BPEL 1.1 Syntax Reference

This appendix provides a syntax reference for the BPEL (BPEL4WS) version 1.1 as defined in the specification dated May 5th, 2003, available at the following URL:

http://www.oasis-open.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf

In the following sections, you will see the syntax for the following:

- BPEL activities and elements
- BPEL functions
- Deadline and duration expressions
- Standard elements
- Standard attributes
- Default attribute values
- Standard faults
- Namespaces

Important BPEL activities and elements

The following section covers important activity and element tags associated with BPEL processing.

<assign>, <copy>, <from>, <to>

The <assign> activity is used to do the following:

- Copy data from one variable to another
- Construct and insert new data using expressions and literal values
- Copy partner link endpoint references
Syntax

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

The from-spec section can have the following forms:

<from variable="ncname" part="ncname"/>
<from variable="ncname" part="ncname"? query="queryString"/>
<from partnerLink="ncname" endpointReference="myRole|partnerRole"/>
<from variable="ncname" property="qname"/>
<from expression="general-expr"/>
<from> ... literal value ... </from>
<from opaque="yes"> <!-- Abstract processes only -->

The to-spec section can have the following forms:

<to variable="ncname" part="ncname"/>
<to variable="ncname" part="ncname"? query="queryString"/>
<to partnerLink="ncname"/>
<to variable="ncname" property="qname"/>

Example

<assign>
  <copy>
    <from variable="TravelRequest" part="flightData"/>
    <to variable="FlightDetails" part="flightData"/>
  </copy>
  <copy>
    <from variable="EmployeeTravelStatusResponse"
         part="travelClass"/>
    <to variable="FlightDetails" part="travelClass"/>
  </copy>
</assign>
<catch>, <catchAll>

The <catch> activities are specified within fault handlers to specify faults that are to be caught and handled. The <catchAll> activity is used to catch all faults. Within a fault handler, at least one <catch> activity needs to be specified. The optional <catchAll> activity can also be specified.

The <catch> activity has two attributes that can be used to specify which fault to handle. At least one of them needs to be specified:

- faultName: Specifies the name of the fault to handle
- faultVariable: Specifies the variable type used for fault data

Syntax

```
<faultHandlers>
  <!-- there must be at least one fault handler or default -->
  <catch faultName="qname"? faultVariable="ncname"?>*
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
</faultHandlers>
```

When used to define an inline fault handler for the <invoke> activity, the syntax is as follows:

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
  inputVariable="ncname"? outputVariable="ncname"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="ncname" initiate="yes|no"?
      pattern="in|out|out-in"/>+
  </correlations>
  <catch faultName="qname" faultVariable="ncname"?>*
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
  <compensationHandler>?
    activity
  </compensationHandler>
</invoke>
```
Example

```xml
<faultHandlers>
  <catch faultName="trv:TicketNotApproved" >
    <!-- Perform an activity -->
  </catch>
  <catch faultName="trv:TicketNotApproved"
      faultVariable="TravelFault" >
    <!-- Perform an activity -->
  </catch>
  <catch faultVariable="TravelFault" >
    <!-- Perform an activity -->
  </catch>
  <catchAll>
    <!-- Perform an activity -->
  </catchAll>
</faultHandlers>
```

**<compensate>**

To invoke a compensation handler, you use the `<compensate>` activity. The `<compensate>` activity has an optional `scope` attribute that can be used to specify the compensation handler to be invoked. The scope name or activity name (for inline compensation handlers) has to be specified.

**Syntax**

```xml
<compensate scope="ncname"? standard-attributes>
    standard-elements
</compensate>
```

**Example**

```xml
<compensate scope="TicketConfirmationPayment" />
```

or

```xml
<compensate scope="TicketConfirmation" />
```

**<compensationHandler>**

Compensation handlers are used to define compensation activities. Compensation handlers gather all activities that have to be carried out to compensate another activity and can be defined for the whole process or scope, or can be inlined for the `<invoke>` activity.
Syntax

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

Example

Compensation handler for a scope:

```xml
<scoop name="TicketConfirmationPayment">
  <compensationHandler>
    <invoke partnerLink="AmericanAirlines"
      portType="aln:TicketConfirmationPT"
      operation="CancelTicket"
      inputVariable="FlightDetails"
      outputVariable="Cancellation" />
    <invoke partnerLink="AmericanAirlines"
      portType="aln:TicketPaymentPT"
      operation="CancelPayment"
      inputVariable="PaymentDetails"
      outputVariable="PaymentCancellation" />
  </compensationHandler>
  <invoke partnerLink="AmericanAirlines"
    portType="aln:TicketConfirmationPT"
    operation="ConfirmTicket"
    inputVariable="FlightDetails"
    outputVariable="Confirmation" />
  <invoke partnerLink="AmericanAirlines"
    portType="aln:TicketPaymentPT"
    operation="PayTicket"
    inputVariable="PaymentDetails"
    outputVariable="PaymentConfirmation" />
</scope>
```

Inline compensation handler for the `<invoke>` activity:

```xml
<invoke name="TicketConfirmation"
  partnerLink="AmericanAirlines"
  portType="aln:TicketConfirmationPT"
  operation="ConfirmTicket"
  inputVariables="FlightDetails"
  outputVariable="Confirmation" />
```
<compensationHandler>
  <invoke partnerLink="AmericanAirlines"
    portType="aln:TicketConfirmationPT"
    operation="CancelTicket"
    inputVariable="FlightDetails"
    outputVariable="Cancellation" />
</compensationHandler>
</invoke>

<correlations>, <correlation>
The <correlation> element is used to associate a correlation set (a collection of key data fields) with an activity. Correlation can be used within the <receive>, <reply>, <invoke>, and <onMessage> activities.

Syntax
<correlations>
  <correlation set="ncname"
    initiate="yes|no"?
    pattern="in|out|out-in"/>+
</correlations>

Example
<correlations>
  <correlation set="TicketOrder" initiate="yes" pattern="in" />
</correlations>

<correlationSets>, <correlationSet>
A correlation set is a set of properties shared by messages and used for correlation. It is used to associate a message with a business process instance. Each correlation set has a name attribute.

Syntax
<correlationSets>
  <correlationSet name="ncname" properties="qname-list"/>+
</correlationSets>

Example
<correlationSets>
  <correlationSet name="VehicleOrder"
    properties="tns:chassisNo tns:engineNo"/>

<correlationSet name="TickerOrder"
   properties="aln:FlightNo"/>
</correlationSets>

<empty>
An activity that does nothing is defined by the <empty> tag.

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

Example
<empty/>

<eventHandlers>
Event handlers react to events that occur while the business process is executing. When these events occur, the corresponding event handlers are invoked. Event handlers can be specified for the whole process as well as for each scope.

Syntax
<eventHandlers>
   <!-- there must be at least one onMessage or onAlarm handler -->
   <onMessage partnerLink="ncname" portType="qname"
      operation="ncname"
      variable="ncname"?>*
      <correlations>?
         <correlation set="ncname" initiate="yes|no">+
      </correlations>
      activity
   </onMessage>
   <onAlarm for="duration-expr"? until="deadline-expr"?>*
      activity
   </onAlarm>
</eventHandlers>
Example

```xml
<!--process name="BusinessTravelProcess" enableInstanceCompensation="yes" ... -->

... ...
<eventHandlers>
    <onMessage partnerLink="client"
        portType="trv:TravelApprovalPT"
        operation="CancelTravelApproval"
        variable="TravelRequest" >
        <terminate/>
    </onMessage>
    <onAlarm for="'PT12H'">
        <terminate/>
    </onAlarm>
</eventHandlers>
... ...
</process>
```

**<faultHandlers>, <faultHandler>**

Fault handlers are used to react to faults that occur while the business process activities are executing. They can be specified for the global process or each scope, or inline for `<invoke>` activities. Multiple `<catch>` activities can be specified within the fault handler for specific faults. You need to specify at least one `<catch>` activity. You can optionally specify the `<catchAll>` activity. The syntax and functionality of the `<catch>` activity is the same as described in the earlier section on `<catch>` and `<catchAll>`.

**Syntax**

```xml
<faultHandlers>
    <!-- There must be at least one fault handler or default -->
    <catch faultName="qname"? faultVariable="ncname"?>*
        activity
    </catch>
    <catchAll>?
        activity
    </catchAll>
</faultHandlers>
```

An inline fault handler for the `<invoke>` activity is specified as shown next:
Example

```xml
<faultHandlers>
  <catch faultName="trv:TicketNotApproved">
    <!-- Perform an activity -->
  </catch>
  <catch faultName="trv:TicketNotApproved" faultVariable="TravelFault">
    <!-- Perform an activity -->
  </catch>
  <catch faultVariable="TravelFault">
    <!-- Perform an activity -->
  </catch>
  <catchAll>
    <!-- Perform an activity -->
  </catchAll>
</faultHandlers>
```

<flow>

The `<flow>` activity provides concurrent execution of enclosed activities and their synchronization.
Syntax

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

Example

```xml
<flow>
  <!-- Synchronously invoke the Employee Travel Status Web Service -->
  <invoke name="EmployeeTravelStatusSyncInv"
    partnerLink="employeeTravelStatus"
    portType="emp:EmployeeTravelStatusPT"
    operation="EmployeeTravelStatus"
    inputVariable="EmployeeTravelStatusRequest"
    outputVariable="EmployeeTravelStatusResponse" />

  <!-- Prepare the input for AA and DA -->
  ...

  <!-- Async invoke of the AA web service -->
  <invoke name="AmericanAirlinesAsyncInv"
    partnerLink="AmericanAirlines"
    portType="aln:FlightAvailabilityPT"
    operation="FlightAvailability"
    inputVariable="FlightDetails" />

  <!-- Receive the callback -->
  <receive name="AmericanAirlinesCallback"
    partnerLink="AmericanAirlines"
    portType="aln:FlightCallbackPT"
    operation="FlightTicketCallback"
    variable="FlightResponseAA" />

  <!-- Async invoke of the DA web service -->
  <invoke name="DeltaAirlinesAsyncInv"
    partnerLink="DeltaAirlines"
    portType="aln:FlightAvailabilityPT"
    operation="FlightAvailability"
    inputVariable="FlightDetails" />

  <!-- Receive the callback -->
  <receive name="DeltaAirlinesCallback"
  ```
<invoke>

The `<invoke>` activity is used to invoke the web service operations provided by partners.

**Syntax**

```
<invoke partnerLink="ncname"
   portType="qname"
   operation="ncname"
   inputVariable="ncname"?
   outputVariable="ncname"?
   standard-attributes>
   standard-elements
   <correlations>?
     <correlation set="ncname" initiate="yes|no"?
       pattern="in|out|out-in"/>+ 
   </correlations>
   <catch faultName="qname" faultVariable="ncname">*
     activity
   </catch>
   <catchAll>?
     activity
   </catchAll>
   <compensationHandler>?
     activity
   </compensationHandler>
</invoke>
```
Example

<!-- Synchronously invoke the Employee Travel Status Web Service -->
<invoke name="EmployeeTravelStatusSyncInv"
   partnerLink="employeeTravelStatus"
   portType="emp:EmployeeTravelStatusPT"
   operation="EmployeeTravelStatus"
   inputVariable="EmployeeTravelStatusRequest"
   outputVariable="EmployeeTravelStatusResponse" />

<!-- Async invoke of the AA web service -->
<invoke name="AmericanAirlinesAsyncInv"
   partnerLink="AmericanAirlines"
   portType="aln:FlightAvailabilityPT"
   operation="FlightAvailability"
   inputVariable="FlightDetails" />

<links>, <link>
Synchronization dependencies in concurrent flows are specified using links.

Syntax

<links>
   <link name="ncname">+</link>
</links>

Example

<links>
   <link name="TravelStatusToTicketRequest" />
   <link name="TicketRequestToTicketConfirmation" />
</links>

<onAlarm>
This activity is used in the <pick> and <eventHandlers> activities to specify the occurrence of alarm events.

Syntax

<onAlarm (for="duration-expr" | until="deadline-expr")>
   activity
</onAlarm>
Example

```xml
<onAlarm for="'PT30M'">
  <throw faultName="trv:CallbackTimeout"/>
</onAlarm>
```

**<onMessage>**

This is used in `<pick>` and `<eventHandlers>` activities to specify the occurrence of message events.

**Syntax**

```xml
<onMessage partnerLink="ncname" portType="qname"
  operation="ncname" variable="ncname"/>
<correlations>?
  <correlation set="ncname" initiate="yes|no"/>+</correlations>
activity
</onMessage>
```

Example

```xml
<onMessage partnerLink="AmericanAirlines"
  portType="aln:FlightCallbackPT"
  operation="FlightTicketCallback"
  variable="FlightResponseAA">
  <!-- Perform an activity -->
</onMessage>
```

**<partnerLinks>, <partnerLink>**

A business process interacts with services that are modeled as partner links. Each partner link is characterized by a `<partnerLinkType>`. More than one partner link can be characterized by the same `<partnerLinkType>`. See the next section for more on `<partnerLinkType>`.

**Syntax**

```xml
<partnerLinks>
  <partnerLink name="ncname" partnerLinkType="qname"
    myRole="ncname"? partnerRole="ncname"/>+
</partnerLink>
</partnerLinks>
```
Example

```xml
<partnerLinks>
  <partnerLink name="insurance"
    partnerLinkType="tns:insuranceLT"
    myRole="insuranceRequester"
    partnerRole="insuranceService"/>
</partnerLinks>
```

**<partnerLinkType>, <role>**

A partner link type characterizes the relationship between two services. It defines roles for each of the services in the conversation between them and specifies the port type provided by each service to receive messages. Partner link types and roles are specified in the WSDL.

**Syntax**

```xml
<definitions name="ncname" targetNamespace="uri"
  xmlns="http://schemas.xmlsoap.org/wsdl/
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  ...
  <plnk:partnerLinkType name="ncname">
    <plnk:role name="ncname">
      <plnk:portType name="qname"/>
    </plnk:role>
    <plnk:role name="ncname">
      <plnk:portType name="qname"/>
    </plnk:role>
  </plnk:partnerLinkType>
  ...
</definitions>
```

Example

```xml
<plnk:partnerLinkType name="flightLT">
  <plnk:role name="airlineService">
    <plnk:portType name="tns:FlightAvailabilityPT" />
  </plnk:role>
  <plnk:role name="airlineCustomer">
    <plnk:portType name="tns:FlightCallbackPT" />
  </plnk:role>
</plnk:partnerLinkType>
```
<partners>
The partner element is used to represent the capabilities required from a business partner. A partner is defined as a set of partner links.

Syntax

```xml
<partners>
  <partner name="ncname">+</n
c  <partnerLink name="ncname"/>+n
c  </partner>
</partners>
```

Example

```xml
<partner name="ServiceProvider">
  <partnerLink name="ReantacarProvider"/>
  <partnerLink name="LodgingProvider"/>
</partner>
```

<pick>
The <pick> activity is used to wait for the occurrence of one of a set of events and then perform an activity associated with the event.

Syntax

```xml
<pick createInstance="yes|no"? standard-attributes>
  standard-elements
  <onMessage partnerLink="ncname" portType="qname"
    operation="ncname" variable="ncname"?>+n
    <correlations>?
      <correlation set="ncname" initiate="yes|no"?>+n
      </correlations>
    activity
  </onMessage>
  <onAlarm (for="duration-expr" | until="deadline-expr")>*n
    activity
  </onAlarm>
</pick>
```

Example

```xml
<pick>
  <onMessage partnerLink="AmericanAirlines"
    portType="aln:FlightCallbackPT"
    operation="FlightTicketCallback"
```
variable="FlightResponseAA">

<!-- Perform an activity -->
</onMessage>

<onMessage partnerLink="AmericanAirlines"
portType="aln:FlightCallbackPT"
operation="FlightNotAvailable"
variable="FlightNAResponseAA">

<!-- Perform an activity -->
</onMessage>

<onMessage partnerLink="AmericanAirlines"
portType="aln:FlightCallbackPT"
operation="TicketNotAvailable"
variable="FlightTNAResponseAA">

<!-- Perform an activity -->
</onMessage>

<onAlarm for="PT30M">
  <throw faultName="trv:CallbackTimeout" />
</onAlarm>
</pick>

<process>

This is the root element of each BPEL process definition.

Syntax

<process name="ncname" targetNamespace="uri"
  queryLanguage="anyURI"?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  enableInstanceCompensation="yes|no"?
  abstractProcess="yes|no"?
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

<partnerLinks>?

<!-- Note: At least one role must be specified. -->
<partnerLink name="ncname" partnerLinkType="qname"
  myRole="ncname"? partnerRole="ncname"?/>

</partnerLink>
</partnerLinks>

<partners>?

<partner name="ncname">+
  <partnerLink name="ncname"/>
</partner>

</partners>?
<partner/>
</partners>

<variables>
  <variable name="ncname" messageType="qname">
    type="qname" element="qname"/>
  </variable>
</variables>

<correlationSets>
  <correlationSet name="ncname" properties="qname-list"/>
</correlationSets>

<faultHandlers>
  <!-- Note: There must be at least one fault handler or default. -->
  <catch faultName="qname" faultVariable="ncname">*
    activity
  </catch>
  <catchAll>
    activity
  </catchAll>
</faultHandlers>

<compensationHandler>
  activity
</compensationHandler>

<eventHandlers>
  <!-- Note: There must be at least one onMessage or onAlarm handler. -->
  <onMessage partnerLink="ncname" portType="qname">
    operation="ncname" variable="ncname"? >
    <correlations>
      <correlation set="ncname" initiate="yes|no">*
        <correlations>
          activity
        </correlations>
    </correlations>
  </onMessage>
  <onAlarm for="duration-exp"? until="deadline-exp">*
    activity
  </onAlarm>
</eventHandlers>

activity
</process>
Example
Examples are shown in Chapters 3 and 4.

<property>
Properties are used to create globally unique names and associate them with data types (XML Schema types). Properties have greater significance than the types themselves. Properties are defined in WSDL.

Syntax
<wsdl:definitions
    ...
    <bpws:property name="ncname" type="qname"/>
    ...
</wsdl:definitions>

Example
<bpws:property name="FlightNo" type="xs:string"/>

<propertyAlias>
Property aliases are used to map global properties to fields in specific message parts. Property aliases are defined in the WSDL.

Syntax
<wsdl:definitions
    ...
    <bpws:propertyAlias propertyName="qname"
        messageType="qname"
        part="ncname"
        query="queryString"/>
    ...
</wsdl:definitions>

Example
<bpws:propertyAlias propertyName="tns:FlightNo"
    messageType="tns:TravelResponseMessage"
    part="confirmationData"
    query="/confirmationData/FlightNo"/>
<receive>
A <receive> activity is used to receive requests in a BPEL business process to provide services to its partners.

Syntax
<receive partnerLink="ncname" portType="qname" operation="ncname"
variable="ncname"? createInstance="yes|no"?
standard-attributes>
standard-elements
<correlations>?
    <correlation set="ncname" initiate="yes|no"?>+</correlations>
</receive>

Example
<!-- Receive the initial request for business travel from client -->
<receive partnerLink="client"
    portType="trv:TravelApprovalPT"
    operation="TravelApproval"
    variable="TravelRequest" />

<reply>
A <reply> activity is used to send a response to a request previously accepted through a <receive> activity. Responses are used for synchronous request/reply interactions. An asynchronous response is always sent by invoking the corresponding one-way operation on the partner link.

Syntax
<reply partnerLink="ncname" portType="qname" operation="ncname"
variable="ncname"? faultName="qname"?
standard-attributes>
standard-elements
<correlations>?
    <correlation set="ncname" initiate="yes|no"?>+</correlations>
</reply>
Example

<!-- Send a response to the client -->
<reply partnerLink="client"
   portType="trv:TravelApprovalPT"
   operation="TravelApproval"
   variable="TravelResponse"/>

<scope>
Scopes define behavior contexts for activities. They provide fault handlers, event handlers, compensation handlers, data variables, and correlation sets for activities.

Syntax

<scope variableAccessSerializable="yes|no" standard-attributes>
standard-elements
<variables>?
...
</variables>
<correlationSets>?
...
</correlationSets>
<faultHandlers>?
...
</faultHandlers>
<compensationHandler>?
...
</compensationHandler>
<eventHandlers>?
...
</eventHandlers>
activity
</scope>

Example

<scope>

<faultHandlers>

<catch faultName="emp:WrongEmployeeName">
   <!-- Perform an activity -->
</catch>

<catch faultName="emp:TravelNotAllowed"
   faultVariable="Description">
   <!-- Perform an activity -->
</catch>

</scope>
<sequence>
A <sequence> activity is used to define activities that need to be performed in a sequential order.

Syntax

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

Example

Examples are covered in Chapters 3 and 4.

<source>
<source> elements are used to declare that an activity is the source of one or more links. This is a standard element.

Syntax

<source linkName="ncname" transitionCondition="bool-expr"/>*
Example

```xml
<invoke name="EmployeeTravelStatusSyncInv"
  partnerLink="employeeTravelStatus"
  portType="emp:EmployeeTravelStatusPT"
  operation="EmployeeTravelStatus"
  inputVariable="EmployeeTravelStatusRequest"
  outputVariable="EmployeeTravelStatusResponse" >

  <target linkName="EmployeeInputToEmployeeTravelStatusSyncInv" />
  <source linkName="EmployeeTravelStatusSyncInvToAirlinesInput" />
</invoke>
```

**<switch>, <case>**

To express conditional behavior, the `<switch>` activity is used. It consists of one or more conditional branches defined by `<case>` elements, followed by an optional `<otherwise>` element. The case branches of the switch are considered in alphabetical order.

**Syntax**

```xml
<switch standard-attributes>
  standard-elements
  <case condition="bool-expr">+
    activity
  </case>
  <otherwise>?
    activity
  </otherwise>
</switch>
```

**Example**

```xml
<!-- Select the best offer and construct the TravelResponse -->
<switch>

  <case condition="bpws:getVariableData('FlightResponseAA',
    'confirmationData','/confirmationData/aln:Price')
  &lt;= bpws:getVariableData('FlightResponseDA',
    'confirmationData','/confirmationData/aln:Price')">)

    <!-- Select American Airlines -->
    <assign>
```
<target>
The <target> element is used to declare that an activity is the target of one or more links. This is a standard element.

Syntax
<target linkName="ncname"/>*

Example
<!-- Select the best offer and construct the TravelResponse -->
<switch name="BestOfferSelect">

  <target linkName="AmericanAirlinesCallbackToBestOfferSelect" />
  <target linkName="DeltaAirlinesCallbackToBestOfferSelect" />

  <case condition="bpws:getVariableData('FlightResponseAA',
      'confirmationData', '/confirmationData/aln:Price')
    &lt;= bpws:getVariableData('FlightResponseDA',
      'confirmationData', '/confirmationData/aln:Price')">
      <!-- Select American Airlines -->
      <assign>
        <copy>
          <!-- Copy from FlightResponseAA to TravelResponse -->
          <from variable="FlightResponseAA" />
          <to variable="TravelResponse" />
        </copy>
      </assign>
  </case>

  <otherwise>
    <!-- Select Delta Airlines -->
    <assign>
      <copy>
        <!-- Copy from FlightResponseDA to TravelResponse -->
        <from variable="FlightResponseDA" />
        <to variable="TravelResponse" />
      </copy>
    </assign>
  </otherwise>
</switch>
The `<terminate>` activity is used to immediately terminate a business process instance.

**Syntax**

```xml
<terminate standard-attributes>
  standard-elements
</terminate>
```

**Example**

```xml
<terminate />
```

The `<throw>` activity is used to explicitly signal internal faults.

**Syntax**

```xml
<throw faultName="qname" faultVariable="ncname"? standard-attributes>
  standard-elements
</throw>
```

**Example**

```xml
<throw faultName="WrongEmployeeName" />
<throw faultName="trv:TicketNotApproved" faultVariable="TravelFault" />
```
<variables>, <variable>

Variables are used to hold messages that constitute the state of a business process. The variable may be of the WSDL message type, an XML Schema simple type, or an XML Schema element.

Syntax

```xml
<variables>
  <variable name="ncname" messageType="qname"?
    type="qname"? element="qname"?>/+
</variables>
```

Example

```xml
<variables>
  <variable name="InsuranceRequest" messageType="ins:InsuranceRequestMessage"/>
  <variable name="PartialInsuranceDescription" element="ins:InsuranceDescription"/>
  <variable name="lastName" type="xs:string"/>
</variables>
```

<wait>

A <wait> activity is used to specify a delay for a certain period of time or until a certain deadline is reached.

Syntax

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

Examples

```xml
<wait until="'2004-03-18T21:00:00+01:00'"/>
<wait until="'18:05:30Z'"/>
<wait for="'PT4H10M'"/>
<wait for="'P1M3DT4H10M'"/>
<wait for="'P1Y11M14DT4H10M30S'"/>
```
A `<while>` activity is used to define an iterative activity. The iterative activity is performed until the specified Boolean condition no longer holds true.

**Syntax**

```xml
<while condition="bool-expr" standard-attributes>
  standard-elements
  activity
</while>
```

**Example**

```xml
<while condition="bpws:getVariableData('Counter') &lt;
  bpws:getVariableData('NoOfPassengers')">
  <sequence>
    <!--Construct the FlightDetails variable with passenger data-->
    ... 
    <!-- Invoke the web service -->
    <invoke partnerLink="AmericanAirlines" 
      portType="aln:FlightAvailabilityPT"
      operation="FlightAvailability"
      inputVariable="FlightDetails" />
    <receive partnerLink="AmericanAirlines"
      portType="trv:FlightCallbackPT"
      operation="FlightTicketCallback"
      variable="FlightResponseAA" />
    ... 
    <!-- Process the results … -->
    ... 
    <!-- Increment the counter -->
    <assign>
      <copy>
        <from expression="bpws:getVariableData('Counter') + 1"/>
        <to variable="Counter"/>
      </copy>
    </assign>
  </sequence>
</while>
```
**BPEL functions**

This section talks about the important and commonly used functions related to BPEL. These are as follows:

- `getLinkStatus()`
- `getVariableData()`
- `getVariableProperty()`

### getLinkStatus()

This function is used to return a Boolean value to indicate the status of a link. Links with positive status are evaluated to `true`, links with negative status to `false`. This function can only be used in join conditions. The `linkName` argument must refer to the name of an incoming link for the activity associated with the join condition.

**Syntax**

```
bpws:getLinkStatus ('linkName')
```

**Example**

```xml
<switch name="BestOfferSelect"
   joinCondition="bpws:getLinkStatus(
      'AmericanAirlinesCallbackToBestOfferSelect')
   and
   bpws:getLinkStatus(
      'DeltaAirlinesCallbackToBestOfferSelect')">
   ...
</switch>
```

### getVariableData()

This function is used to extract arbitrary values from variables.

**Syntax**

```
bpws:getVariableData ('variableName', 'partName'? , 'locationPath'? )
```

**Example**

```xml
bpws:getVariableData ('InsuranceRequest',
   'insuredPersonData')
bpws:getVariableData ('InsuranceRequest',
   'insuredPersonData',
   '/insuredPersonData/ins:Age')
```
**getVariableProperty()**
This function is used to extract global property values from variables.

**Syntax**
```
bpws:getVariableProperty ('variableName', 'propertyName')
```

**Example**
```
bpws:getVariableProperty('TicketApproval', 'Class')
```

**Deadline and duration expressions**
To specify deadlines and durations for activities, BPEL uses lexical representations of corresponding XML Schema data types. For setting deadlines, the data types are either `dateTime` or `date`. For setting the duration (a timeout, for instance), you can use the `duration` data type. The lexical representation of expressions should conform to the XPath 1.0 (or the selected query language) specifications. Such expressions should evaluate to values of corresponding XML Schema types: `dateTime` and `date` for deadline and `duration` for duration expressions.

All three data types use lexical representations that conform to the ISO 8601 standard. For more information on this standard, see the ISO web page at [http://www.iso.ch](http://www.iso.ch). The ISO 8601 lexical format uses characters within date and time information. Characters are appended to the numbers and have the following meaning:

- `C` represents centuries.
- `Y` represents years.
- `M` represents months.
- `D` represents days.
- `h` represents hours.
- `m` represents minutes.
- `s` represents seconds. Seconds can be represented in the format `ss.sss` to increase precision.
- `Z` is used to designate Coordinated Universal Time (UTC). It should immediately follow the time-of-day element.

For the `dateTime` expressions, there is an additional designator:

- `T` is used as a time designator that indicates the start of the representation of the time.
For duration expressions, the following characters can also be used:

- \( P \) is used as the time duration designator. Duration expressions always start with \( P \).
- \( Y \) represents the number of years.
- \( M \) represents the number of months or minutes.
- \( D \) represents the number of days.
- \( H \) represents the number of hours.
- \( S \) represents the number of seconds.

**Standard elements**

Each activity has standard optional nested elements `<source>` and `<target>`.

**Syntax**

\[
\begin{align*}
\langle source \ linkName="ncname" \ transitionCondition="bool-expr"?/\rangle\ast \\
\langle target \ linkName="ncname"/\rangle\ast
\end{align*}
\]

**Standard attributes**

Each activity has some standard optional attributes:

- Name
- A join condition
- An indicator to state whether a join fault should be suppressed if it occurs

**Syntax**

The syntax associated with these attributes within the properties is as follows:

\[
\begin{align*}
\text{name="ncname"}?
\text{joinCondition="bool-expr"}?
\text{suppressJoinFailure="yes|no"}?
\end{align*}
\]
Default values of attributes
In the following table, we cover the default values of some BPEL attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryLanguage</td>
<td><a href="http://www.w3.org/TR/1999/REC-xpath-19991116">http://www.w3.org/TR/1999/REC-xpath-19991116</a></td>
</tr>
<tr>
<td>expressionLanguage</td>
<td><a href="http://www.w3.org/TR/1999/REC-xpath-19991116">http://www.w3.org/TR/1999/REC-xpath-19991116</a></td>
</tr>
<tr>
<td>suppressJoinFailure</td>
<td>No</td>
</tr>
<tr>
<td>variableAccessSerializable</td>
<td>No</td>
</tr>
<tr>
<td>abstractProcess</td>
<td>No</td>
</tr>
<tr>
<td>initiate</td>
<td>No</td>
</tr>
<tr>
<td>createInstance</td>
<td>No</td>
</tr>
<tr>
<td>enableInstanceCompensation</td>
<td>No</td>
</tr>
<tr>
<td>joinCondition</td>
<td>Disjunction of the status of the incoming links</td>
</tr>
<tr>
<td>transitionCondition</td>
<td>True</td>
</tr>
</tbody>
</table>

Standard faults
The following table is a list of standard faults that commonly arise in BPEL execution.

<table>
<thead>
<tr>
<th>Fault name</th>
<th>Fault is thrown when:</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectionFailure</td>
<td>A selection operation performed in either a function, such as bpws:getVariableData, or an assignment encounters an error.</td>
</tr>
<tr>
<td>conflictingReceive</td>
<td>More than one &lt;receive&gt; activity or equivalent (currently &lt;onMessage&gt; branch in a &lt;pick&gt; activity) are enabled simultaneously for the same partner link, port type, operation, and correlation set(s).</td>
</tr>
<tr>
<td>conflictingRequest</td>
<td>More than one synchronous inbound request is active on the same partner link for a particular port type, operation, and correlation set(s).</td>
</tr>
<tr>
<td>mismatchedAssignmentFailure</td>
<td>Incompatible types are encountered in an assign activity.</td>
</tr>
<tr>
<td>joinFailure</td>
<td>Join condition of an activity evaluates to false.</td>
</tr>
<tr>
<td>forcedTermination</td>
<td>A fault occurs in an enclosing scope.</td>
</tr>
</tbody>
</table>
Fault name | Fault is thrown when:  
--- | ---  
correlationViolation | The contents of messages processed in an `<invoke>`, `<receive>`, or `<reply>` activity do not match specified correlation information.  
uninitializedVariable | There is an attempt to access the value of an uninitialized part in a message variable.  
repeatedCompensation | An installed compensation handler is invoked more than once.  
invalidReply | A reply is sent on a partner link, port type, and operation for which the corresponding `<receive>` with the same correlation has not been carried out.  

### Namespaces

In the following table, the BPEL and partner link namespaces are listed.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>bpws</td>
<td><a href="http://schemas.xmlsoap.org/ws/2003/03/business-process/">http://schemas.xmlsoap.org/ws/2003/03/business-process/</a></td>
</tr>
<tr>
<td>plnk</td>
<td><a href="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">http://schemas.xmlsoap.org/ws/2003/05/partner-link/</a></td>
</tr>
</tbody>
</table>