DZone Refcardz

ORACLE
FUSION MIDDLEWARE
BPEL PROCESS MANAGER

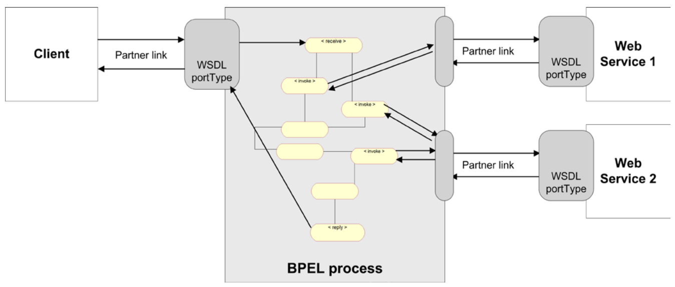
CONTENTS INCLUDE:

Core WS-BPEL: Business Process Execution Language

By Matjaz B. Juric

OVERVIEW

Each BPEL process consists of the following: (a) BPEL code, which defines the orchestration flow; (b) WSDL interface, which defines the interface (and the related XML Schemas) of the BPEL process for its clients; (c) WSDL interfaces of the consumed services (partner links). Partner links define the relations between BPEL process and the web services. Figure 1 shows the overall structure of a BPEL process.



OVERALL STRUCTURE OF A BPEL PROCESS

```
<process name="NCName" targetNamespace="anyURI"
         queryLanguage="anyURI"?
         expressionLanguage="anyURI"?
         suppressJoinFailure="yes|no"?
         exitOnStandardFault="yes|no"?
         xmlns="http://docs.oasis-
                open.org/wsbpel/2.0/process/executable">
   <extensions/>?
   <import/>*
   <partnerLinks/>?
   <messageExchanges/>?
   <variables/>
   <correlationSets/>?
   <faultHandlers/>?
   <eventHandlers/>?
   activity
</process>
```

activity	Two different types of processes: - Use <sequence> for sequential execution of process activities. - Use <flow> together with <links> for concurrent execution of process activities.

PARTNER LINKS

<plnk:partnerLinkType>

Characterizes a relationship between two services. We define roles played by each of the services. We specify the used portType of each service within the context of the conversation. Each <role> specifies exactly one WSDL portType. <plnk:partnerLinkType> is defined in the service WSDL document, not in the BPEL.

```
<wsdl:definitions name="NCName" targetNamespace="anyURI" ...>
   ...
   <plnk:partnerLinkType name="NCName">
      <plnk:role name="NCName" portType="QName" />
      <plnk:role name="NCName" portType="QName" />?
   </plnk:partnerLinkType>
   ...
</wsdl:definitions>
```

Example:

```
<plnk:partnerLinkType name="OrderLT">
   <plnk:role name="OrderService" portType="ord:OrderPT" />
   <plnk:role name="OrderRequester" portType="ord:OrderCallbackPT" />
</plnk:partnerLinkType>
```

<partnerLink>

Defines the relation of the BPEL process to partner web services. For request-reply semantics specify both roles, for one-way semantics specify one role only.

```
<partnerLinks>?
  <!-- At least one role must be specified. -->
  <partnerLink name="NCName"
     partnerLinkType="QName"
     myRole="NCName"?
     partnerRole="NCName"?
     initializePartnerRole="yes|no"?>+
  </partnerLink>
</partnerLinks>
```

Example:

```
<partnerLinks>
  <partnerLink name="Ordering"
      partnerLinkType="tns:OrderLT"
      myRole="OrderRequester" partnerRole="OrderService" />
</partnerLinks>
```

<sref:service-ref>

Endpoint references associated with partnerRole and myRole of <partnerLink>s are manifested as service reference containers (<sref:service-ref>).

```
<sref:service-ref reference-scheme="URI">
   content
</sref:service-ref>
```

Default is WS-Addressing endpoint reference:

```
<sref:service-ref>
  <addr:EndpointReference>
    <addr:Address>
      http://example.com/auction/Registration/
    </addr:Address>
    <addr:ServiceName>
     as:RegistrationService
    </addr:ServiceName>
  </addr:EndpointReference>
</sref:service-ref>
```

VARIABLES

Variable Declaration

<variables>

Declare variables within a process or a scope. Variables hold XML data. Variable can be one of the following types: WSDL message type, XML Schema type, or XML Schema element. Variable is visible in the scope in which it is defined and in all nested scopes.

ORACLE
FUSION MIDDLEWARE
BPEL PROCESS MANAGER

Oracle BPEL Process Manager offers you the greatest ease-of-use and the industry's best scalability. Plus, it's hot-pluggable!

Learn more at
www.oracle.com/appserver/bpel_home.html

Syntax:

```
<variables>
  <variable name="BPELVariableName"
     messageType="QName"?
     type="QName"?
     element="QName"?>+
     from-spec?
  </variable>
</variables>
```

Example:

```
<variables>
  <variable name="OrderForm"
     messageType="ord:OrderFormMsg" />
</variables>
```

Manipulating Variables

<assign>

Update and copy data between variables, expressions, and partner link endpoint references.

```
<assign validate="yes|no"? standard-attributes>
    standard-elements
    <copy keepSrcElementName="yes|no"?
            ignoreMissingFromData="yes|no"?>
    from-spec
    to-spec
  </copy>
</assign>
```

from-spec	`<from variable="BPELVariableName" part="NCName"?>`
	`   <query queryLanguage="anyURI"?>?`
	`      queryContent`
	`   </query>`
	`</from>`
	`<from partnerLink="NCName"`
	`      endpointReference="myRole|partnerRole" />`
	`<from variable="BPELVariableName"`
	`      property="QName" />`
	`<from expressionLanguage="anyURI"?>`
	`   expression`
	`</from>`
	`<from>`
	`   <literal>literal value</literal>`
	`</from>`
	`<from/>`

to-spec	`<to variable="BPELVariableName" part="NCName"?>`
	`   <query queryLanguage="anyURI"?>?`
	`      queryContent`
	`   </query>`
	`</to>`
	`<to partnerLink="NCName" />`
	`<to variable="BPELVariableName"`
	`    property="QName" />`
	`<to expressionLanguage="anyURI"?>`
	`   expression`
	`</to>`
	`<to/>`

messageType	$VariableName.part/ns:node1/ns:node2/…
Type	$VariableName/ns:node1/ns:node2/…
Element	$VariableName/ns:node1/ns:node2/…

Example:

```
<assign>
   <copy>
      <from>$Order/Item/Amount * $ExchangeRate</from>
      <to>$OrderFrgn/Item/Amount</to>
   </copy>
</assign>
```

<validate>

Validates the values of variable against the associated XML or WSDL data definition. For invalid validation, bpel:invalidVariables fault is thrown.

```
<validate variables="BPELVariableNames" standard-attributes>
   standard-elements
</validate>
```

Example:

```
<validate variables="Order OrderFrgn"/>
```

PROCESS FLOW ACTIVITIES

<sequence>

Defines a set of activities that will be executed in sequential order.

```
<sequence standard-attributes>
   standard-elements
   activity+
</sequence>
```

Example:

```
<sequence name="OrderProcessing">
  <receive .../>
  <assign>...</assign>
  <invoke .../>
  <reply .../>
</sequence>
```

<flow>

Specifies activities that should be performed concurrently.

```
<flow standard-attributes>
   standard-elements
   <links>?
      <link name="NCName" />+
   </links>
   activity+
</flow>
```

Example:

```
<flow name="CheckPrice">
  <invoke name="CheckSupl1" .../>
  <invoke name="CheckSupl2" .../>
  <invoke name="CheckSupl3" .../>
</flow>
```

To define synchronization dependencies between activities within <flow>, we use <links>. This way we define the order of execution. Links have to be declared within the <flow>.

"STANDARD-ELEMENTS"

Links: <targets> and <sources>

Define source and destination links for synchronization of flow activities. <source> is used to annotate an activity being a source of one or more links. <target> is used to annotate an activity being a target of one or more links. A link's target activity can be performed only after the source activity has been finished.

```
<targets>?
   <joinCondition expressionLanguage="anyURI"?>?
      bool-expr
   </joinCondition>
   <target linkName="NCName" />+
</targets>

<sources>?
   <source linkName="NCName">+
      <transitionCondition expressionLanguage="anyURI"?>?
         bool-expr
      </transitionCondition>
   </source>
</sources>
```

Example:

```
<flow>
  <links>
    <link name="TravelStatusToTicketRequest" />
    <link name="TicketRequestToTicketConfirmation" />
  </links>
  <assign>
    <sources>
      <source linkName="TravelStatusToTicketRequest" />
    </sources>
    <copy>…</copy>
  </assign>
  <invoke partnerLink="TicketConf" portType="tc:TicketPT"
          operation="ConfirmTicket" inputVariable="TicketRequestVar"
          outputVariable="TicektConfVar" >
    <targets>
      <target linkName="TravelStatusToTicketRequest" />
    </targets>
    <sources>
      <source linkName="TicketRequestToTicketConfirmation" />
    </sources>
  </invoke>
  ...
</flow>
```

<joinCondition>

Defines explicit join condition for incoming (<target>) links. Default is OR (at least one incoming link to be true). Positive join condition (true) is required for starting the activity. If join condition evaluates to false, bpel:joinFailure fault is thrown (except if suppressJoinFailure="yes"). Example:

```
<targets>
  <joinCondition>$TicketApproved and $DiscountGiven</joinCondition>
  <target linkName="OrderTicket" />
  <target linkName="IssuePayment" />
</targets>
```

<transitionCondition>

Defines the condition for outgoing (<source>) links to have positive status. Default is that they evaluate to true. Example:

```
<sources>
  <source linkName="IssuePaymentAutomatic">
    <transitionCondition>
      $Ticket.Amount &lt; 1000
    </transitionCondition>
  </source>
  <source linkName=" IssuePaymentWithApproval">
    <transitionCondition>
```

```
      $Ticket.Amount >= 1000
    </transitionCondition>
  </source>
</sources>
```

"STANDARD-ATTRIBUTES"

suppressJoinFailure

Indicates whether a bpel:joinFailure fault should be suppressed or not. When not specified, it inherits its value from its closest enclosing construct.

```
suppressJoinFailure="yes|no"?
```

name

To give a name to the BPEL activity (not related with links).

```
name="NCName"?
```

BPEL ACTIVITIES

<invoke>

Invokes an operation on a partner link web service.

```
<invoke partnerLink="NCName"
   portType="QName"?
   operation="NCName"
   inputVariable="BPELVariableName"?
   outputVariable="BPELVariableName"?
   standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="NCName" initiate="yes|join|no"?
         pattern="request|response|request-response"? />+
   </correlations>
   <catch faultName="QName"?
      faultVariable="BPELVariableName"?
      faultMessageType="QName"?
      faultElement="QName"?>*
      activity
   </catch>
   <catchAll>?
      activity
   </catchAll>
   <compensationHandler>?
      activity
   </compensationHandler>
   <toParts>?
      <toPart part="NCName" fromVariable="BPELVariableName" />+
   </toParts>
   <fromParts>?
      <fromPart part="NCName" toVariable="BPELVariableName" />+
   </fromParts>
</invoke>
```

Example:

```
<invoke partnerLink="Ordering"
        portType="ord:OrderPT"
        operation="ConfirmOrder"
        inputVariable="Order"
        outputVariable="OrderConf" />
```

<receive>

Waits for an incoming message (operation invocation). Typical uses: first process activity (createInstance="yes"); to wait for callbacks.

```
<receive partnerLink="NCName"
   portType="QName"?
   operation="NCName"
   variable="BPELVariableName"?
   createInstance="yes|no"?
   messageExchange="NCName"?
   standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="NCName" initiate="yes|join|no"? />+
   </correlations>
   <fromParts>?
      <fromPart part="NCName" toVariable="BPELVariableName" />+
   </fromParts>
</receive>
```

Example:

```
<receive partnerLink="Ordering"
         portType="ord:OrderCallbackPT"
         operation="confirmOrder"
         variable="OrderConfirmation"/>
```

<pick>

Waits for one of several possible messages (operation invocations) or for a time-out. <pick> can be the first process activity (createInstance="yes", no <onAlarm> allowed).

```
<pick createInstance="yes|no"? standard-attributes>
   standard-elements
   <onMessage partnerLink="NCName"
      portType="QName"?
      operation="NCName"
      variable="BPELVariableName"?
      messageExchange="NCName"?>+
      <correlations>?
         <correlation set="NCName" initiate="yes|join|no"? />+
      </correlations>
      <fromParts>?
         <fromPart part="NCName" toVariable="BPELVariableName" />+
      </fromParts>
      activity
   </onMessage>
   <onAlarm>*
      (
      <for expressionLanguage="anyURI"?>duration-expr</for>
      |
      <until expressionLanguage="anyURI"?>deadline-expr</until>
      )
      activity
   </onAlarm>
</pick>
```

Example:

```
<pick>
  <onMessage partnerLink="Ordering"
      portType="ord:OrderPT"
      operation="confirmOrder"
      variable="OrderConfirmation">
      ...
   </onMessage>
   <onMessage partnerLink="Ordering"
      portType="ord:OrderPT"
      operation="cancelOrder"
      variable="OrderCancelation">
      ...
   </onMessage>
   <onAlarm>
      <for>'PT12H'</for>
      <!-- Order completion timed out -->
   </onAlarm>
</pick>
```

<reply>

Sends a response for a request-response operation (synchronous). The request is received using either <receive>, <onMessage>, or <onEvent>.

```
<reply partnerLink="NCName"
   portType="QName"?
   operation="NCName"
   variable="BPELVariableName"?
   faultName="QName"?
   messageExchange="NCName"?
   standard-attributes>
   standard-elements
   <correlations>?
      <correlation set="NCName" initiate="yes|join|no"? />+
   </correlations>
   <toParts>?
      <toPart part="NCName" fromVariable="BPELVariableName" />+
   </toParts>
</reply>
```

Example:

```
<reply partnerLink="Ordering"
       portType="ord:OrderPT"
       operation="placeOrder"
       variable="Order"/>
```

<wait>

Waits for a specified time period or until a certain deadline.

```
<wait standard-attributes>
   standard-elements
   (
   <for expressionLanguage="anyURI"?>duration-expr</for>
   |
   <until expressionLanguage="anyURI"?>deadline-expr</until>
   )
</wait>
```

Example:

```
<wait>
  <until>'2010-03-18T20:00+01:00'</until>
</wait>
```

<exit>

Immediately ends the BPEL process instance.

```
<exit standard-attributes>
   standard-elements
</exit>
```

Example:

```
<exit/>
```

<empty>

This activity does not do anything. It is useful for synchronization of concurrent activities.

```
<empty standard-attributes>
   standard-elements
</empty>
```

Example:

```
<empty/>
```

deadline-expr

Used in until expression of <onAlarm> and <wait>. XML Schema date or dateTime types are used to express deadlines (following ISO 8601).

```
<until>’2010-01-01’</until>
<until>’2010-03-18T21:00:00+01:00’</until>
<until>’18:05:30Z’</until>
```

duration-expr

Used in for expression of <onAlarm> and <wait>, and <repeatEvery> expression of <onAlarm>. XML Schema duration type is used (following ISO 8601).

```
<for>’PT4H10M’</for>
<for>’P1M3DT4H10M’</for>
<for>’P1Y11M14DT4H10M30S’</for>
```

P	Time duration designator. Duration expressions always start with P.
Y	Follows the number of years.
M	Follows the number of months or minutes.
D	Follows the number of days.
H	Follows the number of hours.
S	Follows the number of seconds.
T	Date-Time separator

CONDITIONAL BEHAVIOR

<if>

To model decisions. <if> selects exactly one activity from the set of choices.

```
<if standard-attributes>
   standard-elements
   <condition expressionLanguage=”anyURI”?>bool-expr</condition>
   activity
   <elseif>*
      <condition expressionLanguage=”anyURI”?>
           bool-expr
      </condition>
      activity
   </elseif>
   <else>?
      activity
   </else>
</if>
```

Example:

```
<if>
  <condition>
    $Order.Amount > 1000
  </condition>
  ... <!-- Make approval -->
  <elseif>
    <condition>
      $Order.Amount >= 0
    </condition>
    ... <!-- Process automatically -->
  </elseif>
  <else>
    ... <!-- Throw fault -->
  </else>
</if>
```

LOOPS

<while>

Define a loop that repeats as long as the specified <condition> is true.

```
<while standard-attributes>
   standard-elements
   <condition expressionLanguage=”anyURI”?>bool-expr</condition>
   activity
</while>
```

Example:

```
<while>
  <condition>$Order.Amount &lt; 1000</condition>
  <sequence>...</sequence>
</while>
```

<repeatUntil>

Defines a loop that repeats until the specified <condition> becomes true. The <condition> is tested after the loop activities complete. Loop will execute at least once.

```
<repeatUntil standard-attributes>
   standard-elements
   activity
   <condition expressionLanguage=”anyURI”?>bool-expr</condition>
</repeatUntil>
```

Example:

```
<repeatUntil>
  <sequence>...</sequence>
  <condition>$Order.Amount >= 1000</condition>
</repeatUntil>
```

<forEach>

Iterates its child scope activities in parallel (parallel=”yes”) or sequential manner, exactly <finalCounterValue>-<startCounterValue>+1 times. An optional <completionCondition> allows the <forEach> activity to complete without executing or finishing all the branches specified.

```
<forEach counterName=”BPELVariableName” parallel=”yes|no”
   standard-attributes>
   standard-elements
   <startCounterValue expressionLanguage=”anyURI”?>
      unsigned-integer-expression
   </startCounterValue>
   <finalCounterValue expressionLanguage=”anyURI”?>
      unsigned-integer-expression
   </finalCounterValue>
   <completionCondition>?
      <branches expressionLanguage=”anyURI”?
         successfulBranchesOnly=”yes|no”?>
         unsigned-integer-expression
      </branches>
   </completionCondition>
   <scope ...>...</scope>
</forEach>
```

Example:

```
<forEach counterName=”NoOfSuppliers” parallel=”yes”>
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>3</finalCounterValue>
  <scope>
    <invoke name=”CheckPrice” .../>
  </scope>
</forEach>
```

SCOPES

<scope>

Defines a nested process scope with its own associated <partnerLinks>, <messageExchanges>, <variables>, <correlationSets>, <faultHandlers>, <compensationHandler>, <terminationHandler>, and <eventHandlers>.

```
<scope isolated=”yes|no”? exitOnStandardFault=”yes|no”?
   standard-attributes>
   standard-elements
   <partnerLinks/>?
   <messageExchanges/>?
   <variables/>?
   <correlationSets/>?
   <faultHandlers/>?
   <compensationHandler/>?
   <terminationHandler/>?
   <eventHandlers/>?
   activity
</scope>
```

Example:

```
<scope name=”CheckSupplier”>
  <partnerLinks>...</partnerLinks>
  <variables>...</variables>
  <faultHandlers>...</faultHandlers>
  <sequence>
    ...
  </sequence>
</scope>
```

FAULT HANDLING

<throw>

Generates a fault from inside the business process. Fault is identified by a qualified name.

```
<throw faultName=”QName”
   faultVariable=”BPELVariableName”?
   standard-attributes>
   standard-elements
</throw>
```

Example:

```
<throw faultName=”tns:InvalidOrder” />
```

<rethrow>

Rethrows the fault that was originally caught by the enclosing fault handler. <rethrow> can only be used within a fault handler (<catch> or <catchAll>).

```
<rethrow standard-attributes>
   standard-elements
</rethrow>
```

Example:

```
<rethrow />
```

<faultHandlers>

Define the activities that are performed in response to faults. Fault handler can be <catch> or <catchAll>. They can be defined at the <process> level, within <scope>s, or inline for <invoke>.

```
<faultHandlers>?
  <!-- There must be at least one faultHandler -->
  <catch faultName="QName"?
         faultVariable="BPELVariableName"?
         ( faultMessageType="QName" | faultElement="QName" )? >*
      activity
  </catch>
  <catchAll>?
      activity
  </catchAll>
</faultHandlers>
```

Default fault handler:

```
<faultHandlers>
  <catchAll>
    <sequence>
      <compensate />
      <rethrow />
    </sequence>
  </catchAll>
</faultHandlers>
```

EVENT HANDLERS

<eventHandlers>

Allow a process or scope to react on inbound messages (operation invocations) or on alarms. Event handler must contain at least one <onEvent> or <onAlarm> element. It can be defined at the <process> level or within <scope>s.

```
<eventHandlers>?
  <!-- There must be at least one onEvent or onAlarm. -->
  <onEvent partnerLink="NCName"
      portType="QName"?
      operation="NCName"
      ( messageType="QName" | element="QName" )?
      variable="BPELVariableName"?
      messageExchange="NCName"?>*
    <correlations>?
      <correlation set="NCName" initiate="yes|join|no"? />+
    </correlations>
    <fromParts>?
      <fromPart part="NCName" toVariable="BPELVariableName" />+
    </fromParts>
    <scope ...>...</scope>
  </onEvent>
  <onAlarm>*
    <!-- There must be at least one expression. -->
    (
    <for expressionLanguage="anyURI"?>duration-expr</for>
    |
    <until expressionLanguage="anyURI"?>deadline-expr</until>
    )?
    <repeatEvery expressionLanguage="anyURI"?>
      duration-expr
    </repeatEvery>?
    <scope ...>...</scope>
  </onAlarm>
</eventHandlers>
```

Example:

```
<eventHandlers>
  <onEvent partnerLink="Ordering"
         portType="ord:OrderPT"
         operation="CancelOrder"
         messageType="ord:CancelOrderMsg"
         variable="CancelationDetails">
    <scope>
      ...
    </scope>
  </onEvent>
  <onAlarm>
    <for>'PT12H'</for>
    <scope>
      ...
    </scope>
  </onAlarm>
</eventHandlers>
```

COMPENSATION

<compensationHandler>

Defines activities that are processed for compensation. Can be defined within <scope> or inline for <invoke>.

```
<compensationHandler>
   activity
</compensationHandler>
```

Default compensation handler:

```
<compensationHandler>
   <compensate />
</compensationHandler>
```

<compensateScope>

Starts compensation on a specified inner scope that has already completed successfully. <compensateScope> can only be used within a fault handler, another compensation handler, or a termination handler.

```
<compensateScope target="NCName" standard-attributes>
   standard-elements
</compensateScope>
```

Example:

```
<compensateScope target="PlaceOrder" />
```

<compensate>

Starts compensation on all inner scopes that have already completed successfully, in default order. <compensate> can only be used within a fault handler, another compensation handler, or a termination handler.

```
<compensate standard-attributes>
   standard-elements
</compensate>
```

Example:

```
<compensate />
```

TERMINATION HANDLER

<terminationHandler>

Defines the activities that are performed when a BPEL process is force terminated. Can be defined within <scope>.

```
<terminationHandler>
   activity
</terminationHandler>
```

Default compensation handler:

```
<terminationHandler>
   <compensate />
</terminationHandler>
```

MESSAGE EXCHANGES

<messageExchanges>

Used to name message exchanges.

```
<messageExchanges>?
  <messageExchange name="NCName" />+
</messageExchanges>
```

Default compensation handler:

```
<messageExchanges>
  <messageExchange name="OrderME" />
</messageExchanges>
```

messageExchange attribute

Used to associate inbound message activities with <reply> activities (for example <receive> with <reply>). Use only if the execution can result in multiple pairs of inbound message activities and <reply>s.

XSLT TRANSFORMATION

```
object bpel:doXslTransform(string1, node-set, (string2, object)*)
   string1 = style sheet name
   node-set = source document for the transformation
   string2 = XSLT parameter name
   object = XSLT parameter value-can be XPath expression
```

Function returns the result of transformation. Example:

```
<assign>
  <copy>
    <from>bpel:doXslTransform("OrderXSLT.xsl", $Order)</from>
    <to variable="OrderTransformed" />
  </copy>
</assign>
```

PROPERTIES

<vprop:property>

Creates a unique name definition and associates it with an XML Schema type. <vprop:property> is defined in the service WSDL document, not in the BPEL.

```
<vprop:property name="NCName" type="QName"? element="QName"? />
```

Example:

```
<vprop:property name="OrderIDNumber" type="ord:OrdIDType" />
```

<vprop:propertyAlias>

Maps a property to a field in a specific message part or variable value. <vprop:property> is defined in the service WSDL document, not in the BPEL.

```
<vprop:propertyAlias propertyName="QName"
  messageType="QName"? part="NCName"?
  type="QName"?
  element="QName"?>
  <vprop:query queryLanguage="anyURI"?>?
     queryContent
  </vprop:query>
</vprop:propertyAlias>
```

Example:

```
<vprop:propertyAlias propertyName="tns:OrderIDNumber"
     messageType="ord:OrderMsg" part="data">
  <vprop:query>ord:OrderID</vprop:query>
</vprop:propertyAlias>
```

bpel:getVariableProperty()

Extracts property values from variables.

```
object bpel:getVariableProperty(string1, string2)
   string1 – source variable name.
   string2 – property to select from variable
```

Example:

```
bpel:getVariableProperty('Order','ord:OrderIDNumber')
```

CORRELATION

<correlationSets>

Correlation set is a set of properties shared by all messages in the correlated group of operations within a process instance. Can be declared within a process or scope, or inline for <invoke>, <receive>, <reply>, <onMessage>, and <onEvent>.

```
<correlationSets>?
  <correlationSet name="NCName" properties="QName-list" />+
</correlationSets>
```

Example:

```
<correlationSets>
   <correlationSet name="Order"
      properties="ord:OrderIDNumber ord:CustomerID" />
   <correlationSet name="Invoice"
      properties="ord:InvoiceNumber" />
</correlationSets>
```

<correlation>

Correlation can be used on <invoke>, <receive>, <reply>, <onMessage>, and <onEvent>. <correlation> indicates which correlation sets occur in the messages being sent and received. The initiate attribute is used to indicate whether the correlation set is being initiated (yes-initiated, join-initiated if not yet initiated, no-not initiated).

```
<correlations>
  <correlation set="NCName"
      initiate="yes|join|no"?
      pattern="request|response|request-response"? />+
</correlations>
```

Example:

```
<receive partnerLink="Ordering" portType="ord:OrderPT"
   operation="PurchaseRequest" variable="Order">
   <correlations>
      <correlation set="Order" initiate="yes" />
   </correlations>
</receive>
...
<invoke partnerLink="Ordering" portType="ord:OrderPT"
   operation="PurchaseResponse" inputVariable="OrderResponse">
   <correlations>
      <correlation set="Order" initiate="no" />
      <correlation set="Invoice" initiate="yes" />
   </correlations>
</invoke>
```

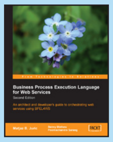
ABOUT THE AUTHORS

Matjaz B Juric, Ph. D. is professor at the University of Maribor and the head of the SOA Competency Centre. He has been consultant for several large companies on the BPM/SOA projects and has worked on projects, such as SOA Maturity Model, SOA in Telcos, performance analysis and optimization of RMI-IIOP, etc. Matjaz is author of courses for the BPEL and SOA consulting company BPELmentor.com. He is also a member of the BPEL Advisory Board.

Publications:

- Business Process Execution Language for Web Services (Packt Publishing, 2006)
- BPEL CookBook: Best Practices for SOA-based integration and composite applications development (Packt Publishing, 2007)
- Business Process Driven SOA using BPMN and BPEL (Packt Publishing, 2008)
- SOA Approach to Integration (Packt Publishing, 2007)

RECOMMENDED BOOK



This book provides detailed coverage of BPEL, its syntax, and where, and how, it is used. It begins with an overview of web services, showing both the foundation of, and need for, BPEL. The web services orchestration stack is explained, including standards such as WS-Security, WS-Coordination, WS-Transaction, WS-Addressing, and others.

BUY NOW
books.dzone.com/books/bpel

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-92-9
ISBN-10: 1-934238-92-9
50795
9 781934 238929
$7.95

Version 1.0