WS-BPEL

A WS-BPEL activity can be any of the following:

* <receive>
* <reply>
* <invoke>
* <assign>
* <throw>
* <exit>
* <wait>
* <empty>
* <sequence>
* <if>
* <while>
* <repeatUntil>
* <forEach>
* <pick>
* <flow>
* <scope>
* <compensate>
* <compensateScope>
* <rethrow>
* <validate>
* <extensionActivity>

The syntax of each of these elements is described in the following paragraphs.

The <receive> activity allows the business process to wait for a matching message to arrive.

The <receive> activity completes when the message arrives. The portType attribute on the

<receive> activity is optional. [SA00005] If the portType attribute is included for readability,

the value of the portType attribute MUST match the portType value implied by the

combination of the specified partnerLink and the role implicitly specified by the activity (see

also partnerLink description in the next section). The optional messageExchange attribute is

used to associate a <reply> activity with a <receive> activity.

<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>

<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>

<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>

<assign validate="yes|no"? standard-attributes>

standard-elements

(

<copy keepSrcElementName="yes|no"? ignoreMissingFromData="yes|no"?>

from-spec

to-spec

</copy>

|

<extensionAssignOperation>

assign-element-of-other-namespace

</extensionAssignOperation>

)+

</assign>

<throw faultName="QName"

faultVariable="BPELVariableName"?

standard-attributes>

standard-elements

</throw>

<wait standard-attributes>

standard-elements

(

<for expressionLanguage="anyURI"?>duration-expr</for>

|

<until expressionLanguage="anyURI"?>deadline-expr</until>

)

</wait>

<empty standard-attributes>

standard-elements

</empty>

<sequence standard-attributes>

standard-elements

activity+

</sequence>

<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>

<while standard-attributes>

standard-elements

<condition expressionLanguage="anyURI"?>bool-expr</condition>

activity

</while>

<repeatUntil standard-attributes>

standard-elements

activity

<condition expressionLanguage="anyURI"?>bool-expr</condition>

</repeatUntil>

<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>

<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>

<flow standard-attributes>

standard-elements

<links>?

<link name="NCName" />+

</links>

activity+

</flow>

<scope isolated="yes|no"? exitOnStandardFault="yes|no"?

standard-attributes>

standard-elements

<partnerLinks>?

... see above under <process> for syntax ...

</partnerLinks>

<messageExchanges>?

... see above under <process> for syntax ...

</messageExchanges>

<variables>?

... see above under <process> for syntax ...

</variables>

<correlationSets>?

... see above under <process> for syntax ...

</correlationSets>

<faultHandlers>?

... see above under <process> for syntax ...

</faultHandlers>

<compensationHandler>?

...

</compensationHandler>

<terminationHandler>?

...

</terminationHandler>

<eventHandlers>?

... see above under <process> for syntax ...

</eventHandlers>

activity

</scope>

<compensateScope target="NCName" standard-attributes>

standard-elements

</compensateScope>

<compensate standard-attributes>

standard-elements

</compensate>

<exit standard-attributes>

standard-elements

</exit>

<rethrow standard-attributes>

standard-elements

</rethrow>

<extensionActivity>

<anyElementQName standard-attributes>

standard-elements

</anyElementQName>

</extensionActivity>

name="NCName"? suppressJoinFailure="yes|no"?

The "standard-elements" referenced above are:

<targets>?

<joinCondition expressionLanguage="anyURI"?>?

bool-expr

</joinCondition>

<target linkName="NCName" />+

</targets>

<sources>?

<source linkName="NCName">+

<transitionCondition expressionLanguage="anyURI"?>?

bool-expr

</transitionCondition>

</source>

</sources>

<plnk:partnerLinkType name="BuyerSellerLink">

<plnk:role name="Buyer" portType="buy:BuyerPortType" />

<plnk:role name="Seller" portType="sell:SellerPortType" />

</plnk:partnerLinkType>

<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>

<partnerLinks>

<partnerLink name="NCName"

partnerLinkType="QName"

myRole="NCName"?

partnerRole="NCName"?

initializePartnerRole="yes|no"? />+

</partnerLinks>

<sref:service-ref reference-scheme="http://example.org">

<foo:barEPR xmlns:foo="http://example.org">...</foo:barEPR>

</sref:service-ref>

<wsdl:definitions name="properties"

targetNamespace="http://example.com/properties.wsdl"

xmlns:tns="http://example.com/properties.wsdl"

xmlns:txtyp="http://example.com/taxTypes.xsd"

xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop"

xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<!-- import schema taxTypes.xsd -->

<!-- define a correlation property -->

<vprop:property name="taxpayerNumber" type="txtyp:SSN" />

...

</wsdl:definitions>

5.5. The Lifecycle of an Executable Business Process

As noted in the introduction, the interaction model that is directly supported by WSDL is essentially a stateless client-server model of request-response or uncorrelated one-way interactions. WS-BPEL, builds on WSDL by assuming that all external interactions of the business process occur through Web Service operations. However, WS-BPEL business processes represent stateful long-running interactions in which each interaction has a beginning, defined behavior during its lifetime, and an end. For example, in a supply chain, a seller's business process might offer a service that begins an interaction by accepting a purchase order through an input message, and then returns an acknowledgement to the buyer if the order can be fulfilled. It might later send further messages to the buyer, such as shipping notices and invoices. The seller's business process remembers the state of each such purchase order interaction separately from other similar interactions. This is necessary because a buyer might be carrying on many simultaneous purchase processes with the same seller. In short, a WS-BPEL business process definition can be thought of as a template for creating business process instances.

The creation of a process instance in WS-BPEL is always implicit; activities that receive messages (that is, <receive> activities and <pick> activities) can be annotated to indicate that the occurrence of that activity causes a new instance of the business process to be created. This is done by setting the createInstance attribute of such an activity to "yes". When a message is received by such an activity, an instance of the business process is created if it does not already exist (see sections 10.4. Providing Web Service Operations – Receive and Reply and 11.5. Selective Event Processing – Pick).

A start activity is a <receive> or a <pick> activity annotated with a createInstance="yes" attribute. [SA00015] Each executable business process MUST contain at least one start activity (see section 10.4. Providing Web Service Operations – Receive and Reply for more details on start activities).

If more than one start activity exists in a process and these start activities contain <correlations> then all such activities MUST share at least one common <correlation> (see the example in section 9.2. Declaring and Using Correlation Sets).

If a process contains exactly one start activity then the use of <correlationSets> is unconstrained. This includes a pick with multiple <onMessage> branches; each such branch can use different <correlationSets> or no <correlationSets>.

A business process instance ends either normally or abnormally. The process ends normally when the main activity and all event handler instances of the process complete without propagating any fault. The process ends abnormally if either:

a process level (explicit or default) fault handler completes without propagating any fault or

the execution of a process level fault handler itself faults (the effect of this particular case is similar to an <exit> activity) or

the process instance is explicitly ended by an <exit> activity (Immediately Ending a Process – Exit).

|  |  |  |
| --- | --- | --- |
| **WSDL 1.1 Term** | **WSDL 2.0 Term** | **Description** |
| Service | Service | Contains a set of system functions that have been exposed to the Web-based protocols. |
| Port | Endpoint | Defines the address or connection point to a Web service. It is typically represented by a simple HTTP URL string. |
| Binding | Binding | Specifies the interface and defines the [SOAP](https://en.wikipedia.org/wiki/SOAP) binding style ([RPC](https://en.wikipedia.org/wiki/Remote_procedure_call)/Document) and transport ([SOAP](https://en.wikipedia.org/wiki/SOAP) Protocol). The binding section also defines the operations. |
| PortType | Interface | Defines a Web service, the operations that can be performed, and the messages that are used to perform the operation. |
| Operation | Operation | Defines the SOAP actions and the way the message is encoded, for example, "literal." An operation is like a method or function call in a traditional programming language. |
| Message | n/a | Typically, a message corresponds to an operation. The message contains the information needed to perform the operation. Each message is made up of one or more logical parts. Each part is associated with a message-typing attribute. The message name attribute provides a unique name among all messages. The part name attribute provides a unique name among all the parts of the enclosing message. Parts are a description of the logical content of a message. In RPC binding, a binding may reference the name of a part in order to specify binding-specific information about the part. A part may represent a parameter in the message; the bindings define the actual meaning of the part. Messages were removed in WSDL 2.0, in which [XML](https://en.wikipedia.org/wiki/XML) schema types for defining bodies of inputs, outputs and faults are referred to simply and directly. |
| Types | Types | Describes the data. The [XML Schema](https://en.wikipedia.org/wiki/XML_Schema_(W3C)) language (also known as [XSD](https://en.wikipedia.org/wiki/XSD)) is used (inline or referenced) for this purpose. |

**How to Embed Java Code Snippets into a BPEL Process with the bpelx:exec Tag**

You can embed Java code snippets directly into the BPEL process using the Java BPEL exec extension bpelx:exec. The benefits of this approach are speed and transactionality. It is recommended that you incorporate only small segments of code. BPEL is about separation of business logic from implementation. If you remove a lot of Java code in your process, you lose that separation. Java embedding is recommended for short utility-like operations, rather than business code. Place the business logic elsewhere and call it from BPEL.

The server executes any snippet of Java code contained within a bpelx:exec activity, within its Java Transaction API (JTA) transaction context.The BPEL tag bpelx:exec converts Java exceptions into BPEL faults and then adds them into the BPEL process.The Java snippet can propagate its JTA transaction to session and entity beans that it calls.

For example, a SessionBeanSample.bpel file uses the bpelx:exec tag shown in below example to embed the invokeSessionBean Java bean:

<bpelx:exec name="invokeSessionBean" language="java" version="1.5">

<![CDATA[

try {

Object homeObj = lookup("ejb/session/CreditRating");

Class cls = Class.forName(

"com.otn.samples.sessionbean.CreditRatingServiceHome");

CreditRatingServiceHome ratingHome = (CreditRatingServiceHome)

PortableRemoteObject.narrow(homeObj,cls);

if (ratingHome == null) {

addAuditTrailEntry("Failed to lookup 'ejb.session.CreditRating'"

+ ". Ensure that the bean has been"

+ " successfully deployed");

return;

}

CreditRatingService ratingService = ratingHome.create();

// Retrieve ssn from scope

Element ssn =

(Element)getVariableData("input","payload","/ssn");

int rating = ratingService.getRating( ssn.getNodeValue() );

addAuditTrailEntry("Rating is: " + rating);

setVariableData("output", "payload",

"/tns:rating", new Integer(rating));

} catch (NamingException ne) {

addAuditTrailEntry(ne);

} catch (ClassNotFoundException cnfe) {

addAuditTrailEntry(cnfe);

} catch (CreateException ce) {

addAuditTrailEntry(ce);

} catch (RemoteException re) {

addAuditTrailEntry(re);

}

]]>

</bpelx:exec>

## Embedding Service Data Objects with bpelx:exec

You can embed service data object (SDO) code in the .bpel file with the bpelx:exec tag. In the syntax provided in the below example mytest.apps.SDOHelper is a Java class that modifies SDOs.

***Example : Embedding SDO Objects with the bpelx:exec tag***

</bpelx:exec>

<**bpelx:exec** name="ModifyInternalSDO" version="1.5" language="java">

<![CDATA[try{

Object o = getVariableData("VarSDO");

Object out = getVariableData("ExtSDO");

System.out.println("BPEL:Modify VarSDO... " + o + " ExtSDO: " + out);

**mytest.apps.SDOHelper.print(o);**

**mytest.apps.SDOHelper.print(out);**

**mytest.apps.SDOHelper.modifySDO(o);**

System.out.println("BPEL:After Modify VarSDO... " + o + " ExtSDO: " + out);

mytest.apps.SDOHelper.print(o);

mytest.apps.SDOHelper.print(out);

}catch(Exception e)

{

e.printStackTrace();

}]]>

</bpelx:exec>

Below example provides an example of the Java classes modifySDO(o) and print(o) that are embedded in the BPEL file.

***Example of the Java Classes***

public static void modifySDO(Object o){

if(o instanceof commonj.sdo.DataObject)

{

((DataObject)o).getChangeSummary().beginLogging();

SDOType type = (SDOType)((DataObject)o).getType();

HelperContext hCtx = type.getHelperContext();

List<DataObject> lines =

(List<DataObject>)((DataObject)o).get("line");

for (DataObject line: lines) {

line.set("eligibilityStatus", "Y");

}

} else {

System.out.println("SDOHelper.modifySDO(): " + o + " is not a

DataObject!");

}

}

. . .

. . .

public static void print(Object o) {

try{

if(o instanceof commonj.sdo.DataObject)

{

DataObject sdo = (commonj.sdo.DataObject)o;

SDOType type = (SDOType) sdo.getType();

HelperContext hCtx = type.getHelperContext();

System.out.println(hCtx.getXMLHelper().save(sdo, type.getURI(),

type.getName()));

} else {

System.out.println("SDOHelper.print(): Not a sdo " + o);

}

}catch(Exception e)

{

e.printStackTrace();

} }