${JobEnd}. &lt;p&gt;To review the job
            log, go to &lt;a
            href='${JobStatusURL}'&gt;${JobStatusURL}&lt;/a&gt;.&lt;/p&gt;
            &lt;hr&gt;&lt;p&gt;This is a machine-generated message. Please do not
            reply directly. If you do not wish to receive this notification, remove yourself from
            the notification list or contact your
            administrator.&lt;/p&gt; &lt;/body&gt; &lt;/html&gt;
          </ns1:content>
        </ns1:messageTemplateContent>
      </ns1:messageTemplate>
    </enumerateMessageTemplatesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The executeScript operation

Executes a script stored on the server. Script execution is for internal purposes only.

For security purposes, this operation cannot be used to exercise any arbitrary script. The script must have been deployed on the server during setup as a trusted component. Typically scripts are stored in a subdirectory of:

<install-dir>\components\notification\scripts

Input fields

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

Table 4-8
Fields for executeScript

Field	Type/Valid Values	Description
script	script	An executable script.

Return information

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

Table 4-9
Return Value

Type	Description
string	The results of running the script.

Java example

To execute a script, create an Script object. Use the setScriptContent method to specify the name of the script file. Supply this object to the executeScript operation.

```
Script myScript = new Script();
myScript.setScriptContent("pem/helloPEM.py");
String response = stub.executeScript(myScript);
System.out.println("Script response: " + response);
```

The findMessageTemplate operation

Returns a notification message template for the specified subscriptions. The operation returns null if the template cannot be found.

Input fields

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

Table 4-10
Fields for findMessageTemplate

Field	Type/Valid Values	Description
messageTemplateSpecification	messageTemplateSpecification	A specification for the message template used to format a notification message.

Return information

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

Table 4-11
Return Value

Type	Description
messageTemplate	The path to and actual content of a notification template.

Java example

To return a message template:

1. Create a MessageTemplateSpecification object to contain the properties of the template to be retrieved.

2. Define the domain containing the event type associated with the template using the setDomainName method.

3. Specify the event type using the setTypeName method.

4. Supply the setStrict method with a boolean indicating whether or not the specification includes the full path to the template. If true, use the setPath method to define the path. If false, the retrieval attempt uses the template folder structure to find the template.

5. Specify the identifier for the subscription associated with the template using the addSubscriptionIdentifier method.

6. Supply the findMessageTemplate operation with the specification object.

The following sample retrieves the notification template for changes in a folder's content from the *Repository* domain.

```
MessageTemplateSpecification msgTemplateSpec = new MessageTemplateSpecification();
msgTemplateSpec.setDomainName("Repository");
msgTemplateSpec.setTypeName("FolderContentEvent");
msgTemplateSpec.setProtocolType(ProtocolType.SMTP);
msgTemplateSpec.setStrict(false);
msgTemplateSpec.addSubscriptionIdentifier("000000000000000000000000000001fb");

MessageTemplate messageTemplate = stub.findMessageTemplate(msgTemplateSpec);

System.out.println("Template: " + messageTemplate.getTemplateName());
System.out.println("Provider Type: " + messageTemplate.getTemplateProviderType().toString());
System.out.println("Path to template: " + messageTemplate.getPath());
MessageTemplateContent templateContent = messageTemplate.getMessageTemplateContent();
System.out.println("Template Content:");
System.out.println(templateContent.getContent());
```

SOAP request example

Client invocation of the findMessageTemplate 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>
  <findMessageTemplate xmlns="http://xml.spss.com/notification/remote">
   <ns2:messageTemplateSpecification protocolType="smtp" strict="false" domainName="Repository"
      typeName="FolderContentEvent" templateProviderType="database"
      xmlns:ns2="http://xml.spss.com/notification">
    <ns2:subscriptionIdentifier>000000000000000000000000000000001fb</ns2:subscriptionIdentifier>
   </ns2:messageTemplateSpecification>
  </findMessageTemplate>
 </soapenv:Body>
</soapenv:Envelope>
```

SOAP response example

The server responds to a findMessageTemplate 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>
  <findMessageTemplateResponse xmlns="http://xml.spss.com/notification/remote">
   <ns1:messageTemplate templateName="folder_link.vm" protocolType="smtp" strict="true"
      domainName="Repository" typeName="FolderContentEvent" templateProviderType="file"
      xmlns:ns1="http://xml.spss.com/notification">
    <ns1:path>/Repository/FolderContentEvent/folder_link.vm</ns1:path>
    <ns1:messageTemplateContent>
     <ns1:content>#*
      The template is used to generate a notification message when changes to folder content
      are made
      (new file or file version)
```

```
          *#

          /mimeMessage/messageSubject=PASW Services: New content in
            ${ResourcePath}
          /mimeMessage/messageContent[text/html;charset=utf-8]=
          &lt;html&gt;
          &lt;head&gt;
          &lt;meta http-equiv=&quot;Content-Type&quot;
            content=&quot;text/html;charset=utf-8&quot;/&gt;
          &lt;/head&gt;
          &lt;body&gt;
          &lt;p&gt;The #if( $ActionType==&quot;FileVersionCreated&quot; ) new version of #end
          file &lt;b&gt;${ChildName}&lt;/b&gt; has been created in folder
          &lt;b&gt;${ResourcePath}&lt;/b&gt;.&lt;/p&gt;

          &lt;p&gt;To review the content of the file, go to &lt;a href='${ChildHttpUrl}'&gt;
            ${ChildHttpUrl}&lt;/a&gt;. &lt;/br&gt;&lt;/p&gt;
          &lt;hr&gt;&lt;p&gt;This is a machine-generated message. Please do not reply directly.
            If you do not wish to receive this notification, unsubscribe or contact your
            administrator.&lt;/p&gt;
          &lt;/body&gt;
          &lt;/html&gt;
        </ns1:content>
      </ns1:messageTemplateContent>
    </ns1:messageTemplate>
  </findMessageTemplateResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

The getNotificationProvider operation

Input fields

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

Table 4-12
Fields for getNotificationProvider

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

Return information

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

Table 4-13
Return Value

Type	Description
notificationProvider	A provider for the notification service.

Java example

The information available in the NotificationProvider object returned by the getNotificationProvider operation includes the following:

- The provider type
- The identifier for the provider
- Parameters for the provider
- The set of dependent notification objects for the provider

The following example uses the getType and getIdentifier methods to access notification provider characteristics.

```
String providerIdentifier = "0a0b32e0cfc420760000010ec3b57caf8252";
NotificationProvider nProvider = stub.getNotificationProvider(providerIdentifier);
System.out.println("Identifier " + nProvider.getIdentifier() +
   " corresponds to notification provider type " + nProvider.getType() + ".");
```

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 4-14
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 4-15
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 as desired.
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 getProviderTypes operation

Returns an array, possibly empty or null, of the types of the notification providers deployed into the given instance of the service.

Return information

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

Table 4-16
Return Value

Type	Description
providerTypeSpecifier[]	A type specifier for the provider deployed into the notification server.

Java example

The following sample retrieves all provider types in the system, sending the type and supported interfaces for each to the standard output stream.

```
ProviderTypeSpecifier[] ptSpecifier = stub.getProviderTypes();
for (int i = 0; i < ptSpecifier.length; i++) {
  System.out.println("Provider Type: " + ptSpecifier[i].getType());
  System.out.println("Supported Interfaces:");
  String[] iName = ptSpecifier[i].getInterfaceName();
  for (int j = 0; j < iName.length; j++) {
    System.out.println(iName[j]);
    System.out.println();
  }
}
```

SOAP request example

Client invocation of the getProviderTypes 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>
```

```
    <getProviderTypes xmlns="http://xml.spss.com/notification/remote"/>
  </soapenv:Body>
</soapenv:Envelope>
```

The getRecipientSet operation

Returns a set of the recipients generated by the deployed recipient provider.

Input fields

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

Table 4-17
Fields for getRecipientSet

Field	Type/Valid Values	Description
recipientSetSpecification	recipientSetSpecification	A specification for the provider to fetch a set of recipients for the given instance of the structured event.

Return information

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

Table 4-18
Return Value

Type	Description
recipientSet	A container for the notification recipients.

The getVersion operation

Returns the version number of the service.

Return information

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

Table 4-19
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 updateNotificationProvider operation

Updates an existing instance of a notification provider and optionally associates it with the specified notification elements.

Input fields

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

Table 4-20
Fields for updateNotificationProvider

Field	Type/Valid Values	Description
notificationProvider	notificationProvider	A provider for the notification service.

Java example

To update a notification provider:

1. Call the getNotificationProvider operation with a provider identifier to return the notification provider to be updated.
2. Set updated values for the notification provider as needed.
3. Supply the updateNotificationProvider operation with the revised notification provider.

   The following code updates the label for an existing notification provider and disables it.

```
String providerIdentifier = "0a0b32e0cfc420760000010ec3b57caf8252";
NotificationProvider nprovider = stub.getNotificationProvider(providerIdentifier);
nprovider.setLabel("Reporting");
nprovider.setEnabled(false);
stub.updateNotificationProvider(nprovider);
```

SOAP response example

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

Microsoft® .NET Framework-based clients

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

1. Add a Service Reference. For more information, see the topic Adding a service reference on p. 36.
2. Configure the web service endpoint. For more information, see the topic Configuring the web service endpoint on p. 38.
3. Programmatically configure the necessary endpoint behaviors. For more information, see the topic Configuring endpoint behaviors on p. 39.
4. Exercise the web service. For more information, see the topic Exercising the service on p. 39.

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

Adding a service reference

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

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

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

Service reference modifications

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

To modify a service reference:

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

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

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

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

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

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

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

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

Configuring the web service endpoint

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

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

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

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

Configuring endpoint behaviors

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

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

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

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

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

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

Exercising the service

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

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

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

The corrected code is as follows:

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

Appendix

B

Notices

This information was developed for products and services offered worldwide.

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

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

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

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

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

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

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

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

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

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

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

Trademarks

Index