ORACLE

BPEL Tutorial

Tutorial 3: Manipulating XML Documents in BPEL

In a BPEL process, everything is XML, including the messages that are passed into and returned from the BPEL process, the messages that are exchanged with external services, and any local variables used by the flow itself. The types for all of these messages and variables are defined with XML Schema, usually in the WSDL file for the flow itself or in the WSDL files for services it invokes. Therefore, all variables in BPEL are XML documents, and any interesting BPEL process will spend a fair amount of its code manipulating those XML variables. Typically this includes doing data transformation between representations required for different services, as well as local manipulation of data (for example, to concatenate the results from several service invocations together). This document provides further details, primarily regarding the large part XPath plays in manipulating XML data.

Contents

Prerequisites

This document assumes you have some familiarity with BPEL, XPath, XML Schema, WSDL, and related Web service standards. You should have a basic understanding of BPEL processes and how to build them, such as what you can learn from the Oracle BPEL Designer tutorials "Developing a Hello World BPEL Process" (Tutorial1) and "Developing a Credit Flow BPEL Process" (Tutorial2). For additional background, or to augment the review provided by this document, refer to the related specifications that are listed on Oracle's developer documentation Web page (at http://otn.oracle.com/bpel).

> Note: Most of the examples in this document assume that the WSDL file defining the associated message types is document-literal style rather than the RPC style. Since there is a difference in how XPath query strings are formed for RPC-style WSDL definitions, if you are working with a type defined in an RPC WSDL file, you should consult the final section in this document.

Overview

The starting point for data manipulation in BPEL is the `<assign>` activity, which builds on the XPath standard. XPath queries, expressions, and functions play a large part in this type of manipulation, and more advanced methods are also available that involve using XQuery, XSLT, or Java (usually to do more complex data transformation or manipulation). This document will review the various methods from the simplest to the most advanced. The explanations will largely be by example, providing an excellent orientation to the supporting specifications without repeating their rigorous details.

The rest of this section provides a general overview of how to manipulate XML documents in BPEL. It summarizes the key building blocks that are then used in various combinations in examples that follow. The remaining sections discuss and illustrate how to apply these building blocks to perform specific tasks.

The `<assign>` activity enables you to copy data from one XML variable to another, or to calculate the value of an expression and store it in a variable. A `<copy>` element within the activity specifies the source and target of the assignment (what to copy from and to), which must be of compatible types. The formal syntax as shown in the BPEL specification is:

```
<assign standard-attributes>
   standard-elements
   <copy>+
      from-spec
      to-spec
   </copy>
</assign>
```

It is spelled out in detail in that specification. The `from-spec` and `to-spec` typically specify a variable or variable part, as in

```
<assign>
   <copy>
      <from variable="c1" part="address"/>
      <to variable="c3"/>
   </copy>
</assign>
```

Rather than repeat all the syntax details, this document will help orient you by showing and explaining excerpts taken primarily from sample projects provided by Oracle in the `C:\orabpel\samples\references` directory (if the Oracle BPEL Process Manager has been installed into the `C:\orabpel` directory).

> Note: As you may recall if you worked through the BPEL Designer tutorials, when you use the Designer to add an `<assign>` activity to your process, you supply details about the activity in a Copy Rule form that includes a **From** section and a **To** section, reflecting the underlying BPEL source code syntax shown above.

XPath enters heavily into the `<assign>` activity, as summarized below. Brief examples are shown here as an orientation; examples with more context and explanation are provided in the sections that follow.

- **XPath queries:** Where XPath most commonly comes into play is that an XPath query can be used to select a field within a source or target variable part. The `<from>` or `<to>` clause can include a `query` attribute whose value is an XPath query string, as in the following example:

  ```
  <from variable="input" part="payload"
        query="/CreditFlowRequest/ssn">
  ```

  For XPath 1.0, the value of the `query` attribute must be an absolute location path that selects exactly one node. Further details about the `query` attribute and XPath syntax can be found in the BPEL specification (section 14.3) and the XPath specification, respectively.

- **XPath expressions:** You can use an XPath expression (specified in an `expression` attribute in the `<from>` clause) to indicate a value to be stored in a variable. For example:

```
<from expression="100"/>
```

  The expression can be any general expression — that is, an XPath expression that evaluates to any XPath value type. For more information about XPath expressions, see section 9.1.4 of the XPath specification.

Within XPath expressions, you can call the following kinds of functions:

- **Core XPath functions:** XPath includes support for a large number of built-in functions, including functions for string manipulation (such as `concat`), numeric functions (like `sum`), and others.

```
<from expression="concat('string one', 'string two')"/>
```

  For a complete list of the functions built into XPath, see section 4 of the XPath specification.

- **BPEL XPath functions:** BPEL adds several extension functions to the core XPath core functions, enabling XPath expressions to access information from a process. The extensions are defined in the standard BPEL namespace http://schemas.xmlsoap.org/ws/2003/03/business-process/ and indicated by the prefix `bpws:`.

```
<from expression=
   "bpws:getVariableData('input', 'payload', '/value') + 1"/>
```

  For more information, see sections 9.1 and 14.1 of the BPEL specification.

- **BPEL XPath functions:** Oracle has provided some additional XPath functions for your use, using the capabilities built into BPEL and XPath for adding new functions. These functions are defined in the namespace http://schemas.oracle.com/xpath/extension and indicated by the prefix `ob:`. If you have the Oracle BPEL Process Manager installed locally, the BPEL Console will list and describe all available XPath functions via the page http://localhost:9700/BPELConsole/domain.jsp?mode=xpath.

- **Custom functions:** You can also create your own custom XPath functions. If you do, you need to register them in your BPEL process' deployment descriptor or in the file `C:\orabpel\domains\default\config\xpath-functions.xml` and bundle the source implementing them into a BPEL suitcase or the Oracle BPEL Process Manager startup environment. For more information about writing custom XPath functions, refer to http://otn.oracle.com/bpel.

Because sophisticated data manipulation can be painful to do with the BPEL `<assign>` activity and the core XPath functions, in practice many Oracle customers do their more complex data manipulation and transformation using XSLT, XQuery, or Java, or as a Web service. For more information regarding calling Java code from within BPEL, see the tutorial on that topic. For XSLT and XQuery, the Oracle BPEL Process Manager

includes XPath functions that will execute these transformations. For more information, see the XPath and XQuery transformation code examples at C:\orabpel\samples\tutorials\114.XSLTTransformations and C:\orabpel\samples\tutorials\115.XQueryTransformations or please submit your comments at http://otn.oracle.com/bpel.

With this background, you can now turn to the section below that covers the specific task or tasks you want to accomplish. We recommend that you read at least the first few sections because they show related definitions in the BPEL and WSDL files that help explain the examples; from there on you should be able to figure out how to track down the related definitions on your own.

Initializing a Variable with Literal XML

It is frequently useful to be able to assign literal XML to a variable in BPEL — for example, to initialize a variable before copying dynamic data into a specific field within the XML data content for the variable. This is also useful for testing purposes when you want to hardcode XML data values into your process.

Sample location: `C:\orabpel\samples\references\Assign`

BPEL code example: This example assigns a literal `<result>` element to the `payload` part of the `output` variable.

```
<assign>
   <!-- copy from literal xml to the variable -->
   <copy>
      <from>
         <result xmlns="http://samples.otn.com">
            <name/>
            <symbol/>
            <price>12.3</price>
            <quantity>0</quantity>
            <approved/>
            <message/>
         </result>
      </from>
      <to variable="output" part="payload"/>
   </copy>
</assign>
```

The variable `output` is defined in `Assign.bpel` as

```
<variable name="output" messageType="tns:AssignResultMessage"/>
```

The message type `AssignResultMessage` is defined in `Assign.wsdl` as

```
<message name="AssignResultMessage">
    <part name="payload" element="tns:result"/>
</message>
```

`Assign.wsdl` defines the `result` element to be made up of the sequence of elements (`name` through `message`) shown in the example above.

Simple Copying Between Variables

Although this type of manipulation is perhaps the least common, it serves as a good starting point for illustrating copying between variables or variable parts, because of its simplicity: it copies directly from one variable (or part) to another of compatible type, without needing to specify a particular field within either variable. In other words, there is no need to specify an XPath query.

Sample location: This example comes from section 9.3.2 of the BPEL specification.

BPEL code example: The example performs two assignments, first copying between two variable of the same type and then copying a variable part to another variable having the same type as that part.

```
<assign>
   <copy>
      <from variable="c1"/>
      <to variable="c2"/>
   </copy>
   <copy>
      <from variable="c1" part = "address"/>
      <to variable="c3"/>
   </copy>
</assign>
```

The BPEL file defines the variables as

```
<variable name="c1" messageType="x:person"/>
<variable name="c2" messageType="x:person"/>
<variable name="c3" element="x:address"/>
```

and the WSDL file defines the `person` message type as

```
<message name="person" xmlns:x="http://tempuri.org/bpws/example">
  <part name="full-name" type="xsd:string"/>
  <part name="address" element="x:address"/>
</message>
```

Accessing Fields Within Complex Type Variables

Given the kinds of definitions that are present in most WSDL files, you will very likely need to go down to the level of copying from or to a field within part of a complex type variable. To do this, you specify an XPath query in the `<from>` or `<to>` clause of the `<assign>` activity.

Sample location: `C:\bpelz\workspace\CreditFlow` and `C:\orabpel\samples\utils\CreditRatingService` (Tutorial2)

BPEL code example: This example copies the `ssn` field from the `CreditFlow` process's input message into the `ssn` field of the credit rating service's input message.

```
<assign>
   <copy>
      <from variable="input" part="payload"
         query="/CreditFlowRequest/ssn"/>
      <to variable="crInput" part="payload" query="/ssn"/>
   </copy>
</assign>
```

Notice that this example does not specify any namespaces in the queries (as in `tns:CreditFlowRequest`). Namespaces are optional in this syntax; they are required only when necessary to disambiguate the query — for example, when two siblings in an XML document have the same name but in different namespaces. Also note that the two fields here need not both have the same name, although they happen to both be named `ssn` in this example.

The BPEL file defines the variables involved in this assignment as

```
<variable name="input" messageType="tns:CreditFlowRequestMessage"/>
<variable name="crInput"
          messageType="services:CreditRatingServiceRequestMessage"/>
```

Regarding the `crInput` variable, which is used as an input message to a credit rating service, its message type, `CreditFlowRequestMessage`, is defined in `CreditFlowService.wsdl` as

```
<message name="CreditFlowRequestMessage">
   <part name="payload" element="tns:CreditFlowRequest"/>
</message>
```

and `CreditFlowRequest` is in turn defined to have a field named `ssn`.

The message type `CreditRatingServiceRequestMessage` is defined in `CreditRatingService.wsdl` as

```
<message name="CreditRatingServiceRequestMessage">
   <part name="payload" element="tns:ssn"/>
</message>
```

Assigning Numeric Values

Sample location: `C:\orabpel\samples\references\Assign`

BPEL code example: The following example illustrates the assignment of an XPath expression constituting the integer value `100`.

```
<assign>
   <!-- copy from integer expression to the variable -->
   <copy>
      <from expression="100"/>
      <to variable="output" part="payload" query="/result/quantity"/>
   </copy>
</assign>
```

Mathematical Calculations with XPath

You will frequently find the need to use simple mathematical expressions like the one in the following example, which increments a numeric value.

Sample location: `C:\orabpel\samples\references\Assign`

BPEL code example: The value being incremented is retrieved with the commonly used BPEL XPath function named `getVariableData`. The arguments to `getVariableData` are equivalent to the `variable`, `part`, and `query` attributes of the `<from>` clause (including that the last two arguments are optional).

```
<assign>
   <copy>
      <from expression="bpws:getVariableData('input', 'payload',
         '/value') + 1"/>
      <to variable="output" part="payload" query="/result"/>
   </copy>
</assign>
```

Assigning String Literals

Sample location: `C:\orabpel\samples\references\Assign`

BPEL code example: This example copies an expression evaluating to the string literal `'GE'` to the `symbol` field within the indicated variable part. (The trickiest thing here is to get the double and single quotes surrounding the string right.)

```
<assign>
   <!-- copy from string expression to the variable -->
   <copy>
       <from expression="'GE'"/>
       <to variable="output" part="payload" query="/result/symbol"/>
   </copy>
</assign>
```

Concatenating Strings

Rather than simply copy the value of one string variable (or variable part, or field) to another, you may first want to do some string manipulation, such as concatenating several strings together. If you worked through Tutorial 1, you may recall an example of this from the `SyncHelloWorld` project, in which the string `'Hello '` is concatenated with a name supplied in an input message. A similar example, from the `Assign` reference sample, is shown below. The concatenation is accomplished with the core XPath function named `concat`; in addition, the variable value involved in the concatenation is retrieved with the BPEL XPath function `getVariableData`.

Sample location: `C:\orabpel\samples\references\Assign`

BPEL code example: In this example, `getVariableData` is used to fetch the value of the `name` field from the `input` variable's `payload` part. The string literal `'Hello '` is then concatenated to the beginning of this value.

```
<assign>
   <!-- copy from XPath expression to the variable -->
   <copy>
      <from expression="concat('Hello ',
         bpws:getVariableData('input', 'payload', '/name'))"/>
      <to variable="output" part="payload" query="/result/message"/>
   </copy>
</assign>
```

Other string manipulation functions available in XPath are listed in section 4.2 of the XPath specification.

Assigning Boolean Values

Sample location: `C:\orabpel\samples\references\Assign`

BPEL code example: In this example, the XPath expression in the `<from>` clause is a call to XPath's boolean function `true`, and the specified `approved` field is set to true. The function `false` is also available.

```
<assign>
   <!-- copy from boolean expression function to the variable -->
   <copy>
      <from expression="true()"/>
      <to variable="output" part="payload" query="/result/approved"/>
   </copy>
</assign>
```

Assigning Date or Time

It is not uncommon to encounter a field that is defined to be a date, time, or both, in which case you can assign the appropriate current value by using the BPEL XPath function `getCurrentDate`, `getCurrentTime`, or `getCurrentDateTime`, respectively. In addition, if you have a date-time value in the standard XSD format, you can convert it to characters more suitable for output by calling the BPEL XPath function `formatDate`. (For related information, see section 9.1.2 of the BPEL specification.)

Sample location: `C:\orabpel\samples\references\XPathFunction`

BPEL code examples:

```
<!-- execute the XPath extension function getCurrentDate() -->
<assign>
   <copy>
      <from expression="ora:getCurrentDate()"/>
      <to variable="output" part="payload"
         query="/invoice/invoiceDate"/>
   </copy>
</assign>
```

In the next example, the `formatDate` function converts the date-time value provided in XSD format to the string `'Jun 10, 2003'` (and assigns it to the string field `formattedDate`).

```
<!-- execute the XPath extension function formatDate() -->
<assign>
   <copy>
      <from expression="ora:formatDate('2003-06-10T15:56:00',
         'MMM dd, yyyy')"/>
      <to variable="output" part="payload"
         query="/invoice/formattedDate"/>
   </copy>
</assign>
```

Manipulating Attributes

You will occasionally want to copy to or from something defined as an XML attribute. An at sign (@) in XPath query syntax is used to refer to an attribute instead of a child element.

Sample location: `C:\orabpel\samples\references\XPath`

BPEL code example: The code example that follows fetches and copies the `custId` attribute out of this XML data:

```
<payload>
   <invalidLoanApplication xmlns="http://samples.otn.com">
      <application xmlns = "http://samples.otn.com/XPath/autoloan">
         <customer custId = "111" >
            <name>
               Mike Olive
            </name>
            ...
         </customer>
         ...
      </application>
   </invalidLoanApplication>
</payload>
```

The example below selects the `custId` attribute of the `customer` field and assigns it to the variable `custId`.

```
<assign>
   <!-- get the custId attribute and assign to variable custId -->
   <copy>
      <from variable="input" part="payload"
            query="/tns:invalidLoanApplication/autoloan:application
                   /autoloan:customer/@custId"/>
      <to variable="custId"/>
   </copy>
</assign>
```

Note that the fact that this example uses names qualified with namespace prefixes is not integral to the example; the prefixes are in fact optional here, as described earlier.

The WSDL file defines `customer` to have a type in which `custId` is defined as an attribute, as follows:

```
<complexType name="CustomerProfileType">
   <sequence>
      <element name="name" type="string"/>
      ...
   </sequence>
   <attribute name="custId" type="string"/>
</complexType>
```

Manipulating Arrays

Arrays are used frequently in BPEL processes; however, manipulating them can be nontrivial. Based on XML Schema, the way you can identify an array definition is from its attribute `maxOccurs="unbounded"`; see the XML Schema specification for more information. The examples in this section illustrate several basic ways of manipulating arrays in BPEL, but there are other associated requirements, such as the need to perform looping or dynamic referencing of endpoints. For additional code samples and further information regarding real-world use cases for array manipulation in BPEL, visit http://otn.oracle.com/bpel.

Each of the following subsections describes a particular requirement for array manipulation. If you want to see a code example that puts it all together, you may want to look at `C:\orabpel\samples\tutorials\112.Arrays\ArraySample.bpel`, which takes an array as input and loops through it, adding together individual line items in each array element into a total value.

Statically Indexing into an Array

The two examples shown here illustrate how to use XPath to select an array element when the index of the element you want is known at design time. (In these cases, it is the first element.)

Sample location: `C:\orabpel\samples\references\XPath`

BPEL code examples: In the following example, `addresses[1]` selects the first element of the `addresses` array.

```
<assign>
   <!-- get the first address and assign to variable address -->
   <copy>
      <from variable="input" part="payload"
         query="/tns:invalidLoanApplication/autoloan:application
                /autoloan:customer/autoloan:addresses[1]"/>
      <to variable="address"/>
   </copy>
</assign>
```

In the query above, `addresses[1]` is equivalent to `addresses[position()=1]`, where `position` is one of the core XPath functions (see sections 2.4 and 4.1 of the XPath specification). The query in the next example calls the `position` function explicitly to select the first element of the `addresses` array, and then selects that address's `street` element (which the activity assigns to the variable `street1`).

```
<assign>
   <!-- get the first address's street and assign to street1 -->
   <copy>
      <from variable="input" part="payload"
         query="/tns:invalidLoanApplication/autoloan:application
                /autoloan:customer/autoloan:addresses[position()=1]
                /autoloan:street"/>
      <to variable="street1"/>
   </copy>
</assign>
```

If you track down the definition of the `input` variable and then its `payload` part in the WSDL file, you will go a few levels down before coming to the definition of the `addresses` field; there you will see the `maxOccurs="unbounded"` attribute.

The two XPath indexing methods are functionally identical, so you can use whichever you prefer.

Determining Array Size

Should you need to know the run-time size of an array — that is, the number of elements (nodes) in the array— you can get it by using the BPEL XPath function named `countNodes`. This functionality is commonly used when there is a need to iteratively process an entire array, starting with the first element and ending with the last element.

Sample location: `C:\orabpel\samples\references\XPathFunction`

BPEL code example: This example calculates the number of elements in the `item` array and assigns it to the integer variable `lineItemSize`.

```
<!-- execute the XPath extension function countNodes(arrayOfElements[])
  -->
<assign>
   <copy>
      <from expression="ora:countNodes('output', 'payload',
         '/invoice/lineItems/item')"/>
      <to variable="lineItemSize"/>
   </copy>
</assign>
```

Dynamically Indexing into an Array

Often a dynamic value is needed to index into an array — that is, you will want to get the *n*th element of an array, where the value of *n* is defined at run time. Two different ways to accomplish this are described below.

Dynamically indexing by Constructing an XPath at Run Time

The dynamic indexing method shown here is a two-step process: you use an XPath expression to dynamically create another XPath expression, which is then used to fetch the array element.

Sample location: `C:\orabpel\samples\tutorials\112.Arrays\ArraySample.bpel`

BPEL code example:

```
<variable name="iterator" type="xsd:integer"/>
...

<assign>
   <copy>
      <from expression="concat('/invoice/line-item[',
         bpws:getVariableData('iterator'), ']/line-total')"/>
      <to variable="xpath"/>
   </copy>
   <copy>
      <from expression="bpws:getVariableData('input', 'payload',
         bpws:getVariableData('xpath')) + ..."/>
   </copy>
</assign>
```

At run time, the variable `xpath` will hold a string value like

```
/invoice/line-item[2]/line-total
```

which is then used as an XPath query string in an expression — specifically, in the third argument of the outermost call to `getVariableData` in the second `<copy>` element above. The full code example shown in `ArraySample.bpel` illustrates how to iterate through an entire array, in this case to add up a series of individual `line-item` values into a total.

Dynamically Indexing with the BPEL getElement Function

With the Oracle BPEL Process Manager, a second way of dynamically indexing into an array is available that does not require the two-step process of creating an XPath query; instead, it uses the BPEL XPath function `getElement`. This function takes an array and an index (which can be a dynamic value, such as a variable) and returns the appropriate array element.

Sample location: `C:\orabpel\samples\references\XPathFunction`

BPEL code example:

```
<variable name="lineItemIndex" type="xsd:int"/>
...

<!-- execute the XPath extension function getElement(arrayOfElements[],
     index) to fetch one element from an array of elements
  -->
<assign>
   <copy>
      <from expression="ora:getElement('output', 'payload',
         '/invoice/lineItems/item',
         bpws:getVariableData('lineItemIndex'))"/>
      <to variable="myLineItem"/>
   </copy>
</assign>
```

Merging Arrays

A common requirement when working with arrays is to be able to take two arrays (of compatible types) and merge them into a single array. Oracle provides an XPath function named `mergeChildNodes` to simplify this task.

Sample location: `C:\orabpel\samples\references\XPathFunction`

BPEL code example:

```
<!-- execute the XPath extension function mergeChildNodes(e1, e2) and
     assign to a variable
  -->
<assign>
   <copy>
      <from expression="ora:mergeChildNodes(
        bpws:getVariableData('input', 'payload', '/invoice/lineItems'),
        bpws:getVariableData('literalLineItems'))"/>
      <to variable="mergedLineItems"/>
   </copy>
</assign>
```

Appending New Items to an Array

The BPEL XPath function `addChildNode` enables BPEL processes to append new elements to an existing array.

Sample location: `C:\orabpel\samples\references\XPathFunction`

BPEL code example:

```
<!-- execute the XPath extension function addChildNode(Element e,
     Node childNode)
  -->
<assign>
   <copy>
      <from expression="ora:addChildNode(bpws:getVariableData('output',
         'payload', '/invoice/lineItems'),
         bpws:getVariableData('escapedLineItem'))"/>
      <to variable="output" part="payload" query="/invoice/lineItems"/>
   </copy>
</assign>
```

Converting From a String to an XML Element

Sometimes a service is defined to return a string, but the content of the string will actually be XML data. The problem is that, although BPEL provides a lot of support for manipulating XML data (using XPath queries, expressions, and so on), this functionality is not available if the variable or field is of type `string`. In the Java world, you would use DOM functions to convert the string to a structured XML object type, and Oracle provides a built-in function that does the same kind of thing: the BPEL XPath function `parseEscapedXML` takes XML data, parses it via DOM, and returns structured XML data that can be assigned to a typed BPEL variable.

Sample location: `C:\orabpel\samples\references\XPathFunction`

BPEL code example:

```
<!-- execute the XPath extension function
     parseEscapedXML('&lt;item&gt;') and assign to a variable
  -->
<assign>
   <copy>
      <from expression="ora:parseEscapedXML(
         '&lt;item xmlns=&quot;http://samples.otn.com&quot;
                   sku=&quot;006&quot;&gt;
             &lt;description&gt;sun ultra sparc VI server
             &lt;/description&gt;
             &lt;price&gt;1000
             &lt;/price&gt;
             &lt;quantity&gt;2
             &lt;/quantity&gt;
             &lt;lineTotal&gt;2000
             &lt;/lineTotal&gt;
          &lt;/item&gt;')"/>
      <to variable="escapedLineItem"/>
   </copy>
</assign>
```

Differences for RPC-style WSDL Files

As mentioned previously, all of the examples shown here so far have been for document-style WSDL files, in which a message is defined with an XML Schema “element” as in the following example:

```
<message name="LoanFlowRequestMessage">
   <part name="payload" element="s1:loanApplication"/>
</message>
```

This is in contrast to RPC-style WSDL files, in which the message is defined with an XML Schema “type” as in

```
<message name="LoanFlowRequestMessage">
   <part name="payload" type="s1:LoanApplicationType"/>
</message>
```

This affects the material in this document because there is a difference in how XPath queries are constructed for the two WSDL message styles. For an RPC-style message, the top-level element (and therefore the first node in an XPath query string) is the part name (`payload` in the example above). In document-style, the top-level node is the element name (`loanApplication`).

Below is an example of what an XPath query string would look like if the LoanServices (for example, `C:\orabpel\samples\utils\AsyncLoanService`) used in BPEL demo applications (such as `C:\orabpel\samples\demos\LoanDemo\LoanFlow`) were RPC style.

RPC-style WSDL:

```
<message name="LoanServiceResultMessage">
   <part name="payload" type="s1:LoanOfferType"/>
</message>

<complexType name="LoanOfferType">
   <sequence>
      <element name="providerName" type="string"/>
      <element name="selected" type="boolean"/>
      <element name="approved" type="boolean"/>
      <element name="APR" type="double"/>
   </sequence>
</complexType>
```

BPEL:

```
<variable name="output"
          messageType="tns:LoanServiceResultMessage"/>

...

<assign>
   <copy>
      <from expression="9.9"/>
      <to variable="output" part="payload" query="/payload/APR"/>
   </copy>
</assign>
```