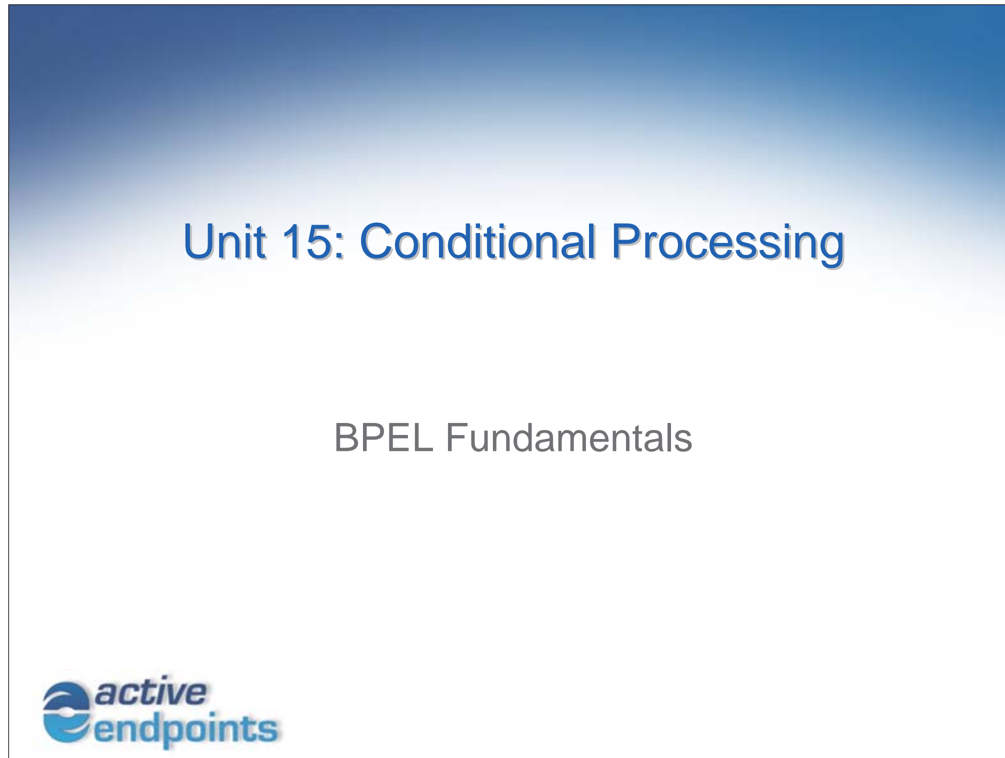


This is Unit #15 of the BPEL Fundamentals course. In past Units we've looked at ActiveBPEL Designer, Workspaces and Projects, created the Process itself and then declared our Imports, PartnerLinks and Variables and created Interaction Activities in various ways. Then, we looked at the Sequence activity, Assignments and Copies and then we studied Correlation, Scopes and Fault Handling. In our last three units we examined Compensation, Event Handling, and Termination Handlers. In this Unit we'll look at BPEL's Conditional Processing capabilities, which use the If activity.

Copyright © 2004-2007 Active Endpoints, Inc.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
  – Conditional processing

2 Copyright © 2004-2007 Active Endpoints, Inc.

active endpoints

Copyright © 2004-2007 Active Endpoints, Inc.

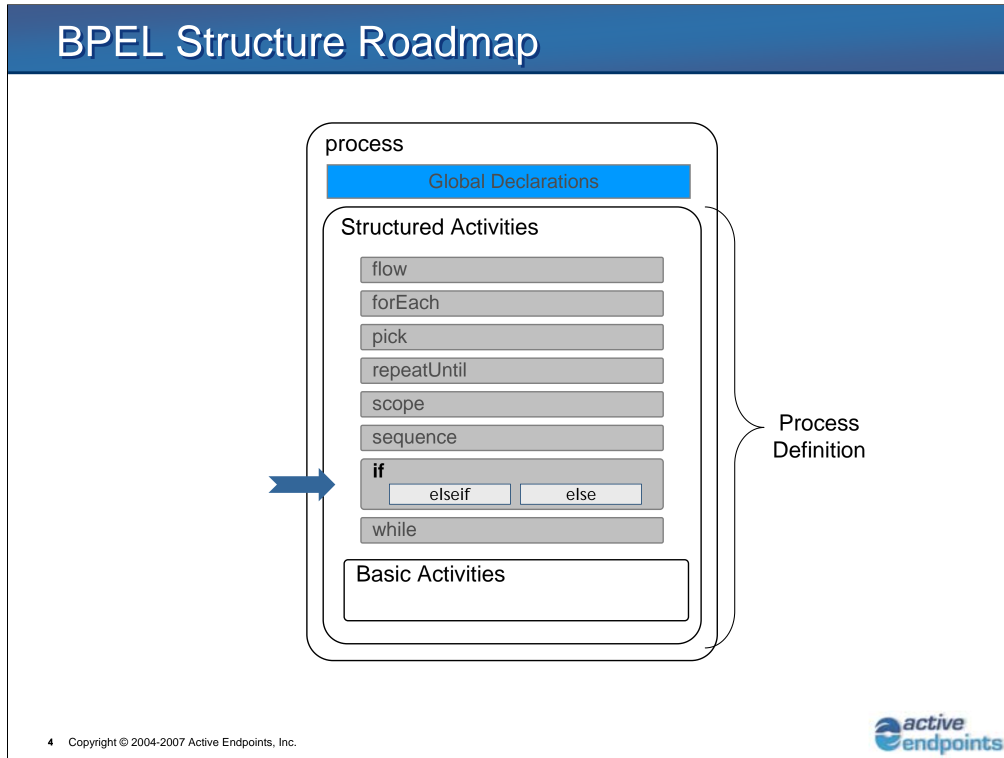
Conditional Processing Overview

- Conditional processing is used to choose one of a number of different branches within a process
  - `if` activity

 Copyright © 2004-2007 Active Endpoints, Inc.

In this Unit we will look at the If activity, which allows you to choose a process execution route, usually based on the process' data.

Copyright © 2004-2007 Active Endpoints, Inc.



The IF is a structured Activity, part of our process definition, and is built using an if-elseif-else format. In BPEL version 1.1 the conditional activity was implemented using a Switch statement. The If activity was added in BPEL version 2.0, while the Switch was removed.

Copyright © 2004-2007 Active Endpoints, Inc.

if Activity Overview

- Used to express a series of alternative choices, of which exactly one is executed
  - Primary conditional branch specified by `condition` element
  - Optionally followed by
    - Unlimited number of `elseif` branches
      - Each with its own `condition` expression
    - One `else` branch

 Copyright © 2004-2007 Active Endpoints, Inc.

The If activity lets you choose exactly one execution path from among many.

The Conditional Element controls the flow, branching on:

1.) an initial, single If statement, which has its own conditional expression

2.) then to zero, one or more elseifs, each of which has its own conditional expression

3.) a single else statement with no conditional expression

Copyright © 2004-2007 Active Endpoints, Inc.

if Activity Syntax

```
<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>
```

6 Copyright © 2004-2007 Active Endpoints, Inc.

Now let's take a look at the If Activity's syntax. It has all of the standard attributes and elements, and a conditional expression that determines whether the IF statement's activity is executed. The If's conditional expression – as well as all of the elseif's conditional expressions - **must** evaluate to a boolean, as evaluated by the designated expression language. The final Else statement has no conditional expression, and will be executed if no other condition expression evaluates to TRUE.

Copyright © 2004-2007 Active Endpoints, Inc.

Expressing Boolean-valued Conditions

- Uses the specified expression language
  - –Default language is XPath 1.0
    - Any valid expression that returns a boolean value can be used
- Variables are often used in boolean-valued conditions
  - –Variable values can be accessed using the *$<variableName>* syntax
  - –BPEL also defines the `getVariableProperty()` function that extracts arbitrary values from variables
    - One of the BPEL extensions to XPath functions

```
bpel:getVariableProperty ('variableName',
                          'propertyName')
```

7 Copyright © 2004-2007 Active Endpoints, Inc.

When creating conditional statements the default language is XPath1.0, and we also support JavaScript1.5 and XQuery1.0. Any expression that evaluates to a Boolean can be used. We can also access the values of our process and scope-level variables and use them to create a conditional statement. Alternatively, we can use the $*variable* format in our conditional expressions, and we can also use the "bpel:getVariableProperty ('variable name', 'property name') function to extract values from variable properties. Note that this is one of the BPEL extensions to XPath1.0 functions.

Copyright © 2004-2007 Active Endpoints, Inc.

getVariableProperty Function Example

WSDL Fragment

```
<definitions>
   <types>
      ...
   </types>

   <message name="StudentResponse">
      <part name="return" type="..." />
   </message>
   ...
   <vprop:property name="TestScore"
                   type="xsd:integer"/>
   <vprop:propertyAlias
      messageType="tns:StudentResponse"
      part="return"
      propertyName="tns:TestScore"
      query="/student/testScore"/>
   ...
</definitions>
```

BPEL Fragment

```
<process>
   ...
   <variables>
      <variable name="studentResp"
         messageType="ns:StudentResponse" />
   </variables>
   ...
</process>
```

XML Instance

```
<student>
   ...
   <testScore>83</testScore>
   ...
</student>
```

variableName *propertyName*

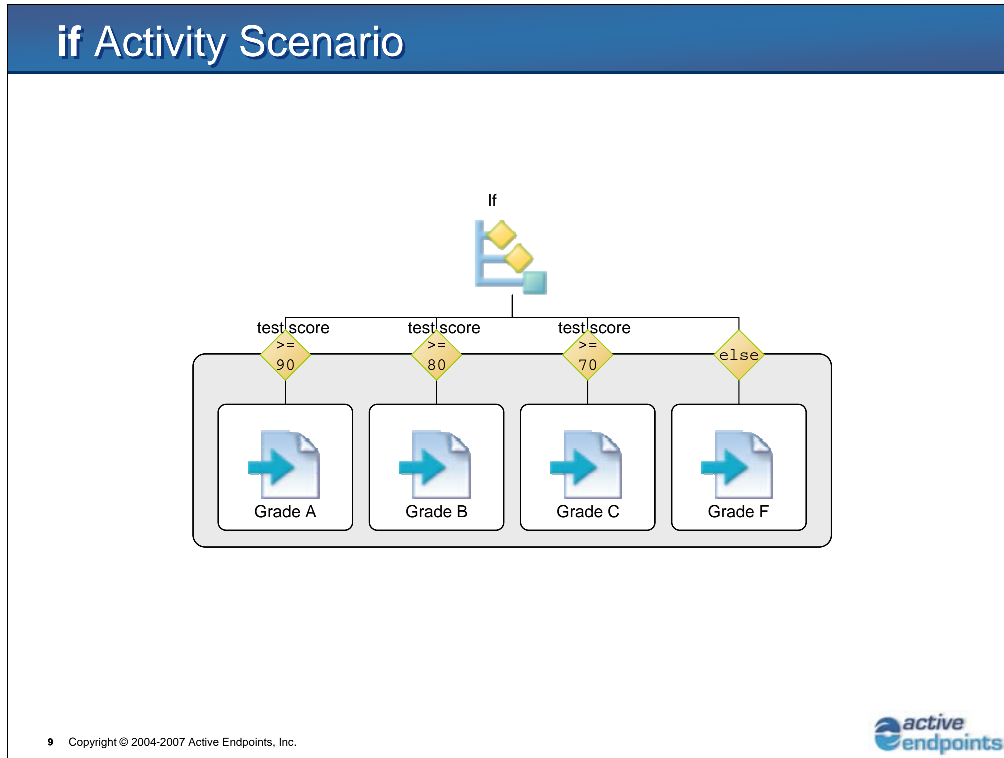
```
bpel:getVariableProperty('studentResp', 'TestScore') > 80
```

8 Copyright © 2004-2007 Active Endpoints, Inc.

active endpoints

Let's examine the "getVariable" function in BPEL. We have a BPEL fragment on the top right, that has a variable called “studentResp." It is a messageType variable, defined as “StudentResponse” in the WSDL. If we go to the WSDL fragment on the top left, we see the actual definition of our “StudentResponse” message. We see that the message “StudentResponse”, has a Part that is called “return." This variable has an integer property called “TestScore”, and one *property alias* that tells the system how to find a value for “return." For the messageType “tns:StudentResponse” it uses the query “/student/testScore” to get the value of the “return” Part. This query is run against the XML instance on the lower right, drilling into “student” to find the “testScore”, which in this example, is the value 83.

Copyright © 2004-2007 Active Endpoints, Inc.



This is how an “If” activity looks in ActiveBPEL Designer’s Process Editor. The evaluation of the If's conditionals moves from left to right. The mechanics of this activity are very much like those of a coin sorter, i.e., it is checking each slot for the *first* one – and *only* the first one - that matches.

So, in this example, it looks to see if the value is:

- first, greater than or equal to 90, for a grade of “A”

or

- second, less than 90 and greater than or equal to 80, for a grade of “B”

or

- third, less than 80 and greater than or equal to 70, for a grade of “C”

or

- fourth, less than 70, for a grade of “F”

Copyright © 2004-2007 Active Endpoints, Inc.

if Activity Example

```
<if>
   <condition>bpel:getVariableProperty(...) &gt;= 90</condition>
   <assign>
      <copy> <from><literal>A</literal></from> <to>...</to> </copy>
   </assign>
   <elseif>
      <condition>bpel:getVariableProperty(...) &gt;= 80</condition>
      <assign>
         <copy> <from><literal>B</literal></from> <to>...</to> </copy>
      </assign>
   </elseif>
   <elseif>
      <condition>bpel:getVariableProperty(...) &gt;= 70</condition>
      <assign>
         <copy> <from><literal>C</literal></from> <to>...</to> </copy>
      </assign>
   </elseif>
   <else>
      <assign ...>
   </else>
</if>
```

10 Copyright © 2004-2007 Active Endpoints, Inc.

Here is the syntax for the If activity. From a code view, Conditions are evaluated in lexical order. The Conditional expression for the “if” is on the top line, here using the getVariableProperty function to return a test score whose value will be part of the evaluation. Note that we use the string “>” as the escape sequence for the “>” greater than symbol. Similarly, we use an escape key: “<” for the “<“ less than symbol.

If the **first** conditional expression – i.e., for the If - evaluates to TRUE then we execute the Assign activity, which has a copy using a Literal that copies “A” to something else.

If the If conditional expression evaluates to FALSE, then we evaluate the **first elseif’s** condition – “>= 80“ - and if it evaluates to TRUE we copy “B” to something else.

If the **first elseif’s** condition evaluates to FALSE then we evaluate the **2nd elseif’s** condition – “>=70” - and if it is TRUE we copy “C” to something else.

If the 2nd elseif evaluates to FALSE then we execute the else statement, which contains an Assign activity, and somebody gets an “F.”

Copyright © 2004-2007 Active Endpoints, Inc.

if Activity Semantics

- **`if`** activity's **`condition`** is evaluated first and primary activity is executed if true
- **`elseif`** branches of the **`if`** are evaluated in the order in which they appear, if necessary
  - First branch that evaluates to true executes its primary activity
- Only one of the branches is ever executed
- If no **`else`** branch is specified then one is implied which does nothing

 Copyright © 2004-2007 Active Endpoints, Inc.

The "if" condition is evaluated first and if the condition is true then that statement's activities will be executed. Then the "elseif" conditions are evaluated, in order. The first of the conditionals that evaluates to TRUE will have its activities execute. Then the "Else" is evaluated, if no previous conditionals evaluated to TRUE. Remember that only the *first true* statement is executed. No other statement blocks are ever executed, even if they evaluate to true. The "Else" is optional, but implied. If it is not used, it is executed as an Empty activity.

Copyright © 2004-2007 Active Endpoints, Inc.

Unit Summary

- Now you are familiar with:
  - –Conditional Processing

12 Copyright © 2004-2007 Active Endpoints, Inc.

active endpoints

Copyright © 2004-2007 Active Endpoints, Inc.