

Building Notification-based Web Services

WSDL specification, BlackBerry MDS Studio Version 4.1 and BlackBerry MDS Services requirements

Part number: SWD_MDS(EN)-007.000

At the time of publication, this documentation is based on BlackBerry MDS Studio Version 4.1.

Send us your comments on product documentation: <https://www.blackberry.com/DocsFeedback>.

©2005 Research In Motion Limited. All Rights Reserved. The BlackBerry and RIM families of related marks, images, and symbols are the exclusive properties of Research In Motion Limited. RIM, Research In Motion, "Always On, Always Connected", the "envelope in motion" symbol, BlackBerry, and BlackBerry Enterprise Server are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries.

The BlackBerry device and/or associated software are protected by copyright, international treaties and various patents, including one or more of the following U.S. patents: 6,278,442; 6,271,605; 6,219,694; 6,075,470; 6,073,318; D445,428; D433,460; D416,256. Other patents are registered or pending in various countries around the world. Visit www.rim.com/patents.shtml for a list of RIM [as hereinafter defined] patents.

This document is provided "as is" and Research In Motion Limited and its affiliated companies ("RIM") assume no responsibility for any typographical, technical or other inaccuracies in this document. RIM reserves the right to periodically change information that is contained in this document; however, RIM makes no commitment to provide any such changes, updates, enhancements or other additions to this document to you in a timely manner or at all. RIM MAKES NO REPRESENTATIONS, WARRANTIES, CONDITIONS OR COVENANTS, EITHER EXPRESS OR IMPLIED (INCLUDING WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OF FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, MERCHANTABILITY, DURABILITY, TITLE, OR RELATED TO THE PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE REFERENCED HEREIN OR PERFORMANCE OF ANY SERVICES REFERENCED HEREIN). IN CONNECTION WITH YOUR USE OF THIS DOCUMENTATION, NEITHER RIM NOR ITS RESPECTIVE DIRECTORS, OFFICERS, EMPLOYEES OR CONSULTANTS SHALL BE LIABLE TO YOU FOR ANY DAMAGES WHATSOEVER BE THEY DIRECT, ECONOMIC, COMMERCIAL, SPECIAL, CONSEQUENTIAL, INCIDENTAL, EXEMPLARY OR INDIRECT DAMAGES, EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, INCLUDING WITHOUT LIMITATION, LOSS OF BUSINESS REVENUE OR EARNINGS, LOST DATA, DAMAGES CAUSED BY DELAYS, LOST PROFITS, OR A FAILURE TO REALIZE EXPECTED SAVINGS.

This document might contain references to third party sources of information, hardware or software, products or services and/or third party web sites (collectively the "Third-Party Information"). RIM does not control, and is not responsible for, any Third-Party Information, including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third-Party Information. The inclusion of Third-Party Information in this document does not imply endorsement by RIM of the Third Party Information or the third party in any way. Installation and use of Third Party Information with RIM's products and services may require one or more patent, trademark or copyright licenses in order to avoid infringement of the intellectual property rights of others. Any dealings with Third Party Information, including, without limitation, compliance with applicable licenses and terms and conditions, are solely between you and the third party. You are solely responsible for determining whether such third party licenses are required and are responsible for acquiring any such licenses relating to Third Party Information. To the extent that such intellectual property licenses may be required, RIM expressly recommends that you do not install or use Third Party Information until all such applicable licenses have been acquired by you or on your behalf. Your use of Third Party Information shall be governed by and subject to you agreeing to the terms of the Third Party Information licenses. Any Third Party Information that is provided with RIM's products and services is provided "as is". RIM makes no representation, warranty or guarantee whatsoever in relation to the Third Party Information and RIM assumes no liability whatsoever in relation to the Third Party Information even if RIM has been advised of the possibility of such damages or can anticipate such damages

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>) and/or licensed pursuant to Apache License, Version 2.0 (<http://www.apache.org/licenses/>). For more information, see the NOTICE.txt file included with the software.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Research In Motion UK Limited
Centrum House, 36 Station Road
Egham, Surrey TW20 9LF
United Kingdom

Published in Canada

Contents

Overview	5
The current state of notification-based web services.....	5
Industry standardization.....	5
Audience.....	5
The web services definition.....	5
Import the WS-EVENTING namespace.....	6
Include a subscription operation	6
Include an unsubscription operation	8
Include a subscriptionEnd operation	8
Mark the port type for notifications	9
Structure the notification operation	9
The BlackBerry MDS Studio contract	11
Set an operation as a notification.....	12
Building the filter	12
Standard filter approach	12
Advanced filter approach.....	13
Determining the notification delivery mode.....	13
Keep last notification.....	13
Background processing of notification	13
Identifying the subscribe, unsubscribe and subscriptionEnd operations.....	14
Application products	15
Notification filter component.....	15
Notification data component	15
Notification message.....	16
Subscription message.....	16
Build the client subscription	16
Unsubscription message	18
Modifying notification data addressing.....	19
Modify a notification address	19
BlackBerry MDS Services contract.....	21
How to process subscription	21
The subscription message.....	22
Generating the subscription response.....	23
How to process unsubscriptions	24
How to deliver notifications	25

How to deliver subscriptionEnd	26
Appendix A: Modifying Price notification addressing.....	29
Glossary.....	33

Overview

The current state of notification-based web services

Most publicly available web services are based on a request/response paradigm. Notification-based web services are not widely available, likely due to a failure to adopt a standardized mechanism across web service toolkits. A widespread adoption would extend the interoperability of service-oriented architectures into the realm of asynchronous communication.

As a result of this failure to adopt a standardized notification capability, internet service providers are faced with developing their own ad hoc solutions or building to an emerging standard. Of the two available options, a platform technology such as BlackBerry Mobile Data System™ (BlackBerry MDS™) is better served by adhering as closely as possible to a standard.

Industry standardization

The clearest direction for standardization in this area is the Web Services-EVENTING (WS-EVENTING) standard, which primarily provides the language or schema to which WS-EVENTING compliant notification-based web services must adhere. A number of software companies have contributed to and used this standard.

BlackBerry MDS™ version 4.1 has embraced the WS-EVENTING standard as a means to describe the subscription for services through the Web Services Description Language (WSDL). Both BlackBerry MDS Studio™ and BlackBerry MDS Services rely on the WS-EVENTING standard in the following ways:

- BlackBerry MDS Studio looks for patterns in the WSDL to detect notifications.
- BlackBerry MDS Studio creates subscription and notification messages to reveal the subscription paradigm to the application in a simplified way.
- BlackBerry MDS Services constructs subscription and unsubscription Simple Object Access Protocol (SOAP)-based messages on the WS-EVENTING specification.
- BlackBerry MDS Services has requirements for subscription management and for the information that is required on the notification.

This document describes the implementation requirements for developing a web service that works with the BlackBerry MDS Version 4.1 platform. This document's secondary goal is to show how to use BlackBerry MDS Studio to create a client that supports notification.

Audience

This information is designed for developers who already have a good understanding of both web service and WSDL concepts and constructs. The information is related to BlackBerry Enterprise Server version 4.0 and later software and to BlackBerry Device Software version 4. Despite your level of expertise, you should read both the *BlackBerry MDS Studio - Getting Started Guide* and the *BlackBerry MDS Studio - Developer Guide* before starting this document.

If you are concerned primarily with building the web service, see "The web services definition" on page 5 and "BlackBerry MDS Services contract" on page 21 for more information.

Developers of the application client should see "The BlackBerry MDS Studio contract" on page 11 for more information.

This material does not apply to BlackBerry Connect devices.

The web services definition

The web service definition describes how to create a suitable WSDL for describing a service that works with BlackBerry MDS Version 4.1. The web service definition includes the following five tasks:

- Import the WS-EVENTING namespace

- Include a subscription operation
- Include an unsubscription operation
- Include a subscriptionEnd operation
- Mark the port type for notifications

Note: The WS-EVENTING standard requires that document-style WSDL be used when creating the web service.

Import the WS-EVENTING namespace

The notification-based web service must declare the WS-EVENTING namespace on the <definitions> element, so its data types and elements can be referred to elsewhere in the document. The WS-EVENTING namespace is found at <http://schemas.xmlsoap.org/ws/2004/08/eventing>.

The following code fragment shows how to declare the WS-EVENTING namespace.

```
<wsdl:definitions
  targetNamespace="http://sample.wsdl.subscription"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:ns7="http://schemas.xmlsoap.org/ws/2004/08/eventing"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

The <types> section of the WSDL must import the Eventing namespace or schema as shown in the following code fragment.

```
<wsdl:types>
  <schema
    elementFormDefault="qualified"
    targetNamespace="http://sample.wsdl.subscription"
    xmlns="http://www.w3.org/2001/XMLSchema">

    <xsd:import
      namespace="http://schemas.xmlsoap.org/ws/2004/08/eventing"
      schemaLocation="http://schemas.xmlsoap.org/ws/2004/08/eventing/eventing.wsdl"/>
```

Include a subscription operation

The developer of the web service must decide if one subscription will be used for all notifications or if a separate subscription will be configured for each notification type. When a single subscription is identified for all notifications, the subscription filter must include a field that determines which notification the subscription belongs to.

Note: If you use multiple subscription operations, it might be worthwhile to declare each subscription, unsubscription, and notification operation in separate port types. This method helps remove ambiguity when addressing the subscription operation in document-style WSDL.

The following code fragment shows the messages and operations to support subscription.

```
1 <wsdl:definitions
2   targetNamespace="http://sample.wsdl.subscription"
3   xmlns:ns7="http://schemas.xmlsoap.org/ws/2004/08/eventing"...>
4   <wsdl:types>
```

```
.
.
5  </wsdl:types>

6  <wsdl:message name="unsubscribeGasPrice">
7    <wsdl:part element="ns7:Unsubscribe" name="Unsubscribe_1"/>
8  </wsdl:message>
9  <wsdl:message name="subscribeGasPrice">
10   <wsdl:part element="ns7:Subscribe" name="Subscribe_1"/>
11 </wsdl:message>
12 <wsdl:message name="subscribeGasPriceResp">
13   <wsdl:part element="ns7:SubscribeResponse" name="SubscribeResp_1"/>
14 </wsdl:message>
15 <wsdl:portType name="PriceNotifications" ns7:EventSource="true">
16   <wsdl:operation name="subscribeGasPrice">
17     <wsdl:input message="impl:subscribeGasPrice"/>
18     <wsdl:output message="impl:subscribeGasPriceResp"/>
19   </wsdl:operation>
20   <wsdl:operation name="unsubscribeGasPrice">
21     <wsdl:input message="impl:unsubscribeGasPrice"/>
22   </wsdl:operation>
23 </wsdl:portType>
24 <wsdl:binding
25   name="PriceNotificationsSoapBinding"
26   type="impl:PriceNotifications">
27   <wsdlsoap:binding
28     style="document"
29     transport="http://schemas.xmlsoap.org/soap/http"/>
30   <wsdl:operation name="subscribeGasPrice">
31     <wsdlsoap:operation
32       soapAction="http://sample.wsdl.subscription/subscribeGasPrice"/>
33     <wsdl:input>
34       <wsdlsoap:body use="literal"/>
35     </wsdl:input>
36     <wsdl:output>
37       <wsdlsoap:body use="literal"/>
38     </wsdl:output>
39   </wsdl:operation>
40   <wsdl:operation name="unsubscribeGasPrice">
41     <wsdlsoap:operation
42       soapAction="http://sample.wsdl.subscription/unsubscribeGasPrice"/>
43     <wsdl:input>
```

```

44         <wsdlsoap:body use="literal" />
45     </wsdl:input>
46 </wsdl:operation>
47 </wsdl:binding>
48 <wsdl:service name="SampleIFService">
    .
    .
    .
49 </wsdl:service>
50 </wsdl:definitions>

```

WSDL output messages (lines 12, 18)

WSDL input messages (lines 17, 21)

WS-EVENTING related operations (lines 16, 20)

WS-EVENTING elements brought in by the import (lines 7, 10, 13)

Include an unsubscription operation

As with the subscription operation, there can be one or more unsubscription operations. See "Include a subscription operation" on page 6 for a code fragment that uses the WS-EVENTING unsubscribe operation.

Include a subscriptionEnd operation

The subscriptionEnd operation is optional in a WS-Eventing implementation, but it is supported by MDS Services™ and may be included in your web service.

The SubscriptionEnd operation is a special type of notification that is sent by the event source to the event sink informing the event sink that a particular subscription has ended.

To support this type of notification, include a subscriptionEnd message, define an operation, and indicate the binding as shown in the following code fragments:

```

<wsdl:message name="subscriptionEndOp">
    <wsdl:part element="wse:SubscriptionEnd" name="body" />
</wsdl:message>

```

In the <portType> element that is dedicated to the correspondent subscription, define the operation:

```

<wsdl:operation name="subscriptionEndOp">
    <wsdl:output message="impl:subscriptionEndOp" name="subscriptionEndOp" />
</wsdl:operation>

```

Indicate the binding for the same <portType> element:

```

<wsdl:operation name="subscriptionEndOp">
    <wsdlsoap:operation
        soapAction="http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd" />
    <wsdl:output name="subscriptionEndOp">
        <wsdlsoap:body use="literal" />
    </wsdl:output>
</wsdl:operation>

```

```
</wsdl:operation>
```

Mark the port type for notifications

The `<portType>` element in which the subscription and notification operations are declared should include the `EventSource="true"` extension as shown in the following code fragment.

```
<wsdl:portType name="PriceNotifications" ns7:EventSource="true">
```

Structure the notification operation

A notification can be considered to be an asynchronous response from the web service arising from some stimulus such as filter-matching criteria.

From the perspective of the WSDL, the purest expression of this structure is an operation that returns a message without accepting one. However, some WSDL processing tools might not recognize this structure as valid. As a result, the BlackBerry Mobile Data System version 4.1 is designed to recognize a second pattern. In this pattern, an empty input message is defined on the operation, a message that has no part. The BlackBerry MDS Studio is designed to recognize either of these patterns when building notifications.

Example: The following WSDL code fragment illustrates the definition of the notification operation. The second pattern has been applied, in which the notification input message has no part. Notice that the response data declared on the output message, "Price", is defined in the WSDL types section.

```
1 <wsdl:definitions
2   targetNamespace="http://sample.wsdl.subscription"...>
3   <wsdl:types>
4     <schema elementFormDefault="qualified"
5       targetNamespace="http://sample.wsdl.subscription"
6       xmlns="http://www.w3.org/2001/XMLSchema">
7       <xsd:import
8         namespace="http://schemas.xmlsoap.org/ws/2004/08/eventing"
9         schemaLocation="http://schemas.xmlsoap.org/ws/2004/08/eventing/eventing.wsdl"/>
10      <!-- Complex types -->
11      <complexType name="StationType">
12        <sequence>
13          <element name="regionNo" type="xsd:int"/>
14          <element name="vendorNo" type="xsd:int"/>
15        </sequence>
16      </complexType>
17      <complexType name="StationDesc">
18        <sequence>
19          <element name="intersection" nillable="true" type="xsd:string"/>
20          <element name="stationType" nillable="true" type="impl:StationType"/>
21        </sequence>
22      </complexType>
23      <complexType name="Station">
24        <sequence>
25          <element name="stationDesc" nillable="true" type="impl:StationDesc"/>
```

```
26         <element name="stationId" type="xsd:int"/>
27     </sequence>
28 </complexType>
29 <complexType name="Price">
30     <sequence>
31         <element name="ceilingPrice" type="xsd:double"/>
32         <element name="floorPrice" type="xsd:double"/>
33         <element name="observedBy" nillable="true" type="xsd:string"/>
34         <element name="observedOn" nillable="true" type="xsd:dateTime"/>
35         <element name="price" type="xsd:double"/>
36         <element name="station" nillable="true" type="impl:Station"/>
37     </sequence>
38 </complexType>
39 <element name="p" type="impl:Price"/>
40 </schema>
41 </wsdl:types>

42 <!-- Messages to support notification, empty input message -->
43 <wsdl:message name="gasPriceResponse">
44     <wsdl:part element="impl:p" name="p"/>
45 </wsdl:message>
46 <wsdl:message name="gasPriceRequest">
47 </wsdl:message>

48 <!-- Notification porttype -->
49 <wsdl:portType name="PriceNotifications" ns7:EventSource="true">
50     .
51     .
52     .
53 <wsdl:operation name="gasPrice" parameterOrder="">
54     <wsdl:input message="impl:gasPriceRequest" name="gasPriceRequest"/>
55     <wsdl:output message="impl:gasPriceResponse" name="gasPriceResponse"/>
56 </wsdl:operation>
57 </wsdl:portType>
58 <wsdl:binding name="PriceNotificationsSoapBinding"
59     type="impl:PriceNotifications">
60     .
61     .
62 <wsdl:operation name="gasPrice">
63     <wsdlsoap:operation soapAction=""/>
64     <wsdl:input name="gasPriceRequest">
65         <wsdlsoap:body use="literal"/>
```

```
61     </wsdl:input>
62     <wsdl:output name="gasPriceResponse">
63         <wsdlsoap:body use="literal"/>
64     </wsdl:output>
65 </wsdl:operation>
66 </wsdl:binding>
67 <wsdl:service name="SampleIFService">
68     .
69     .
68 </wsdl:service>
69 </wsdl:definitions>
```

WSDL output messages (lines 52, 43)

WSDL input messages (lines 51, 46)

The notification operation (line 50)

Type the notification is based on (lines 39, 29, 23, 17, 11)

The BlackBerry MDS Studio contract

This section describes how BlackBerry MDS Studio works with notification-based web services.

Note: The notification-based WSDL can provide information about whether or not the web service contains asynchronous messages to the BlackBerry MDS Studio. The developer must know the web service in order to pick notifications and to build the subscription filter.

Set an operation as a notification

To set an operation as a notification, you can create a new project and follow the steps in the Bottom Up Approach wizard, or you can set the operation as a notification manually by performing the following actions:

Action	Procedure
Import the data source (WSDL) to the BlackBerry MDS Studio	<ol style="list-style-type: none"> 1. In the Navigation pane, expand an existing project. 2. Right click Data sources. 3. Click New Data Source.
Bind the operation to the application	<ol style="list-style-type: none"> 1. In the Data source Visualizer window, expand Operations. 2. Right-click the operation. 3. Click Bind this operation.
Set notification	<ol style="list-style-type: none"> 1. In the Data Source Visualizer window, expand Operations. 2. Right-click the operation. 3. Click Set as notification. <p>Note: If the WSDL operation does not conform to the WSDL notification requirements, the Set as notification option is not shown.</p>

See "Structure the notification operation" on page 9 for the WSDL notification requirements.

Building the filter

The first step in building a subscription is creating the filter. A filter is the criteria passed to the web service in the subscription that tells the server what information the client is interested in.

At design time, the filter is a space delimited by a set of tokens and replacement characters, for example,

```
PRICE=%price% AND STATION_ID=%stationId%
```

At runtime, the application passes the information required to populate the filter replacement fields to BlackBerry MDS Services. BlackBerry MDS Services populates the filter with the field values and embeds them into a WS-EVENTING style subscription that is sent to the web service. The web service uses the filter field values to determine when a notification should be sent.

The WSDL does not explicitly state what the set of possible filter fields can be. The BlackBerry MDS Studio uses two approaches to determine the fields that can be included in the filter:

- analysis of pushed data type (standard)
- developer knowledge (advanced)

Note: Regardless of which approach is used, the fields identified in the filter have an impact on the components generated by BlackBerry MDS Studio to support the subscription. See "Application products" on page 15 for more information.

Standard filter approach

BlackBerry MDS Studio analyzes the data that is passed on the notification response message definition. This data definition represents the important fields in the notification. These fields are presented in the Notification

wizard by default and can be combined into the filter expression. See "Structure the notification operation" on page 9 for more information.

Going back to the example in the Structure the notification operation section, the notification message `gasPriceResponse` delivers the `Price` complex type. The fields defined on this type include:

- `ceilingPrice`
- `floorPrice`
- `observedBy`
- `observedOn`
- `price`
- `stationId` (on the `Station` complex type)
- `intersection` (on the `StationDesc` complex type)
- `regionNo` (on the `StationType` complex type)
- `vendorNo` (on the `StationType` complex type)

Where there are nested complex types in the definition of **Price**, primitive fields of those structures would also be included for building the filter.

Advanced filter approach

If the developer knows that the web service processes fields that are not embodied in the notification data, these fields can be manually added by typing the field name in the **Field** section of the first page of the Notification wizard.

Determining the notification delivery mode

The following two settings control how the application processes received notifications:

- keep last
- background processing

These settings influence how notifications are processed by the BlackBerry MDS Runtime™.

Keep last notification

Keep last notification means that only the latest notification message will be processed. When messages are queued by the BlackBerry MDS Runtime, each subsequent notification replaces the prior message before it. As a result, only the most recent notification is processed when regular message consumption resumes.

Such queuing of messages can occur if the application is not running or is running in the background without the background processing setting turned on.

When the **Keep Last** check box is cleared, all received messages are queued and processed at the first opportunity. This mode of operation generally results in a higher cost on message processing than when keep last is indicated. If the notification content in every message simply refreshes the current view of the subscription, the **Keep Last** check box should be selected. For example, in a subscription that refreshes a stock price periodically, if a history is not desired, likely only the last notification is relevant. The use of this setting is at the discretion of the developer and must be considered within the context of the application's functionality.

Background processing of notification

Background processing means that notifications are processed when the application is not available. This can be the case if the application is not running, or it is running but is not the currently selected application.

Background processing is limited to implementing message mappings; if a mapping has been identified in the message editor, the application data will be updated. See the *Getting Started Guide* or the *BlackBerry MDS Developer's Guide* for more information about message mapping.

Note: Background processing does not process a script.

Additionally, you can define an alert when background processing is used. An alert is defined in the BlackBerry MDS Studio message editor.

When the **Background Processing** check box is cleared, all received notification messages are queued until the application is activated or is moved to the foreground.

Identifying the subscribe, unsubscribe and subscriptionEnd operations

The application developer must identify both subscription and unsubscription operations for the notification operation. This identification is enforced by BlackBerry MDS Studio.

BlackBerry MDS Studio attempts to automatically set the subscribe and unsubscribe operations by looking for naming patterns. The developer can, and in some cases must, set the appropriate subscription and unsubscription operations manually. This is done on page 2 of the Notification wizard.

Action	Procedure
Designate a WSDL operation as the subscribe operation	<ol style="list-style-type: none"> 1. In page 2 of the Notification Wizard, select the desired WSDL operation 2. Right click to produce the context menu and select Set Subscribe Op
Designate a WSDL operation as the unsubscribe operation	<ol style="list-style-type: none"> 1. In page 2 of the Notification Wizard, select the desired WSDL operation 2. Right click to produce the context menu and select Set UnSubscribe Op

Although the subscriptionEnd operation is optional, it is automatically identified by BlackBerry MDS Studio if it is present in the WSDL. If it is not automatically identified, the developer can manually designate a subscriptionEnd operation.

Application products

When the Notification wizard is completed, BlackBerry MDS Studio generates the structures to support subscription, unsubscription, and notification delivery.

Notification filter component


BlackBerry MDS Studio generates a filter data component definition that carries all the fields that were identified when the filter was constructed. The filter data component type is declared on the message fields for related subscription messages. This subscription message is designed to provide all the required information for BlackBerry MDS Services to subscribe to the web service. See "Building the filter" on page 12 for more information.

The filter data component is defined with a primary key field ID, which results in a collection of filters. The **ID** field is a unique value created by the application or collected from the user through the User Interface. It is used to identify and enable multiple subscriptions that are based on the same filter type.

Notification data component

The BlackBerry MDS Studio generates a data component to hold the notification data. This component contains the data type that models the notification data, and an **ID** field. The **ID** field is the same ID used in the original subscription; at runtime the value of the field for this notification will be identical to that provided when the subscription was formulated. See "Build the client subscription" on page 16 for detail on how this **ID** is provided.

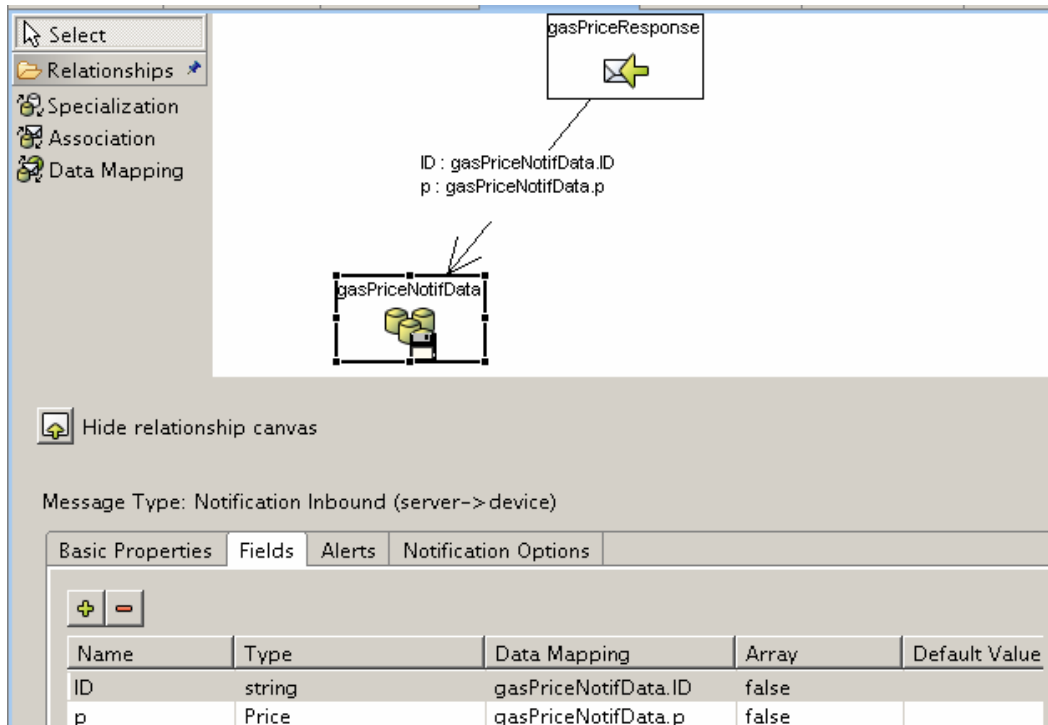
For example, the Price complex type from the preceding WSDL (see "Structure the notification operation" on page 9) might result in the data component called **gasPriceNotifData** (refer to the following figure showing the fields of data component **gasPriceNotifData**). The **gasPriceNotifData** includes the primary key field **ID**, and the notification data field type **Price**.

Name	Type	Array	Default Value
 ID	string	false	
p	Price	false	

Notification message

The BlackBerry MDS Studio generates a message that passes the required notification information. By default, a mapping is created that orients the message reception to the keyed collection. See "Notification filter component" on page 15 for more information.

The following graphic illustrates the generated messages and mappings for notification data. In this example, the **Price** notification results in a message called **gasPriceResponse** that maps to the keyed data collection **gasPriceNotifData**.



The **ID** field of this message is a special field inserted by BlackBerry MDS Services for transmitting the notification to the device. The **ID** field enables correlation between each notification and the original subscription that triggered it. The **ID** is not supplied by the web service.

Subscription message

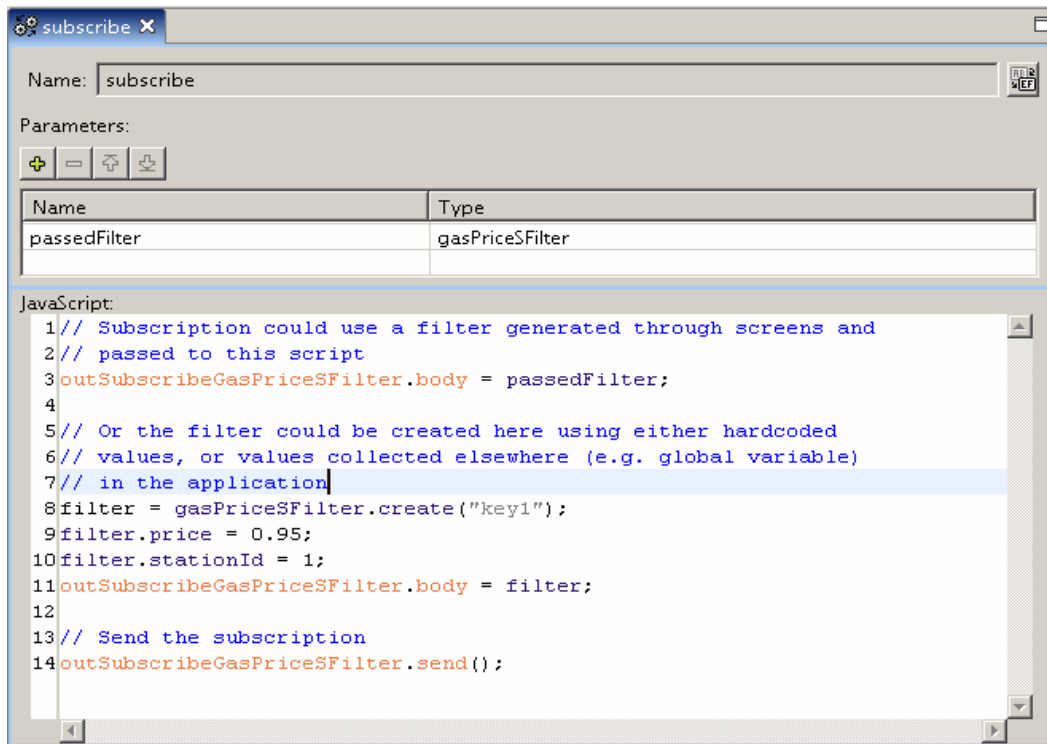
BlackBerry MDS Studio creates a subscription message that is used by the application to subscribe to the web service. The subscription message contains a field of the filter data component type. BlackBerry MDS Services processes this subscribe message and dynamically formulates the subscription filter by inserting these values into place-holders. This expression is then passed to the web service by BlackBerry MDS Services as a part of the WS-EVENTING style subscription. See "Building the filter" on page 12 for more information.

Build the client subscription

There are 4 steps to building subscription into your application. The following graphic illustrates these four steps, showing how they can be done using JavaScript.

1. Generate a unique key to identify the subscription (the filter **ID**). This value is used to create a new filter data component (line 8).
2. Assign the filter data component fields. The information to feed these fields could either be collected from the user in other screens, or in some cases may be "hard-coded" into the application (line 9, 10).
3. Assign the filter data component object to the subscription message (line 11).

4. Send the subscription message (line 14).

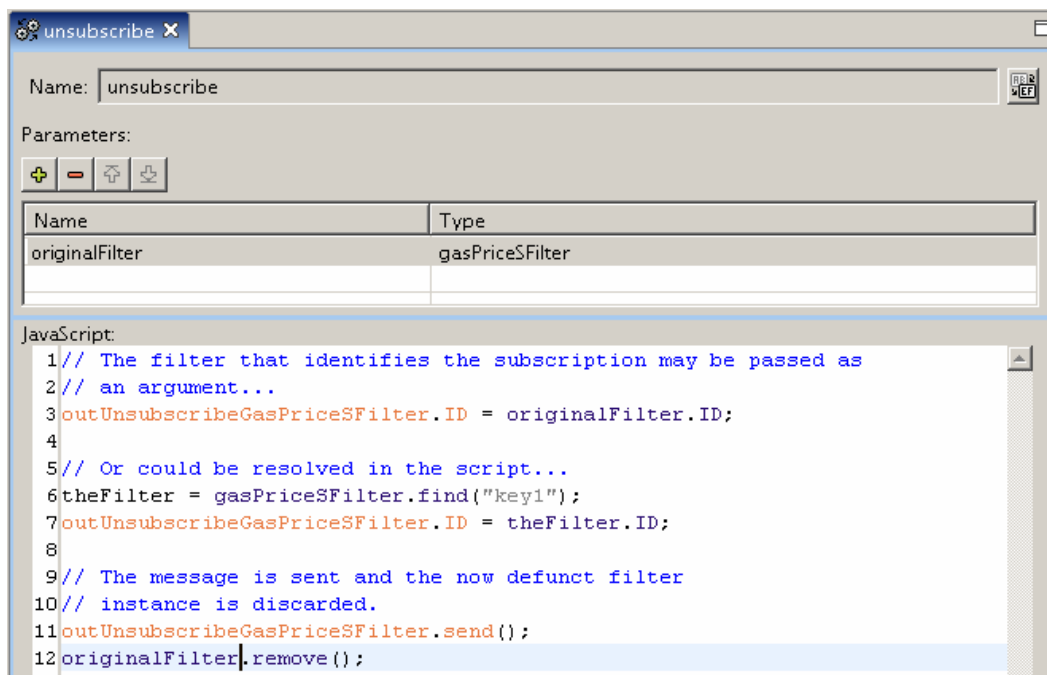


Unsubscription message

BlackBerry MDS Studio also generates an unsubscription message that enables the application to detach from notifications. Because the application can hold multiple subscriptions for the same filter definition (recall that the filter data component generated by BlackBerry MDS Studio is keyed), it is required to indicate which subscription is to be cancelled. This is specified through the filter **ID** field on the appropriate unsubscribe message.

There are 3 steps to building unsubscribe into your application. The following graphic illustrates these three steps, showing how they can be done using JavaScript.

1. The original filter ID must be assigned to the unsubscribe message. It may either be available as a passed parameter (line 3) or could be resolved explicitly through the script (as in line 6, 7).
2. Send the unsubscribe message (line 11).
3. Remove the filter data component for the subscription (line 12)



Name: unsubscribe

Parameters:

Name	Type
originalFilter	gasPriceSFilter

JavaScript:

```
1 // The filter that identifies the subscription may be passed as
2 // an argument...
3 outUnsubscribeGasPriceSFilter.ID = originalFilter.ID;
4
5 // Or could be resolved in the script...
6 theFilter = gasPriceSFilter.find("key1");
7 outUnsubscribeGasPriceSFilter.ID = theFilter.ID;
8
9 // The message is sent and the now defunct filter
10 // instance is discarded.
11 outUnsubscribeGasPriceSFilter.send();
12 originalFilter.remove();
```

Modifying notification data addressing

In some situations, the developer might want to address the notification data differently than is imposed by the default components generated by the Notification wizard. In this case, the notification message mappings can be reoriented to point the message data elsewhere. Consider the following two points when changing message mapping:

- preserve the **ID** field on the message.
- rebind the notification message to the data source using the Message Binding wizard.


The ID is always required on the notification message. When the existing mappings are removed, BlackBerry MDS Studio also removes the data source bindings because they might no longer be relevant.




This practice can be best illustrated through an example. Suppose that the application must store a series of price notifications and needs to keep the following information:



- `stationId`
- `price`
- `observedBy`
- `observedOn`
- `intersection`

The `stationId` is unique and will be used as a primary key for the new data component called **PriceWatch**. Because **PriceWatch** is keyed by `stationId`, the application can maintain price information for multiple stations. When a notification arrives, it updates the price and related information for that particular station.

Modify a notification address

Action	Procedure
Create storage for the notification data	<ol style="list-style-type: none"> 1. Store the notification data in either a global variable or data collection. Perform one of the following actions: <ul style="list-style-type: none"> • For a global variable, the type may be either a primitive type or a data component type. Proceed to the "Create a new global variable" action. • For collections that are the result of keyed data components, follow the "Define the data component structure" and "Create a data collection" actions.
Define the data component structure	<ol style="list-style-type: none"> 1. In the Project Navigator, expand the Data folder, right-click on the Definitions folder, and select New Data Definition. 2. Type a name the Name field. 3. In the Data Editor, select the Fields tab. 4. Add data component fields as required. <ol style="list-style-type: none"> a. Add a new field.  b. Type a new name in the Name field. c. Launch the Choose Data Type dialog the Type field.

	<p>d. Select the fields type.</p>
<p>Create a data collection</p>	<p>With the Data Editor positioned at the Fields tab:</p> <ol style="list-style-type: none"> 1. Select the field that will be the primary key. 2. Select the key action  to assign the key.
<p>Create a new global variable</p>	<ol style="list-style-type: none"> 1. In the Project Navigator, expand the Data folder, right-click on the Globals folder, and select New Global. 2. Type a new name in the Name field. 3. Assign the global variable Data Type through the "Choose Data Type" dialog. <ol style="list-style-type: none"> a. Launch the Choose Data Type dialog field  b. Select the desired type. <ul style="list-style-type: none"> • The type may be that of a data component (action 2). • The type may be a primitive type (boolean, string, decimal, integer, long).
<p>Remove the existing data component mapping</p>	<ol style="list-style-type: none"> 1. In the Project Navigator, double-click the notification message to open the message editor. 2. Select the Fields tab. <p>Note: The ID field indicates the subscription ID. See the "Notification message" section on page 16 for details. A secondary field carries the notification data. This is the information that is to be remapped.</p> <p>Warning: Do not remove the ID field.</p> 3. Delete  the Secondary notification data field. 4. Unmap the ID field. <ol style="list-style-type: none"> a. Select the Data Mapping column for the ID field, opening the Choose Data Mapping dialog box. b. Clear the Is Mapped check box.

<p>Map the message to the new storage</p>	<ol style="list-style-type: none"> 1. With the notification message open in the Message Editor, select Show Relationship Canvas  (if not already shown). 2. Drag the global variable or data collection to map to onto the canvas. 3. Select the Data Mapping  tool from the palette. 4. Click the message, drag the connector to the target storage and release, 5. Click the target to reveal a drop-down list of the fields to which the mapping may be applied. Select the desired target mapping field.
<p>Restore data source binding information</p>	<ol style="list-style-type: none"> 1. In the Navigator window, expand Messages. 2. Right-click the appropriate message and click Message Binding wizard. 3. In the Message Binding wizard component linkage page, assign MDS message fields to WSDL message fields. <ol style="list-style-type: none"> a. In the left pane, select the source field. b. In the right pane, select the type compatible WSDL field. c. Click Bind Field.

For an example of modifying notification data addressing, see "Appendix A: Modifying **Price** notification addressing" on page 29 which presents the steps in the context of the ongoing Price notification example.

BlackBerry MDS Services contract

There are several areas of the web service design that must be addressed in order to work with BlackBerry MDS Services:

- how to process subscriptions
- how to process unsubscriptions
- how to deliver notifications
- how to deliver subscriptionEnd

How to process subscription

A WS-EVENTING style subscription component is formulated by BlackBerry MDS Services and is sent to the web service when an application subscribes. The subscription SOAP contains a filter that specifies what data is of interest to the subscribing party.

The subscription message also contains other information that must be maintained by the web service in order for it to manage the subscription. The following list details the information that the web service must maintain.

- The web service must store the **subscription-id** that is included in the notification generated for this subscription.

- The web service must store the **notify-to** address that is used to indicate where to send the notification.
- The web service must store the **namespace** of the subscribing party. It is also required on the notification.
- The web service can store the filter information. The filter may be tested periodically according to changing states in the web service. The results of these tests determine whether or not a notification is warranted

The web service is required to return a subscription response message to BlackBerry MDS Services. This subscription response must contain a unique **cancellation-id** that is used by BlackBerry MDS Services to unsubscribe. As a result, the web service must store this **cancellation-id** and relate it to the original subscription id. The address to which unsubscribe is directed by BlackBerry MDS Services is also included in the subscription response.

Note: The Expiry property of the subscription response is currently ignored and does not have an effect on subscription lifetime as managed by BlackBerry MDS Services.

The subscription message

A typical subscription SOAP message is shown in the following code sample. This subscription message relates to the ongoing example dealing with gas price subscriptions.

```
1 POST /axis/services/PriceNotificationsPort HTTP/1.1
2 Content-Type: text/xml; charset=utf-8
3 SOAPAction: ""
4 User-Agent: Jakarta Commons-HttpClient/3.0-rc2
5 Host: localhost:8060
6 Content-Length: 1281

7 <?xml version="1.0" encoding="UTF-8"?>
8 <soapenv:Envelope
9   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
10  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
11  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
12 <soapenv:Body>
13   <Subscribe xmlns="http://schemas.xmlsoap.org/ws/2004/08/eventing">
14     <EndTo>
15       <ns1:Address
16         xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/08/addressing">
17         http://myhostname:8060/mds/SubscriptionEndListener
18       </ns1:Address>
19     <ns2:ReferenceProperties
20       xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/08/addressing">
21       <ns3:MySubscription
22         xmlns:ns3="http://www.rim.net/wica/ag/ws-eventing-event-sink">
23         ASAAAAAACHtMt6BwEAAff4//gKL3gkFgMih9OeIlnAHfCe
24       </ns3:MySubscription>
25     </ns2:ReferenceProperties>
26   </Subscribe>
```

```
27     </EndTo>
28     <Delivery>
29         <NotifyTo>
30             <ns4:Address
31                 xmlns:ns4="http://schemas.xmlsoap.org/ws/2004/08/addressing">
32                 http://myhostname:8060/mds/NotificationListener
33             </ns4:Address>
34             <ns5:ReferenceProperties
35                 xmlns:ns5="http://schemas.xmlsoap.org/ws/2004/08/addressing">
36                 <ns6:MySubscription
37                     xmlns:ns6="http://www.rim.net/wica/ag/ws-eventing-event-sink">
38                     ASAAAAAACHtMt6BwEAAff4//gKL3gkFgMih9OeIlnAHfCe
39                 </ns6:MySubscription>
40             </ns5:ReferenceProperties>
41         </NotifyTo>
42     </Delivery>
43     <Filter>stationId=5 and floorPrice+=50.0 and ceilingPrice-=95.0</Filter>
44 </Subscribe>
45 </soapenv:Body>
46 </soapenv:Envelope>
```

Note: The subscription id is available from the NotifyTo->ReferenceProperties->MySubscription element (line 38). The namespace (line 37) declared in the MySubscription element must also be stored because it is required to pass on the notification. The notify to address (line 32) is available from the NotifyTo->Address element. The filter (line 43) is available from the subscribe element.

Generating the subscription response

Processing the subscription requires a response that contains the unique **cancellation-id** (line 24). The cancellation id is generated by the web service and returned to BlackBerry MDS Services for future reference.

The address to which the unsubscription is to be sent is also specified (line 16). Most often, this is the same address on which the subscription/notification port of the service is declared.

The subscription id (line 22) and namespace (line 21) are reflected within the ReferenceParameters.

Note: The BlackBerry MDS Services implementation of WS-EVENTING chooses the MySubscription tag to hold this information. This part of the WS-EVENTING schema is loosely defined and is left to the implementation.

The following code fragment illustrates a subscription response.

```
1 HTTP/1.1 200 OK
2 Content-Type: text/xml;charset=utf-8
3 Transfer-Encoding: chunked
4 Date: Thu, 10 Nov 2005 15:29:28 GMT
5 Server: Apache-Coyote/1.1

6 <?xml version="1.0" encoding="utf-8"?>
```

```
7 <soapenv:Envelope
8   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
9   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
11  <soapenv:Body>
12    <SubscribeResponse xmlns="http://schemas.xmlsoap.org/ws/2004/08/eventing">
13      <SubscriptionManager>
14        <ns1:Address
15          xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/08/addressing">
16          http://myhostname:8060/axis/services/PriceNotificationsPort
17        </ns1:Address>
18        <ns2:ReferenceParameters
19          xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/08/addressing">
20          <ns3:MySubscription
21            xmlns:ns3="http://www.rim.net/wica/ag/ws-eventing-event-sink">
22            ASAAAAAACHtMt6BwEAAff4//gKL3gkFgMih9OeIlnAHfCe
23          </ns3:MySubscription>
24          <Identifier>46</Identifier>
25        </ns2:ReferenceParameters>
26      </SubscriptionManager>
27      <Expires>2006-11-10T15:29:28.415Z</Expires>
28    </SubscribeResponse>
29  </soapenv:Body>
30 </soapenv:Envelope>
```

How to process unsubscriptions

BlackBerry MDS Services passes the unsubscription to the web service when the application unsubscribes.

The unsubscribe message contains the original cancellation id **that is** generated by the web service. The web service should remove the record of this subscription and no longer send notifications for it.

The following code fragment illustrates a sample unsubscription message. The cancellation id (line 15) is provided to the web service through the Unsubscribe->Identifier element.

```
1 POST /axis/services/PriceNotificationsPort HTTP/1.1
2 Content-Type: text/xml; charset=utf-8
3 SOAPAction: ""
4 User-Agent: Jakarta Commons-HttpClient/3.0-rc2
5 Host: localhost:8060
6 Content-Length: 370

7 <?xml version="1.0" encoding="UTF-8"?>
8 <soapenv:Envelope
9   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
10  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```

11   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
12   <soapenv:Body>
13   <Unsubscribe
14       xmlns="http://schemas.xmlsoap.org/ws/2004/08/eventing">
15       <Identifier>46</Identifier>
16   </Unsubscribe>
17   </soapenv:Body>
18 </soapenv:Envelope>

```

How to deliver notifications

A notification is sent in response to matching application subscription criteria provided by the filter. A notification is tied to the subscription that initiated it. A notification is directed to BlackBerry MDS Services and is thereafter processed and sent to the application running on the device.

To process a notification the web service must:

- direct the notification to the BlackBerry MDS Services that originated the subscription
- include the original subscription id as a header of the SOAP (named MySubscription)
- include the objects that are declared in the WSDL as part of the notification response message

The following code fragment illustrates how the Price complex type is included in a notification.

```

1  POST /mds/NotificationListener HTTP/1.0
2  Content-Type: text/xml; charset=utf-8
3  Accept: application/soap+xml, application/dime, multipart/related, text/*
4  User-Agent: Axis/1.2.1
5  Host: dopey-yyz:8062
6  Cache-Control: no-cache
7  Pragma: no-cache
8  SOAPAction: ""
9  Content-Length: 1206

10 <?xml version="1.0" encoding="UTF-8"?>
11 <soapenv:Envelope
12     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
13     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
14     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
15   <soapenv:Header>
16     <nsl:MySubscription
17         xsi:type="xsd:string"
18         xmlns:nsl="http://www.rim.net/wica/ag/ws-eventing-event-sink">
19       ASAAAAAAAAACHtMt6BwEAAff4//gKL3gkFgMih9OeIlnAHfCe
20     </nsl:MySubscription>
21   </soapenv:Header>
22   <soapenv:Body>
23     <p

```

```

24     xsi:type="ns2:Price"
25     xmlns="http://sample.wsd1.subscription"
26     xmlns:ns2="http://sample.wsd1.subscription">
27     <ceilingPrice xsi:type="xsd:double">0.0</ceilingPrice>
28     <floorPrice xsi:type="xsd:double">0.0</floorPrice>
29     <observedBy xsi:type="xsd:string">ddebruin@rim.com</observedBy>
30     <observedOn xsi:type="xsd:dateTime">2005-04-15T04:00:00.000Z</observedOn>
31     <price xsi:type="xsd:double">79.9</price>
32     <station xsi:type="ns2:Station">
33         <stationDesc xsi:type="ns2:StationDesc">
34             <intersection xsi:type="xsd:string">Dixie and Matheson</intersection>
35             <stationType xsi:type="ns2:StationType">
36                 <regionNo xsi:type="xsd:int">16</regionNo>
37                 <vendorNo xsi:type="xsd:int">18</vendorNo>
38             </stationType>
39         </stationDesc>
40         <stationId xsi:type="xsd:int">5</stationId>
41     </station>
42 </p>
43 </soapenv:Body>
44 </soapenv:Envelope>

```

In the preceding code fragment, the original subscription id (line 19) is included as a parameter header on the SOAP message. The MySubscription element name (line 16) is again expected by BlackBerry MDS Services and the appropriate namespace is included (line 18). The Price object (lines 23 - 42) defined through the WSDL complex types is embedded in the SOAP body.

How to deliver subscriptionEnd

A web service can send a WS_EVENTING SubscriptionEnd message to BlackBerry MDS Services to indicate unexpected subscription termination.

The web service must:

- direct the SubscriptionEnd message to the BlackBerry MDS Services URL indicated in subscription request EndTo element
- include the original subscription id in the header of the SOAP message in the ReferenceProperties element as an element named MySubscription
- include the reason for cancellation

The following code fragment illustrates the subscriptionEnd message. Notice the inclusion of both the subscription id (line 8) and the MySubscription tag (line 7) to identify this information.

```

1 <soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/"
2   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"

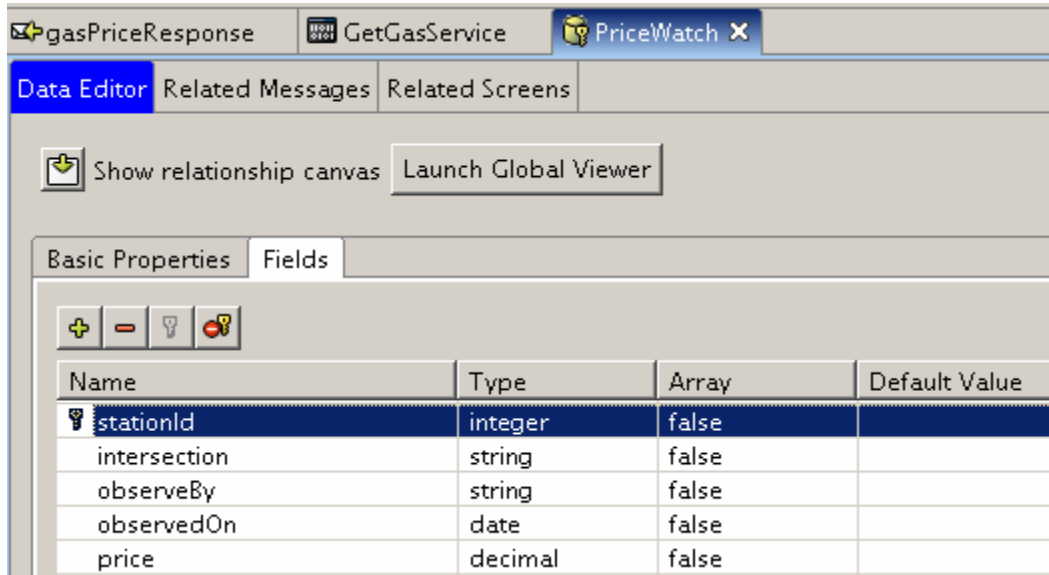
```

```
5   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
6   <soapenv:Header>
7     <ns1:MySubscription xmlns:ns1="http://www.rim.net/wica/ag/ws-eventing-event-sink">
8       shoAAAAAADBmYtdBwEAAOQT74JHUYFgPDKXjMdZrMR2ys8V
9     </ns1:MySubscription>
10  </soapenv:Header>
11  <soapenv:Body>
12    <wse:SubscriptionEnd>
13      <wse:Reason>
14        Discontinued
15      </wse:Reason>
16    </wse:SubscriptionEnd>
17  </soapenv:Body>
18 </soapenv:Envelope>
```


Appendix A: Modifying Price notification addressing

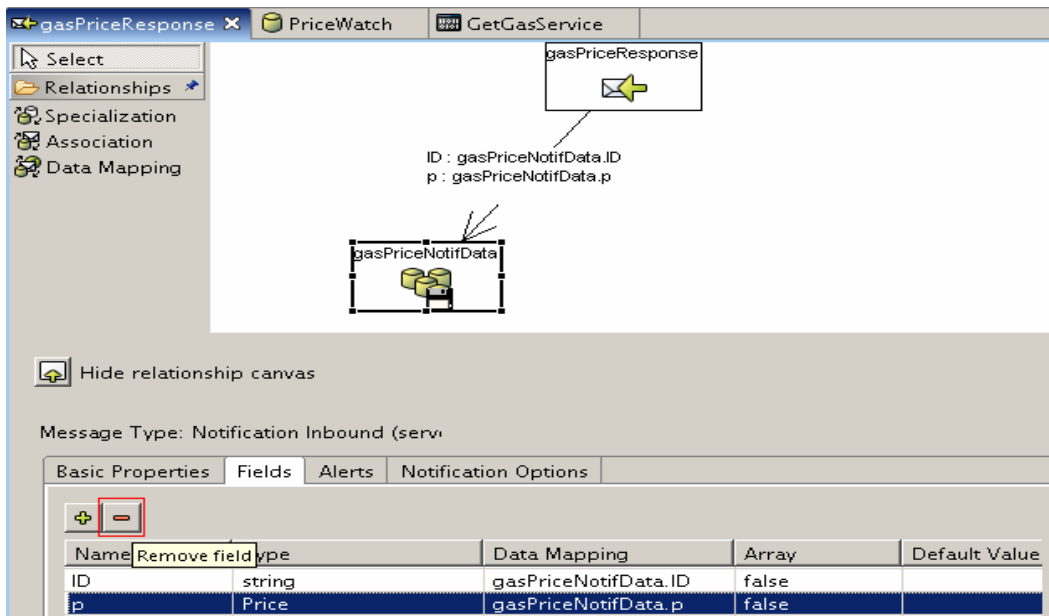
The following steps extend the Price notification example by illustrating the concepts of “Modifying notification data addressing” on page 19.


1. Create the PriceWatch component and set the primary key to `stationId`.

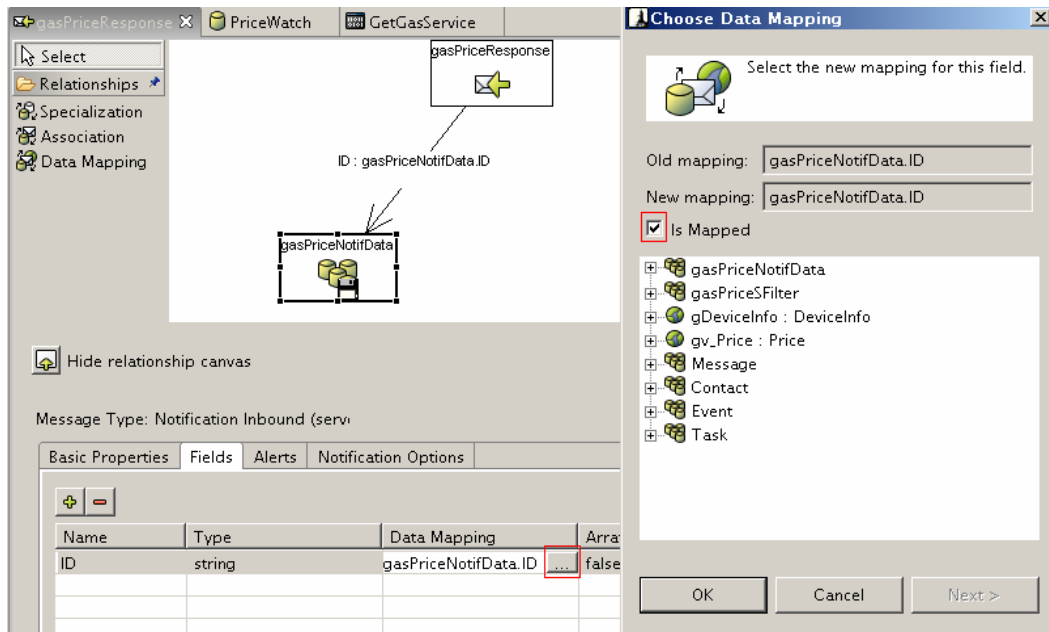


The PriceWatch data component has been created, and the `stationId` field is now designated as the primary key field.

2. Remove the existing message mapping, but NOT the ID field.

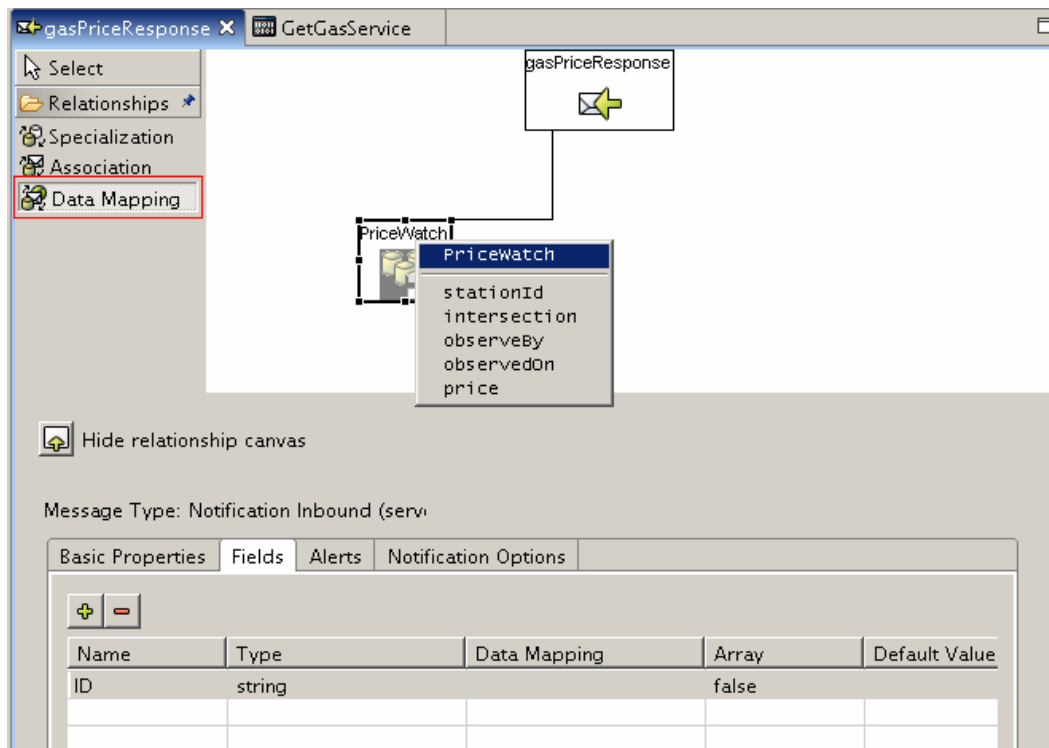


The Price type field `p` is removed using the Remove field () action.



The ID field Data Mapping is modified (...) and the Is Mapped check box is cleared.

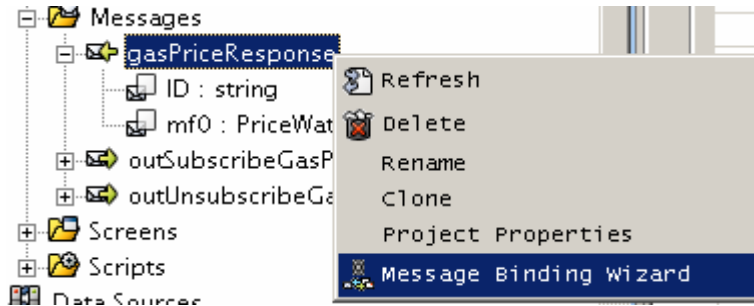
3. Map the message to the **PriceWatch** collection.



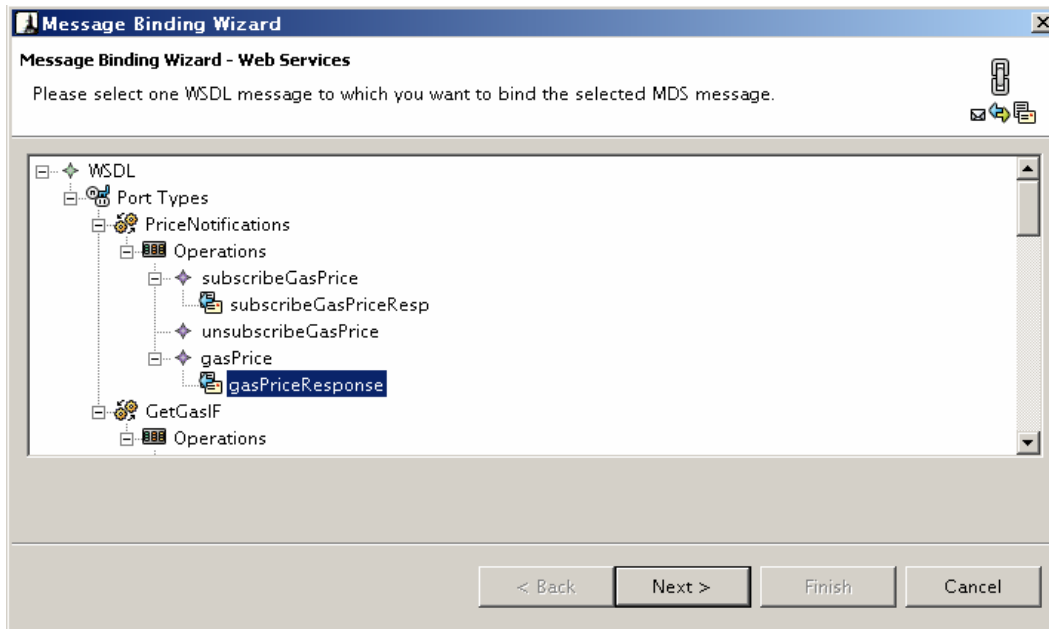
The Data Mapping tool is used to make a mapping to the PriceWatch collection.

The result is an additional field on the **gasPriceResponse** message of type **PriceWatch**.

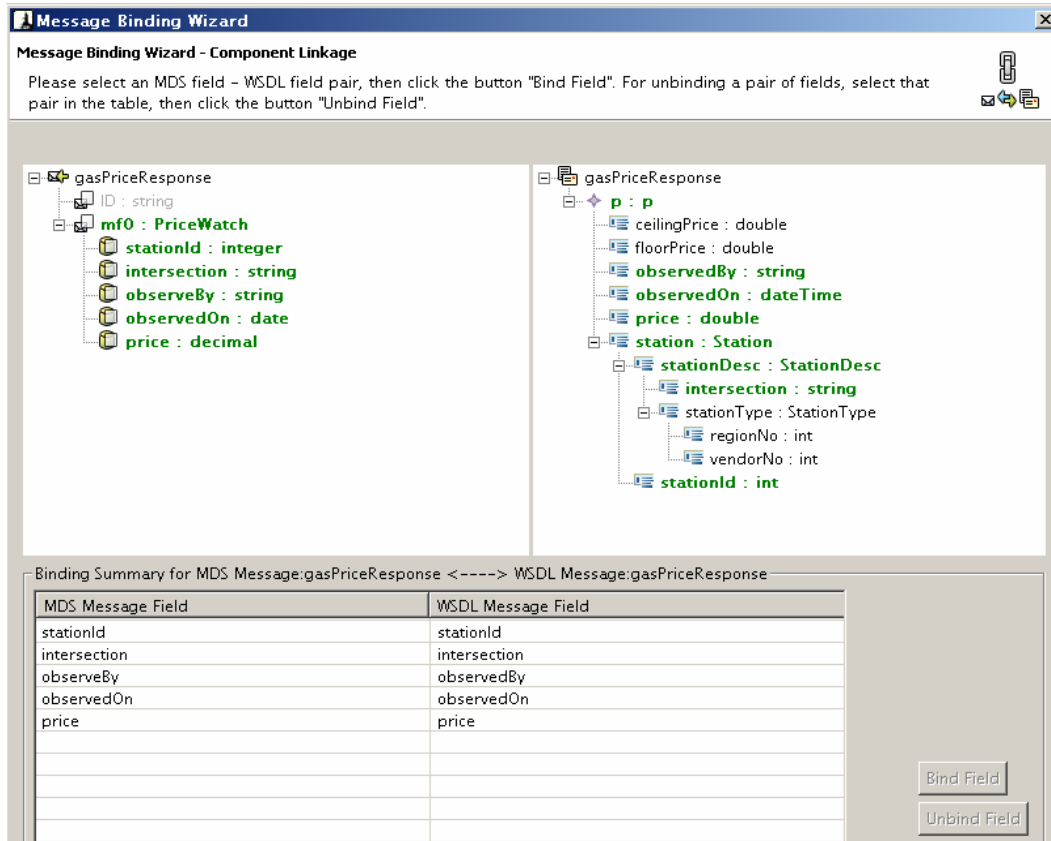
- Specify how the application message **gasPriceResponse** is bound to the data source. Since the structure defined in the data source (Price complex type) is different from the PriceWatch structure, the bindings must be specified in the context of fields. The Message Binding wizard is used, and the data source message **gasPriceResponse** is selected.



The Message Binding Wizard is launched from the context menu of the gasPriceResponse message.



The WSDL message gasPriceResponse is selected as the recipient of binding.



Each field of the BlackBerry MDS Studio gasPriceResponse is bound to the appropriate WSDL message field using Bind Field. Bound fields appear in green, whereas unbound fields remain black.

Note: The Message Binding Wizard recognizes the special **ID** field and prevents it from being involved in explicit bindings.

Glossary

E

Event Sink

A client (web service or Service Proxy) that receives notifications; in the BlackBerry MDS, the Event Sink role is assumed by BlackBerry MDS Services, which serves as a generic Service Proxy on behalf of notification-enabled BlackBerry MDS applications residing on wireless devices.

Event Source

A web service that sends notifications and accepts requests to create subscriptions

F

Filter

Defines the criteria controlling the notifications for which the Event Sink subscribes

S

Subscription

A message that allows the Event Sink to request notifications for a particular criteria

U

Unsubscription

A message that allows the Event Sink to remove itself from a particular subscription

W

Web Services-EVENTING (WS-EVENTING)

A standard based on WSDL that expresses the elements, types, and messages required to provide a subscription-based system through web services and SOAP