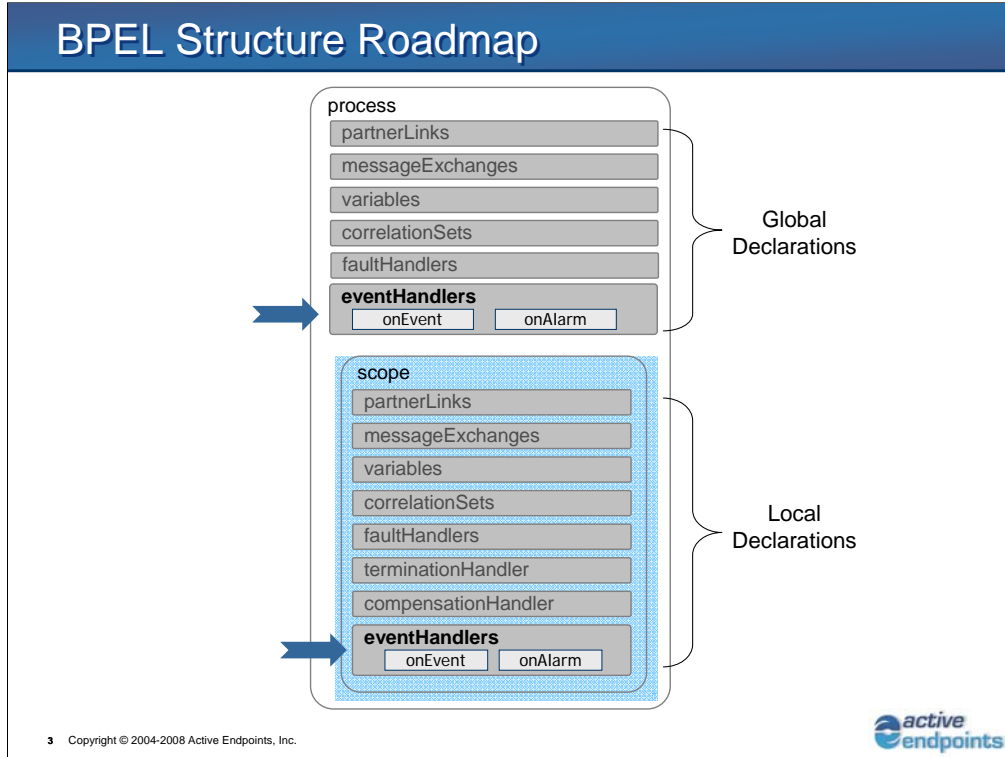




This is Unit #13 of the BPEL Fundamentals course. In past Units we've looked at ActiveVOS Designer, created the Process itself and then made our global declarations. After getting set up, we created the Interaction Activities, Sequences, Assignments and added Correlation. Scopes, Fault Handling and Compensation followed. Our next topic is Event Handlers.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - Event handlers
 - onEvent events
 - onAlarm events



As we look at our BPEL roadmap, we see that Event handlers are part of our process definitions and they can be defined at either the scope or process levels.

Event Handler Overview

- Used to allow a process to deal with events that may come about
 - Independent of, and asynchronously to, the process
 - Event handlers can be defined
 - For the `process`
 - For a `scope`
- There are two types of events that can occur
 - Message events
 - Triggered via incoming messages received from Web service operations
 - Alarm events
 - Triggered either for a certain period of time or until a certain deadline is reached

4 Copyright © 2004-2008 Active Endpoints, Inc.



Event Handlers define how the process will deal with events that occur independently of the process itself. They can be defined at either the process (global) or scope (local) levels and they remain active as long as their enclosing scope or process remains active. An Event Handler's event is triggered when a defined event occurs, either a message event or an alarm event. The message events are triggered by an incoming message, whereas Alarms are triggered by either a deadline or duration.

Note that Event Handlers are part of the “normal” behaviour of the scope, unlike CHs, FHs and THs, which are executed in response to an abnormal event that affects the process. So you could say that a fault handler is like a smoke alarm going off, and an event handler is like the phone ringing.

Event Handler Syntax

```

<eventHandlers>?
  <!-- Note: 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>*
    ( <for expressionLanguage="anyURI"?>duration-expr</for> |
      <until expressionLanguage="anyURI"?>deadline-expr</until> )?
    <repeatEvery expressionLanguage="anyURI"?>duration-expr
    </repeatEvery?>
    <scope ...>...</scope>
  </onAlarm>
</eventHandlers>

```

5 Copyright © 2004-2008 Active Endpoints, Inc.



On this slide we see the full syntax of a Event Handler, which has an onEvent definition section and an onAlarm definition section. You must have either an OnMessage or an OnAlarm defined for an Event Handler. Notice also that the Event Handler section can have a Correlation Set. This is necessary, of course, because the Event Handler is part of the conversation with one of our partners and the Correlation Set will continue to maintain the ongoing conversation's integrity, as we learned earlier. The onEvent section is followed by an "onAlarm" activity definition. The OnAlarm can be based on either a duration (e.g. wait for 3 hours) or a deadline (e.g. wait until 10AM) and has an optional "repeatEvery" setting which allows a duration alarm to reset after being triggered.

Event Handler Semantics

- Must contain at least one **onEvent** or **onAlarm**
 - Event handlers may be triggered simultaneously
 - They are performed concurrently
 - Event handlers can be rescheduled
- Enabled when the associated **scope** starts
 - If associated with the global `process` scope, then enabled as soon as the process instance is created
- Disabled when the associated **scope** completes
- **onEvent** and **onAlarm** require a **scope** as primary activity
- Faults within event handlers are treated the same as any other fault

6 Copyright © 2004-2008 Active Endpoints, Inc.



An Event Handler must have at least one `onEvent` or one `onAlarm`, and can have lots of them. Both `onEvents` and `onAlarms` can fire at the same time, they can be executed in parallel and they can be rescheduled, as necessary. The life of an Event Handler is same as the life of its scope, it is enabled when the scope is enabled, and then disabled when scope is disabled. The Primary activity of an Event Handler must be a Scope and any Faults in the Event handler are handled just like any other faults.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - ✓ Event handlers
 - onEvent events
 - onAlarm events

onEvent Overview

- Used to accept messages into a process that may
 - Impact processing behavior
 - Occur multiple times while the corresponding scope is active
 - Variable accepting the incoming message must be explicitly declared within the `onEvent`
- Executes in parallel with the **process or scope**
 - Has access to all `process` (or `scope`) variables
 - Usually requires the use of correlations

8 Copyright © 2004-2008 Active Endpoints, Inc.



The `onEvent` activity accepts messages into a process that can impact the execution of the process and that can occur multiple times during the life of the process/scope. It should also be noted that `onEvents` can have a variable to accept the incoming message's data if the variable is explicitly declared in the `onEvent`. When the `onEvent` is fired, it executes in parallel to the scope or process that encloses it. The `onEvent` activity can access all variables visible at the level of the `onEvent` (i.e., process or scope), and usually requires a Correlation Set to ensure conversational integrity with the partner who sent the message.

onEvent Syntax

```

<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>

```

© Copyright © 2004-2008 Active Endpoints, Inc.



Here is the syntax for an onEvent declaration. As you look this over, you might notice that it looks a lot like a Receive activity. It has a partnerLink, an optional portType, an Operation, a messageType and a Variable. Following this it has a section for the Correlation Set, the fromParts and then for the primary activity of the onEvent, which must be a scope.

onEvent Semantics

- Represents an event that waits for a message to arrive
 - When the message arrives, the primary activity specified in the corresponding handler is performed
- Attributes and semantics are the same as the attributes and semantics of the **receive** activity **except**
 - An `onEvent` can not specify the `createInstance` attribute

10 Copyright © 2004-2008 Active Endpoints, Inc.



Here is an overview of the `onEvent` activity's semantics. Note again that the `onEvent` activity is very much the same as a `Receive` activity, except that an `onEvent` activity cannot create a process instance. This is an Event handler, so in order to accept an "out of band" message it must have a process instance already running.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - ✓ Event handlers
 - ✓ onEvent events
 - onAlarm events

onAlarm Overview

- Used to make the process time-aware
 - May impact processing behavior
- Executes in parallel with the **process** or **scope**
- An alarm event can either be
 - *For* a certain period of time
 - Duration-valued expression
 - *Until* a certain deadline is reached
 - Deadline-valued expression
- Can optionally be rescheduled after being triggered using **repeatEvery** element
 - A duration-valued expression

12 Copyright © 2004-2008 Active Endpoints, Inc.



The purpose of the onAlarm activity is to make the process time aware – via a deadline or duration. The onAlarm activity executes in parallel with defined activities of the Process or Scope that encloses it. The Alarm can be triggered in one of two ways: *for* a certain period of time, i.e., a duration or *until* a certain time, i.e., at a deadline. OnAlarms can be fired more than once by using RepeatEvery element with a duration-valued expression only.

onAlarm Syntax

```
<onAlarm>*
  ( <for expressionLanguage="anyURI"?>
    duration-expr
  </for>
  |
  <until expressionLanguage="anyURI"?>
    deadline-expr
  </until> )?
  <repeatEvery expressionLanguage="anyURI"?>
    duration-expr
  </repeatEvery>?
  <scope ...>...</scope>
</onAlarm>
```

13 Copyright © 2004-2008 Active Endpoints, Inc.



Here is the syntax for onAlarm activity, showing the For/Until choice, then the optional "RepeatEvery" expression, followed by the primary activity of the onAlarm, which must be a scope.

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>

14 Copyright © 2004-2008 Active Endpoints, Inc.



The expressions used for our Deadlines and Durations must result in values that are of one of the following types: "duration," "dateTime" or "date." These values use the ISO 8601 International standard for the representation of dates and times. These standards can be referenced by going to the URL shown on the slide.

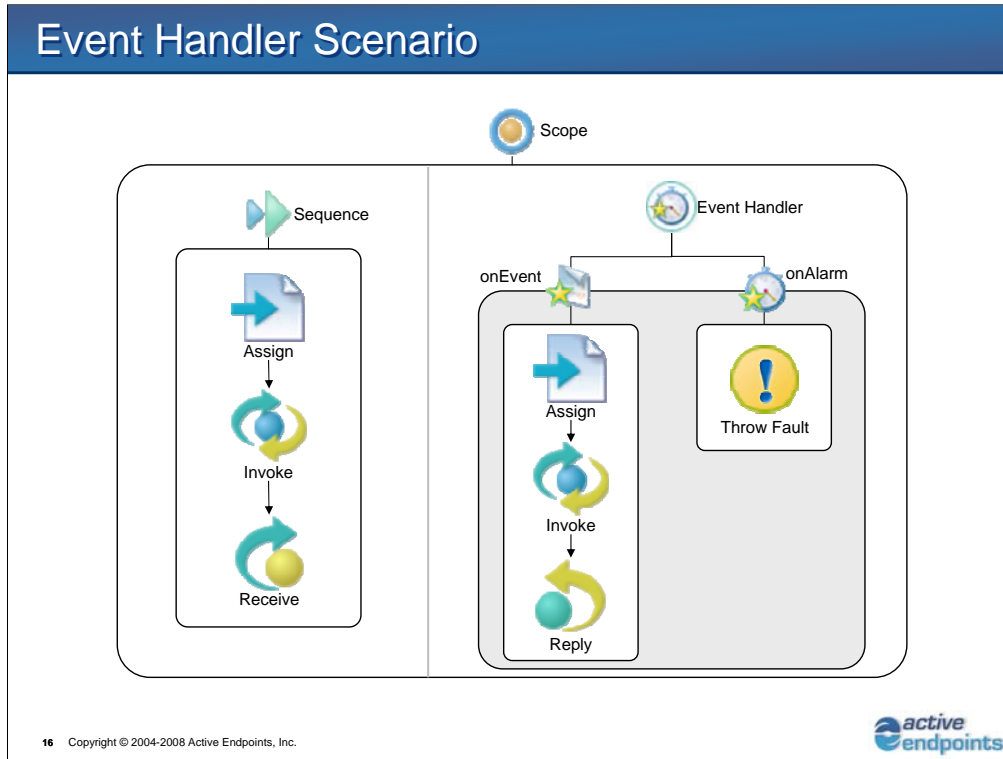
onAlarm Semantics

- For a duration-based **onAlarm** event
 - Counting of time starts when the enclosing event handler is activated
 - If optional `repeatEvery` element is used, retriggering occurs at the interval specified there
- An alarm event goes off when the specified deadline or duration has been reached

15 Copyright © 2004-2008 Active Endpoints, Inc.



Duration-based onAlarms begin counting the time to the firing of the alarm when the enclosing Event Handler is activated, which is exactly the same as when the scope is executed. If the optional "repeatEvery" element is used with the Duration, it starts counting from the first firing of the alarm. Any onAlarm-defined activity fires when the deadline or duration has been reached.



Note that a Scope context is required in order to define either an onEvent or an onAlarm activity. If we look at the scenario on the slide we see that on the left is the primary activity of the Scope – a Sequence with an Assign, an Invoke and a Receive. On the right hand side is the Event Handler, which has both onEvent and onAlarm definitions. In the middle is the primary activity of the onEvent, which is a Scope that has three activities defined: Assign, Invoke and Reply.

If we take a walk through the scenario above:

- The Scope starts, its primary Sequence starts, followed by the Sequence's Assign, Invoke and Receive activities.
- The event handler's are then executed *in parallel* with the execution of the scope's primary sequence's activities.
- Both the onEvent and onAlarm activities are now active, waiting for either an “out-of-band” message to be received or a timed event to occur.

Event Handler Example

```

<scope>
  <eventHandlers>
    <onEvent partnerLink="Customer" ...>
      <scope>
        <sequence>
          <assign .../>
          <invoke .../>
          <reply .../>
        </sequence>
      </scope>
    </onEvent>
    <onAlarm>
      <for>'PT24H'</for>
      <scope><throw .../></scope>
    </onAlarm>
  </eventHandlers>
  <sequence>
    <assign .../>
    <invoke .../>
    <receive .../>
  </sequence>
</scope>

```

Where:

- P - time duration designator
- T - time designator
- H - follows the number of hours

17 Copyright © 2004-2008 Active Endpoints, Inc.

Here we see the complete syntax for the example's Event Handler, which has both an onEvent and an onAlarm defined. The P, H, T designators are shown with their definitions. Below is the format for time designator's, with an example.

FORMAT: -yyyy-mm-ddThh:mm:ss.ss

EXAMPLE: 2007-05-17T10:15:20.35 = May 17, 2007 @ 10:15 AM + 20.35 seconds

"-" == optional "BCE" designator for years "Before Christian Era" (a "+" is not allowed)

yyyy == year

mm == month

dd == day

T == time follows

hh == hours

mm == minutes

ss == seconds

"." == fractions of seconds

zzzzzz == timezone

Lab 8 – Event Handler

- Overview of Lab Exercises
 - Add a global Event Handler to allow order status inquires

18 Copyright © 2004-2008 Active Endpoints, Inc.



The next Lab in the BPEL Fundamentals class is Lab #8. In this lab we will create a Process-level Event Handler:

- 1.) We'll add a Sequence, enclosed in a Scope and add an Event Handler.
- 2.) We'll add activities to the Event Handler that will send the current Order Status to the customer.
- 3.) Finally, we'll add this new set of activities to the Customer Correlation Set.

Unit Summary

- Now you are familiar with:
 - Event handlers
 - onEvent events
 - onAlarm events