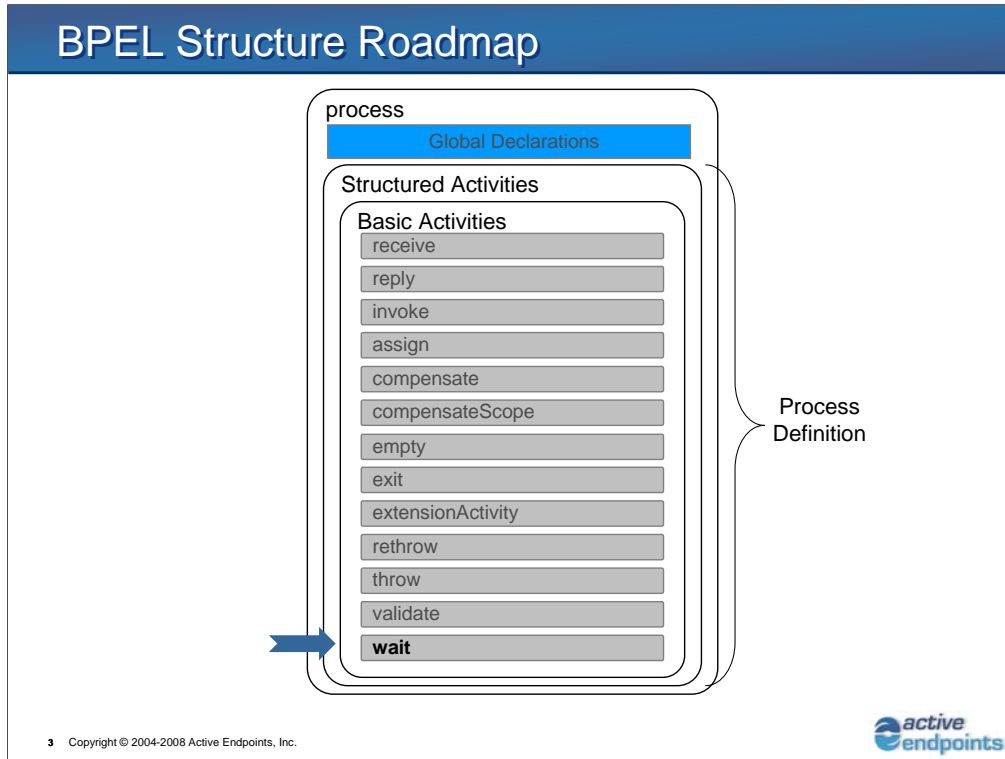




This is Unit #16 of the BPEL Fundamentals course. In past Units we've looked at ActiveVOS Designer, created the Process and made our global declarations. Next, we added interaction activities, Sequences, Assignments, Correlation and Scopes. Then we looked at the handlers: Fault, Compensation, Event and Termination. In the last unit, we examined the If activity, which allows us to do conditional processing. In this Unit we'll look at some more of BPEL's Basic activities and how they are used.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - Adding delays
 - Throwing and re-throwing faults
 - Doing nothing
 - Terminating processes
 - Validating Data
 - Extending BPEL



We'll start with the "wait" activity. The Wait activity is part of our process definition and is one of our Basic activities.

wait Activity Overview and Syntax

- Used to specify a certain delay in a business process
 - *For* a certain period of time
 - Duration-valued expression
 - *Until* a certain deadline is reached
 - Deadline-valued expression

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

4 Copyright © 2004-2008 Active Endpoints, Inc.



The “wait” activity is used to insert a delay into our processing. The delay can be based on either a Duration or on a Deadline, very much like an onAlarm activity. We use a “For” to set a duration, and an “Until” to set a deadline, again, like our onAlarm activity. Note that the delays are evaluated using an *expression* from any supported expression language.

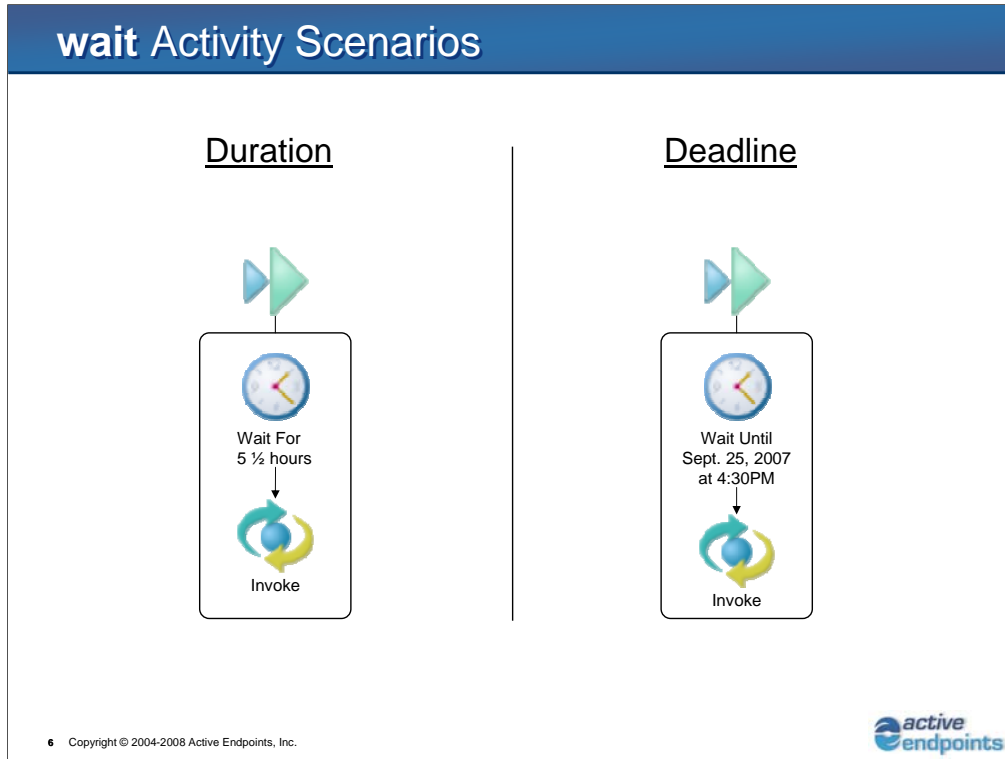
Deadlines and Duration Expressions

- Evaluation of these expressions must result in values being returned that are of one of the following XML Schema types
 - `dateTime` and `date` types for deadlines
 - `duration` type for duration
- Lexical representation of **`dateTime`**, **`date`**, and **`duration`** based on ISO 8601 standard
 - International standard for the representation of dates and times
 - <http://www.w3.org/TR/xmlschema-2/#isoformats>

5 Copyright © 2004-2008 Active Endpoints, Inc.



The expressions used by the Wait activity must evaluate to one of the data types shown: “`dateTime`” or “`date`” for a deadline and “`duration`” for a duration. As we have discussed previously, ActiveVOS uses ISO 8601 standards for the representation of dates and times.



Now let's look at two scenarios for the Wait activity, one that uses a Duration and one that uses a Deadline. On the left we are using a Duration, and the time starts when the Sequence activity that encloses it begins to execute. On the right is Sequence that uses a Deadline, and if you look closely you will see that it will expire immediately because the date used for the deadline has already passed.

wait Activity Example

<u>Duration</u>	<u>Deadline</u>
<pre><sequence> <wait> <for> 'PT5H30M' </for> </wait> <invoke .../> </sequence></pre>	<pre><sequence> <wait> <until> '2005-09-25T16:30:00' </until> </wait> <invoke .../> </sequence></pre>
<p>Where:</p> <ul style="list-style-type: none">P - time duration designatorT - time designatorH - follows the number of hoursM - follows the number of minutes	<p>Note: Time in above expression is UTC time To represent local time, specify the time zone offset after the time. For example 16:30:00-05:00 for 11:30 AM EST</p>

7 Copyright © 2004-2008 Active Endpoints, Inc.

Here is the syntax for a Wait activity that is being used with a Duration:

FORMAT: 'PTHM' // where "P" = duration follows, "T" = Time follows (after a date), "H" = Time's Hours follow, "M" = Time's Minutes follow

EXAMPLE: 'PT5H30M' // Duration is 5 hours and 30 minutes

wait Activity Semantics

- Exactly one of the expression criteria must be specified
 - Either `duration` or `deadline`
- `duration` and `deadline` expressions must be enclosed within single quotes

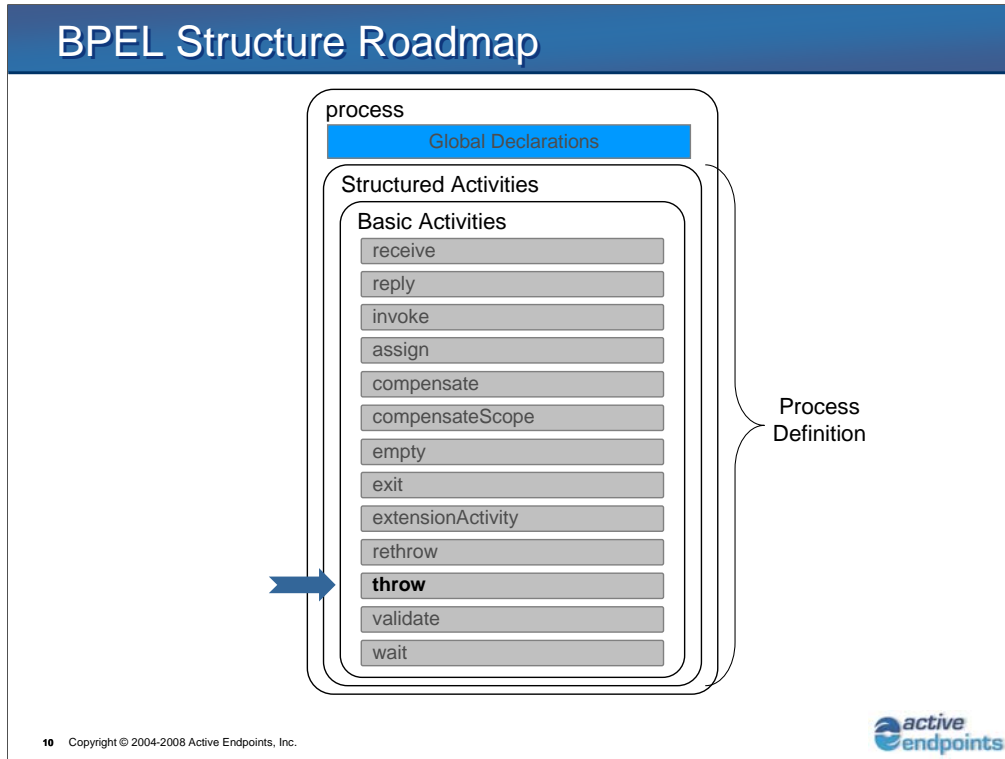
© Copyright © 2004-2008 Active Endpoints, Inc.



BPEL's Wait activities use either a Duration or a Deadline, and the Wait activity's expression always goes in single quotes.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - ✓ Adding delays
 - Throwing and re-throwing faults
 - Doing nothing
 - Terminating processes
 - Validating Data
 - Extending BPEL



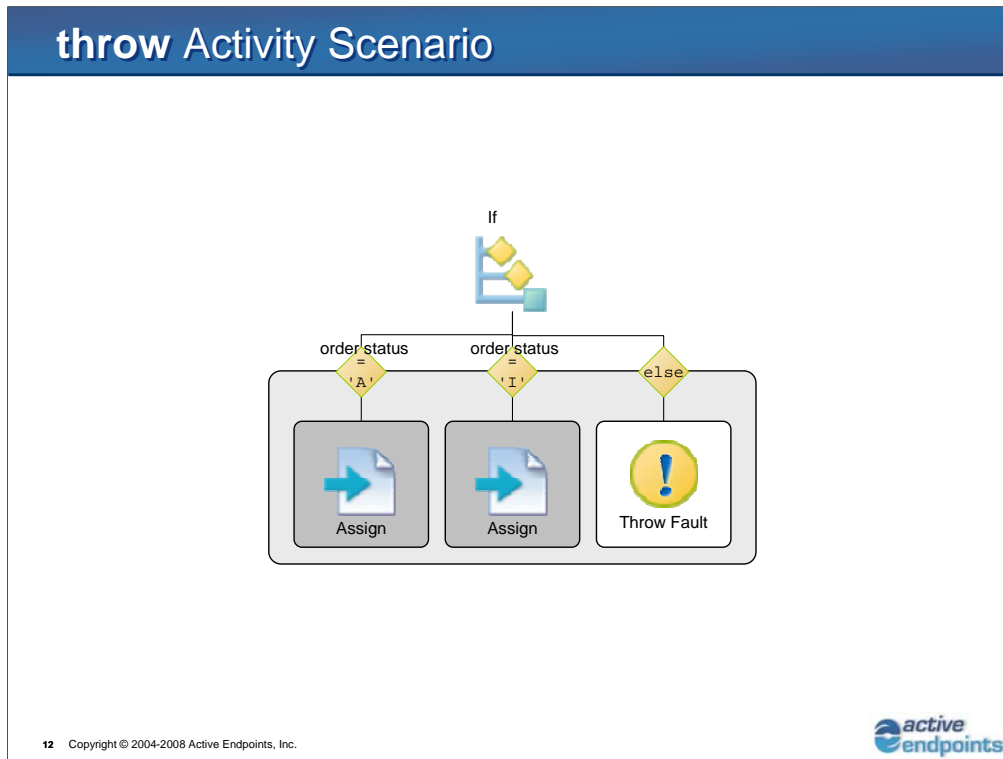
Next we will look at the Throw activity. It is one of our Basic activities and is part of our process definition.

throw Activity Overview and Syntax

- Used to explicitly signal an internal business fault
 - Indicate the name of the fault to be thrown
 - Optionally provide data about the fault via the `faultVariable` attribute

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

The purpose of a Throw activity is to signal an internal business fault, based on business logic. This is different, of course, from faults thrown by the system itself. When defining the Throw activity we have to specify the Fault name and (optionally) a `faultVariable` to provide fault data. As always, variables used by a BPEL process must be initialized before they are used.



Here is a situation where we might want to use a Throw...

Here we have an “if” activity...

- It checks the conditional statement against the value...
- If the value is ‘A’ it executes the first Assign activity...
- If the value is ‘I’ it executes the second Assign activity...

Here is a typical Throw activity scenario. If the value of the conditional statement is evaluated and matches neither of those two (i.e., "A" or "I") then the Else executes and Throws a fault.

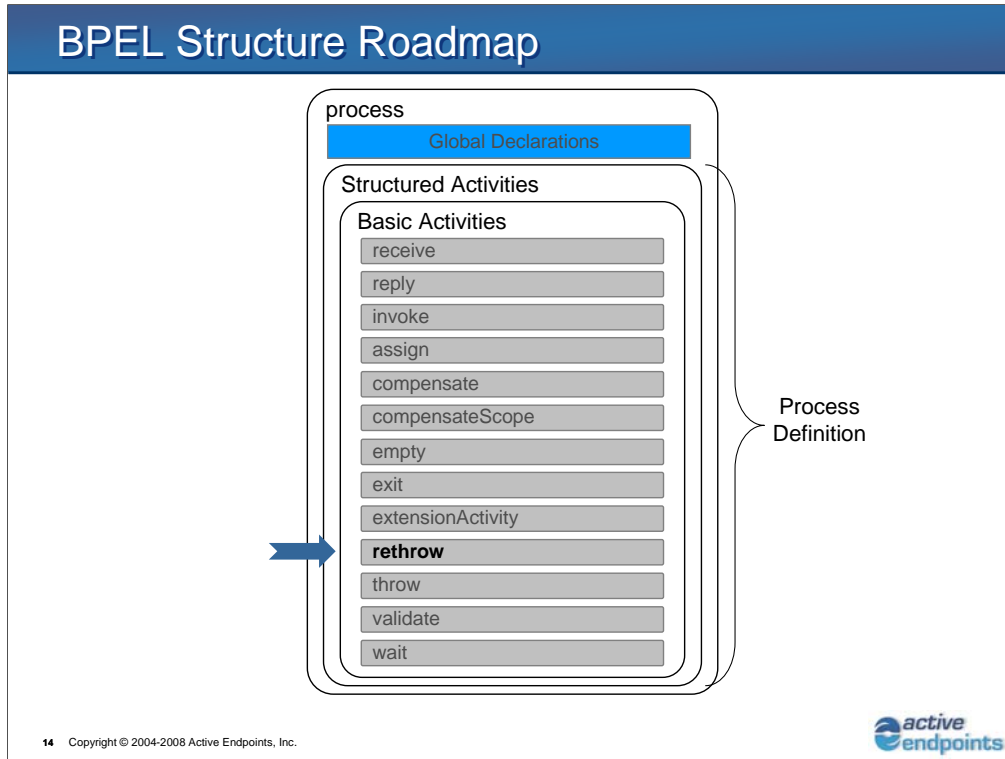
throw Activity Example

```
<if>
  <condition>$orderStatus='A'</condition>
  <assign ... />
  <elseif>
    <condition>$orderStatus='I'</condition>
    <assign ... />
  </elseif>
  <else>
    <throw faultName="ord:invalidProduct"
           faultVariable="productLookup" />
  </else>
</if>
```

13 Copyright © 2004-2008 Active Endpoints, Inc.



Here we see the Throw activity's syntax. The primary If conditional expression checks to see if "orderStatus" = 'A', then the conditional expression of the only elseif checks to see if "orderStatus" = 'I', and if both of these conditional statements evaluate to FALSE then we execute the else statement, which contains the Throw activity. If you look at the else you'll see that it has a Fully Qualified Name - "ord:invalidProduct" and an optional faultVariable called "productLookup"



Now that we've seen the Throw activity, let's look at the reThrow activity, which is closely associated with the Throw, obviously. The ReThrow is a Basic BPEL activity and it is part of our process definition.

rethrow Activity Overview and Syntax

- Used in fault handlers to re-throw the fault that was caught up to the enclosing scope (or process)

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

reThrow is used inside a fault handler to Throw a fault - that has been previously thrown - up one more level.

rethrow Activity Semantics

- Re-throws the *original* fault (and any fault data) caught by the fault handler
- Any modifications to data performed by the fault handler are ignored by **rethrow**, which re-throws the original fault data caught, *not* the modified fault data
- Can only appear inside a fault handler

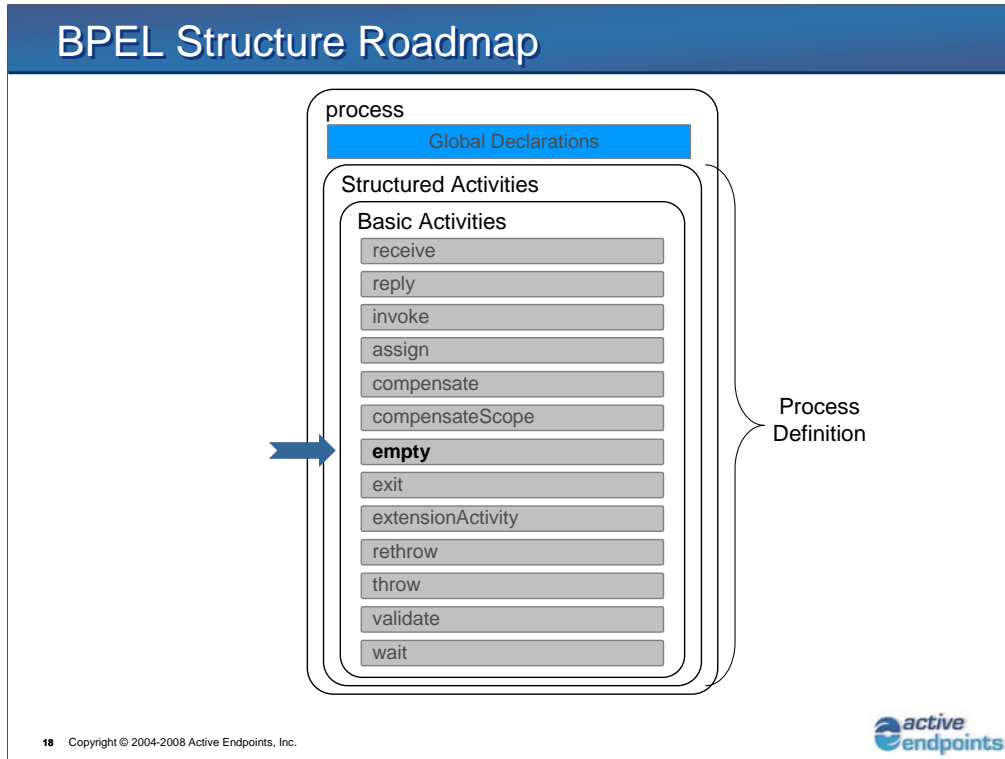
16 Copyright © 2004-2008 Active Endpoints, Inc.



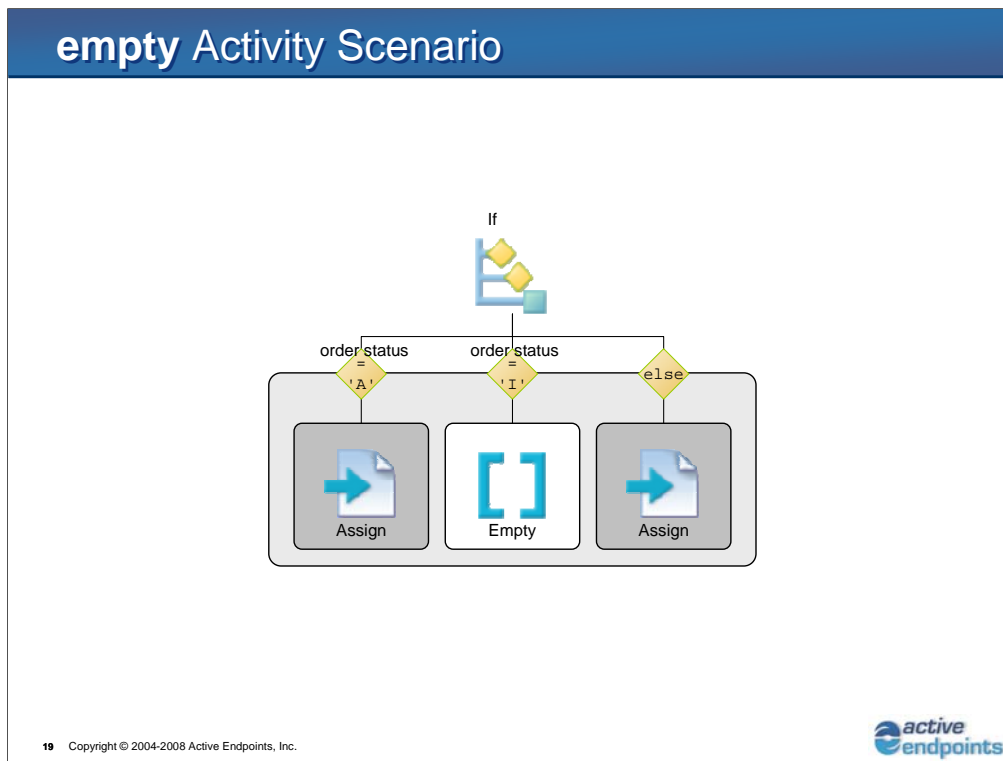
The way it works is that it throws the *original* fault and any *original* fault data that was there when it was first caught by the Fault Handler. We can take the fault data that came in with the Throw and manipulate it, but if the fault is reThrown, the *original* data will be thrown with it, ignoring any manipulation we've performed in the meantime. Note that the Re-throw activity can only be called from inside a Fault Handler.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - ✓ Adding delays
 - ✓ Throwing and re-throwing faults
 - Doing nothing
 - Terminating processes
 - Validating Data
 - Extending BPEL



Now let's take a look at the Empty activity. It is one of BPEL's Basic activities and it is part of our process definition.



Here is an example that uses the Empty activity. If the value of “orderStatus” = ‘A’ then we perform an Assign activity. If the value of “orderStatus” = ‘I’ then we execute the Empty activity, which does nothing. If the value is neither, then we execute the Else’s Assign activity.

empty Activity Example

```
<if>
  <condition>$orderStatus='A'</condition>
  <assign ... />
  <elseif>
    <condition>$orderStatus='I'</condition>
    <empty />
  </elseif>
  <else>
    <assign ... />
  </else>
</if>
```

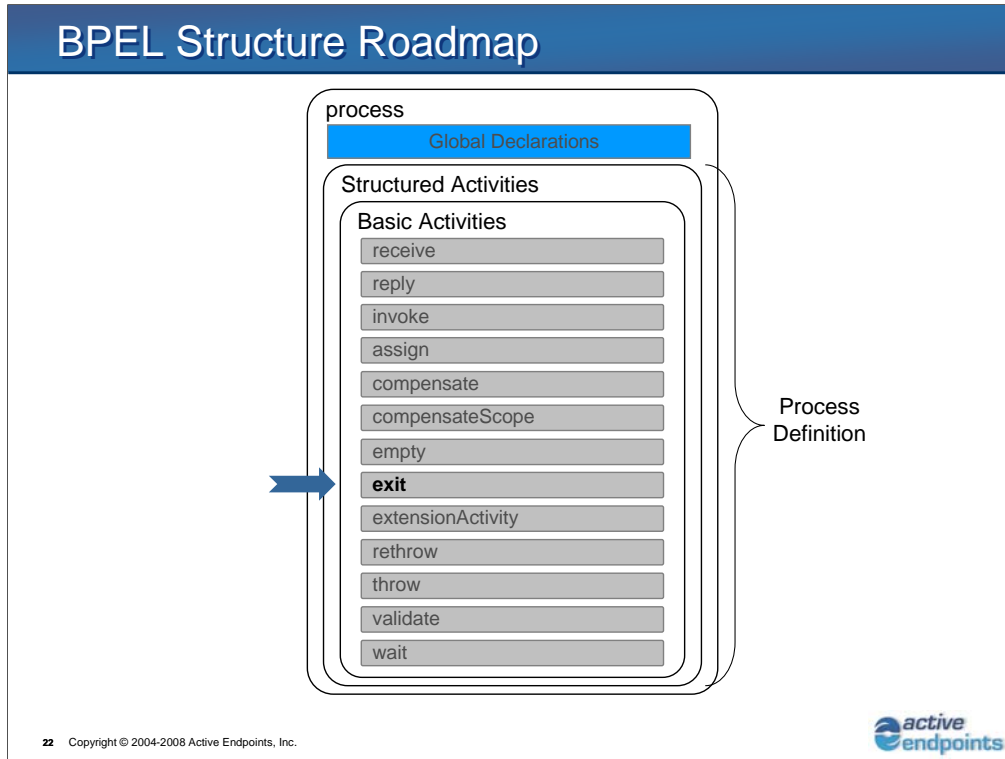
20 Copyright © 2004-2008 Active Endpoints, Inc.



Here is the syntax for the previous example that uses the Empty activity.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - ✓ Adding delays
 - ✓ Throwing and re-throwing faults
 - ✓ Doing nothing
 - Terminating processes
 - Validating Data
 - Extending BPEL



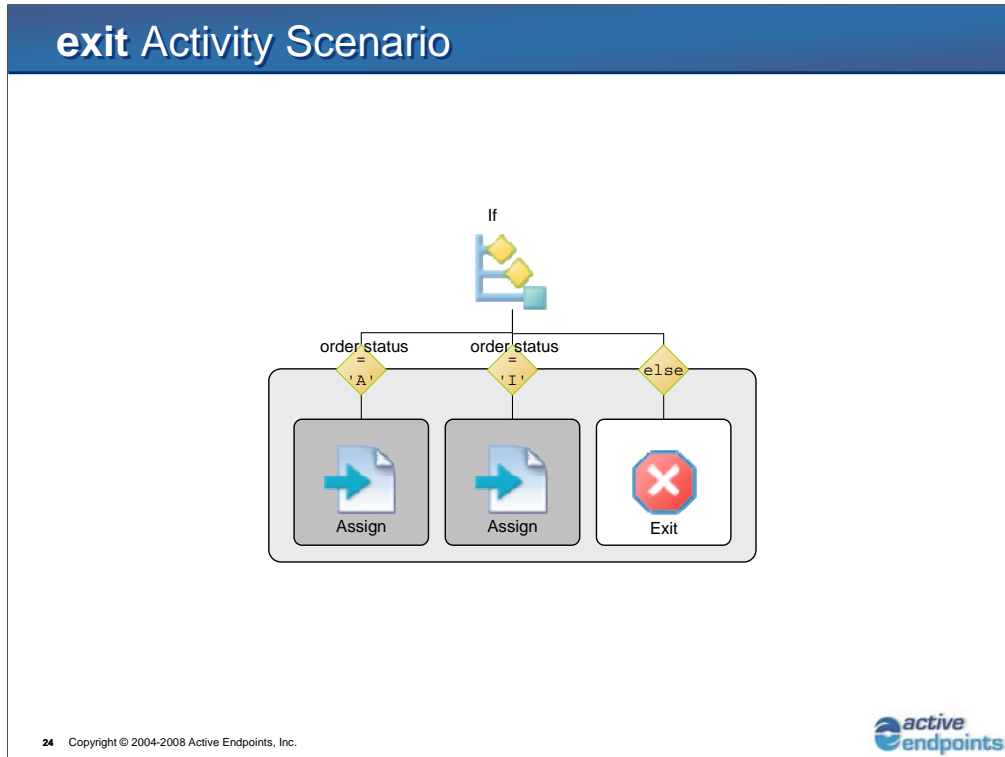
Next we will look at the Exit activity, which used to be the Terminate activity in BPEL 1.1 and became the Exit activity in the BPEL v2.0 specification. The Exit activity is one of BPEL's Basic activities and is part of our process definition.

exit Activity Overview and Syntax

- Used to immediately terminate a business process instance before it has properly completed

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

The Exit activity's purpose is to *immediately* terminate a business process before completion. There's nothing special about this Basic activity, and here's the syntax.



Here is a typical Exit activity scenario. We evaluate an Order Status variable and run it through the If statement. If the If conditional and the elseif conditional both evaluate to False, then the else statement is executed, which fires off the Exit activity.

exit Activity Example

```
<if>
  <condition>$orderStatus='A'</condition>
  <assign ... />
  <elseif>
    <condition>$orderStatus='I'</condition>
    <assign ... />
  </elseif>
  <else>
    <exit />
  </else>
</if>
```

And here is the syntax for the Exit activity example we saw on the previous slide.

exit Activity Semantics

- All currently running activities are halted immediately and the process ends
 - Without any fault handling, termination handling or compensation behavior

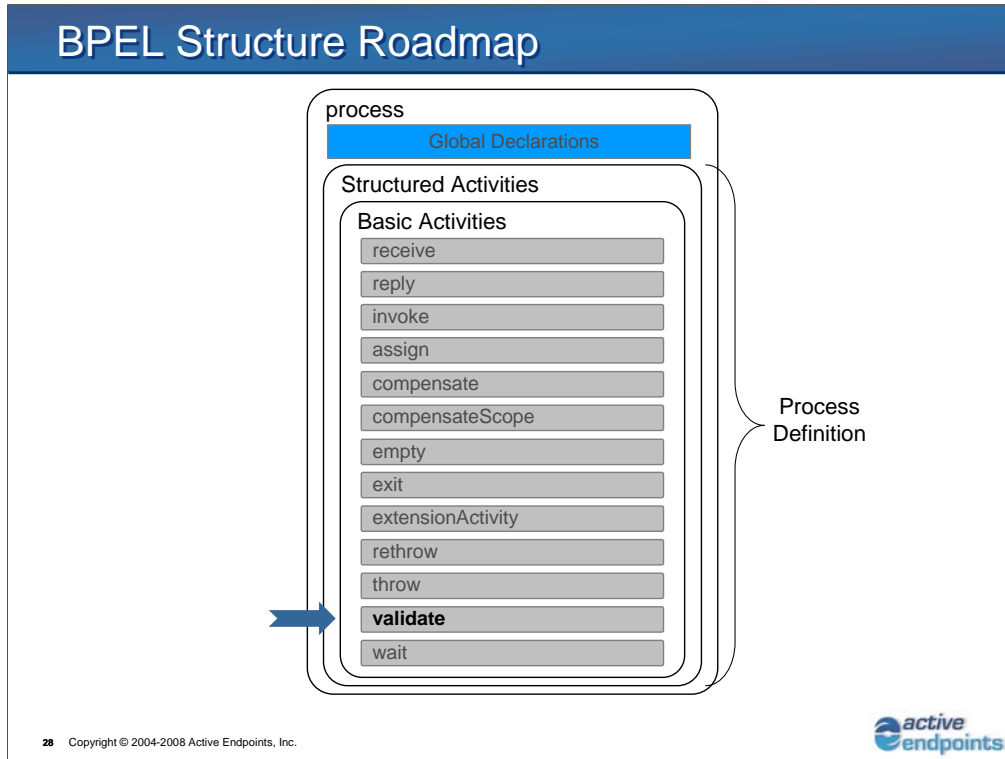
26 Copyright © 2004-2008 Active Endpoints, Inc.



When an Exit activity is fired, all currently running activities are halted, and the process ends *immediately* and *abnormally*. Nothing else happens, period. It is as if you pulled the plug. There is no Fault Handling, no Compensation Handling, no Termination Handling, no matter what is defined for any of these.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - ✓ Adding delays
 - ✓ Throwing and re-throwing faults
 - ✓ Doing nothing
 - ✓ Terminating processes
 - Validating Data
 - Extending BPEL



Now we'll look at the "Validate" activity, which is one of BPEL's Basic activities and is part of our process definition.

validate Activity Overview and Syntax

- Forces validation of one or more variables' contents against the schema for the variable type's definition

```
<validate variables="V1 V2 ..." standard-attributes >  
  standard-elements  
</validate>
```

The purpose of this activity is to validate one or more variables against their schema definitions. Note that the variables are entered in a list, separated by SPACES... which is a bit odd, since we're all used to seeing lists that are delimited by commas. This activity also has all the usual standard attributes or elements.

validate Activity Semantics

- **variables** attribute lists each variable to validate, separated by spaces
- Validation failures result in an **invalidVariables** fault

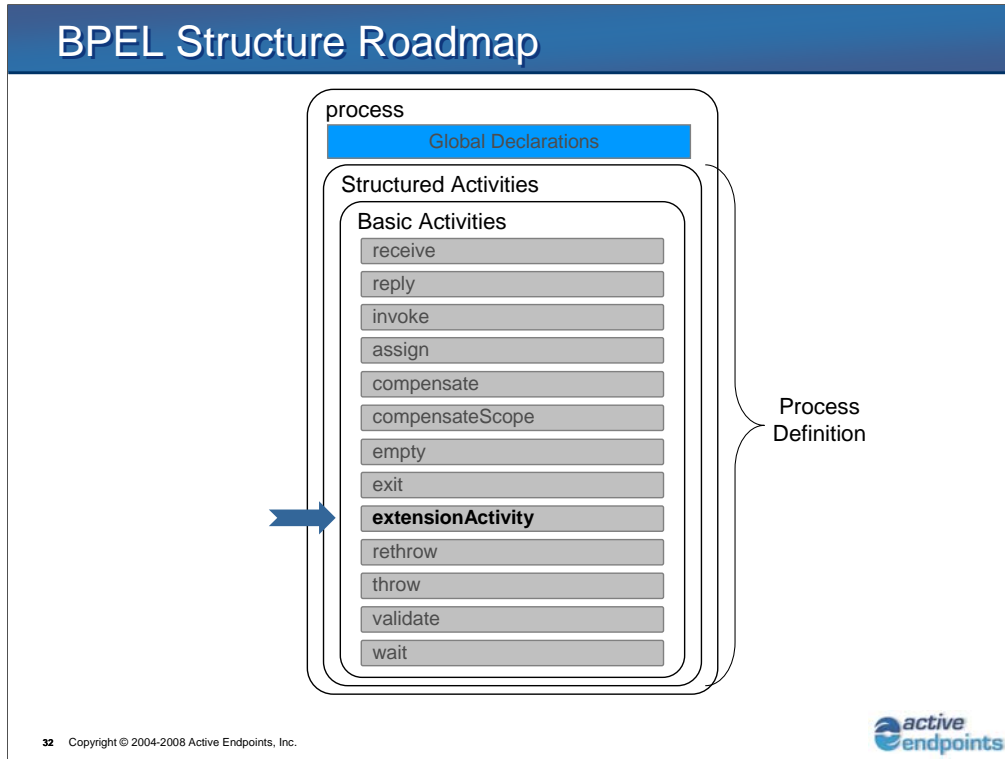
30 Copyright © 2004-2008 Active Endpoints, Inc.



Note again that the Variables list is *space* separated list. If any of the Variables in the list are invalid, the return is the “invalidVariables” fault. They are not validated one by one, but rather as a *set*. The obvious question is if there is a fault thrown how do we know which one caused it? The answer is you don’t. Validate your variables one by one if you need to know which one specifically is throwing the error.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - ✓ Adding delays
 - ✓ Throwing and re-throwing faults
 - ✓ Doing nothing
 - ✓ Terminating processes
 - ✓ Validating Data
 - Extending BPEL



Next we'll look at the "extensionActivity," which is a BPEL Basic activity, and is part of our process definition.

extensionActivity Overview

- Provides a standard way for BPEL vendors to extend WS-BPEL 2.0 with new activity types not defined in the specification
- Can be used to declare a new basic or structured extension activity

33 Copyright © 2004-2008 Active Endpoints, Inc.



So, how does a Vendor extend the BPEL specification? They can define a new *extension activity*, which can be either basic or structured.

extensionActivity Syntax

```
<extensionActivity standard-attributes>
  standard-elements
  <anyElementQname standard-attributes>
    standard-elements
  </anyElementQname>
</extensionActivity>
```

34 Copyright © 2004-2008 Active Endpoints, Inc.



Here's the syntax for an extensionActivity, and you'll notice that it has all of the standard activity attributes and elements. We need a Fully Qualified Name for the extensionActivity, and we place it where "anyElementQname" is located.

extensionActivity Semantics

- Contents of **extensionActivity** MUST be a single activity element which:
 - uses a qualified namespace, different from the WS-BPEL namespace and which is declared in the `extensions` declaration section of the process
 - supports standard BPEL attributes and elements
- Will be treated as an **<empty>** activity if:
 - it is not understood by the engine on which it's running
 - it is not subject to a `mustUnderstand="yes"` requirement from an `extensions` declaration
- If the extension contains nested activities, it should be subject to **`mustUnderstand="yes"`**

35 Copyright © 2004-2008 Active Endpoints, Inc.



Here are the semantics for an extension activity. It must be a single activity element and must be declared in a WSDL “extensions” section with a fully qualified namespace that is *NOT* the same as the WS-BPEL namespace. The extension activity must support standard BPEL attributes and elements.

So, here's how extensionActivity development works:

- 1.) Someone (a vendor) writes an extension activity (in Java or some other programming language.)
- 2.) Extension activity “awareness” is brought about when the extension’s code is added to BPEL-compliant engine’s source code.
- 3.) Only engines that have been “made aware” of the extension activity’s workings/code can run them (i.e., it must understand the activity).

Keeping this in mind, if the BPEL engine encounters an extension activity:

- If the “mustUnderstand” setting is set to “yes” and BPEL understands it, it executes the extensionActivity.
- If the “mustUnderstand” setting is set to “no” and BPEL understands it, it executes the extensionActivity.
- If the “mustUnderstand” setting is set to “yes”, and BPEL does not understand it, the activity is rejected and process terminates.
- If the “mustUnderstand” setting is set to “no”, and BPEL does not understand it, it is treated as an Empty activity.

If the activity has nested activities, they should also have `mustUnderstand = yes`, with the same results as above.

Lab 9 – Process Enhancements

- Overview of Lab Exercises
 - Add Audit functions
 - Add Credit Check functionality
 - Add a Conditional branch based on credit response

36 Copyright © 2004-2008 Active Endpoints, Inc.



The next Lab in the BPEL Fundamentals class is Lab #9.

- 1.) In this lab we will add an Audit function.
- 2.) We'll add a Credit Check function.
- 3.) Finally, we'll add a conditional branch statement, based on our credit response.

Lab 10 – Conditional with throw

- Overview of Lab Exercises
 - Add Invoke
 - shipping service
 - Enhance Fault Handler functionality

37 Copyright © 2004-2008 Active Endpoints, Inc.



The next lab in the BPEL Fundamentals course is Lab #10. In this lab we'll add the service that will ship the order, and we'll rearrange some process activities and add a fault handler for the scope.

Unit Summary

- Now you are familiar with:
 - Adding delays
 - Throwing and re-throwing faults
 - Doing nothing
 - Terminating processes
 - Validating Data
 - Extending BPEL