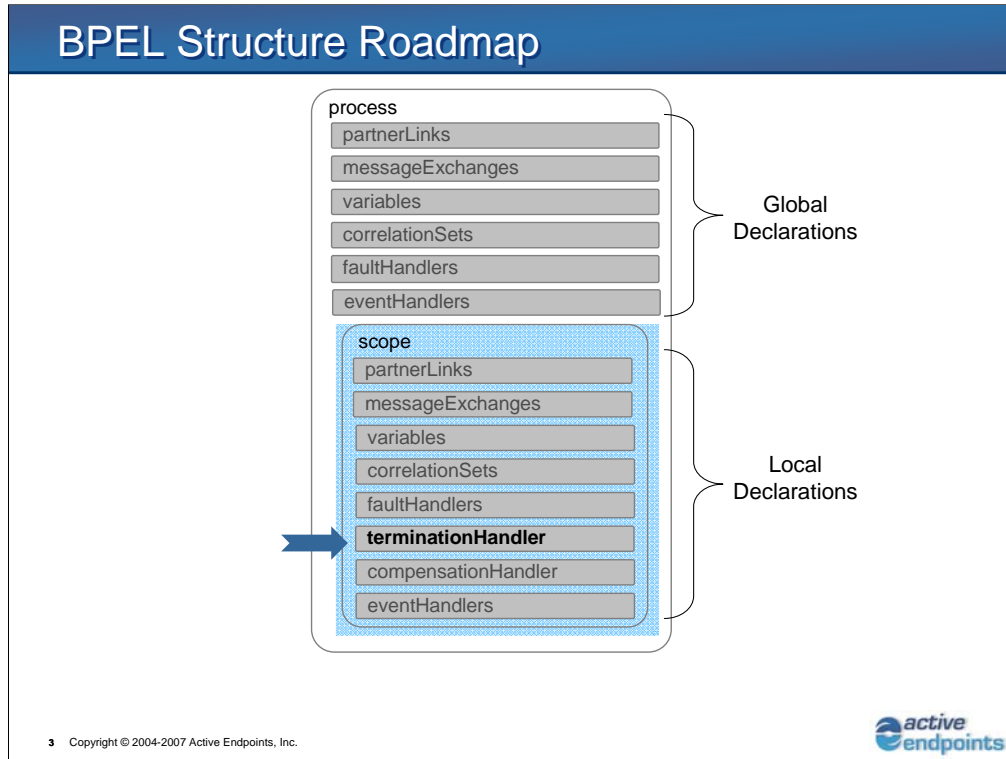




This is Unit #14 of the BPEL Fundamentals I course. In past Units we've looked at ActiveBPEL Designer, Workspaces and Projects, created the Process itself and then declared our Imports, PartnerLinks and Variables and created Interaction Activities in various ways. Then, we looked at the Sequence activity, Assignments and Copies and then we studied Correlation, Scopes and Fault Handling. In our last two units we examined Compensation and Event Handling, and in this Unit we will take a look at a new topic, Termination Handlers.

Unit Objectives

- At the conclusion of this unit, you will be familiar with:
 - Termination handlers



Termination Handlers are defined at the scope level and are part of our process definition.

Termination Handler Overview

- Used to allow a running scope to deal with forced termination events
 - Termination Handlers are only defined at a `scope` level
 - ActiveBPEL provides a process Termination handler extension for sub-process invocations
- Can use the set of activities available to a `scope`'s fault handler, including the `compensateScope` or `compensate` activity
- Is called prior to the parent `scope`'s fault handler executing
- Forced termination of nested `scopes` occurs in innermost-first order

4 Copyright © 2004-2007 Active Endpoints, Inc.



A Termination Handler controls the forced termination of a process. Its purpose is to allow a running scope to deal with a forced termination event, such as a fault that is not handled. Termination Handlers can call `Compensate` or `CompensateScope` as appropriate, and are designed to be called prior to the parent `Scope`'s fault handler being called. If we have a scope with an unhandled fault, the enclosing scope terminates. Forced termination begins execution in the most deeply nested scope, and then works its way out to top level scope, and finally to the process.

From the WS-BPEL v2.0 specification, Sec. 12.6:

“The forced termination of a scope begins by disabling the scope's event handlers and terminating its primary activity and all running event handler instances. Following this, the custom `<terminationHandler>` for the scope, if present, is run. Otherwise, the default termination handler is run. Forced termination for a scope applies only if the scope is in normal processing mode. If the scope has already invoked fault handling behavior, then the termination handler is uninstalled, and the forced termination has no effect. The already active fault handling is allowed to complete. If the fault handler itself throws a fault, this fault is propagated to the next enclosing scope.”

Q. Difference between Compensation Handler and Termination Handler?

Termination Handlers deal with forced scope termination caused by external faults.

Compensation Handlers deal with the persisted effects of already completed activities.

Termination Handler Syntax

```
<terminationHandler>  
  activity  
</terminationHandler>
```

5 Copyright © 2004-2007 Active Endpoints, Inc.



Termination Handler must contain one primary activity, which can be a structured activity. They are enabled when the scope starts, and disabled when the scope completes. The Termination Handler is handed control after all scope activities are terminated. Termination Handlers cannot throw a fault, even if one occurs during execution, and the next higher level of scope does not get a thrown fault in this situation, as with a Fault Handler.

Termination Handler Semantics

- Must contain one primary activity
- Enabled when the associated **scope** starts
- Disabled when the associated **scope** completes
- Is handed control after the termination of all the **scope**'s running activities and event handlers
- Can not throw a fault
 - Even if an uncaught fault occurs during its execution, it is not re-thrown to the next enclosing **scope**

6 Copyright © 2004-2007 Active Endpoints, Inc.



The Termination Handler was added to the BPEL specification in the 2.0 version, which was approved in April.

Unit Summary

- You are now familiar with:
 - Termination handlers