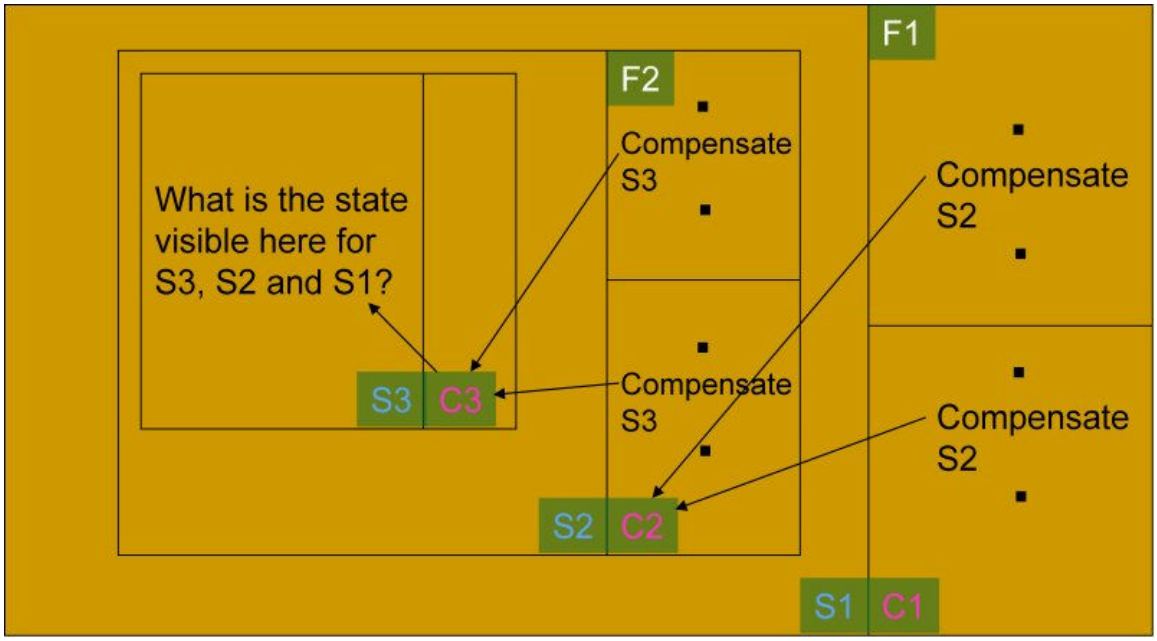


13.3.3. Invoking a Compensation Handler

The compensation handler can be invoked by using the `compensate` activity, which names the scope for which the compensation is to be performed, that is, the scope whose compensation handler is to be invoked. A compensation handler for a scope is available for invocation only when the scope completes normally. The first attempt to invoke a compensation handler that has never been installed is equivalent to the `empty` activity (it is a no-op) – this ensures that fault handlers do not have to rely on state to determine which nested scopes have completed successfully. Any subsequent attempts to invoke the never installed compensation handler MUST be ignored. An attempt to invoke an installed compensation handler more than once MUST be ignored too.

Note that in case an invoke activity has a compensation handler defined inline, the name of the activity is the name of the scope to be used in the `compensate` activity.

```
<compensate scope="ncname"? standard-attributes>
        standard-elements
</compensate>
```

13.3.3.1. Compensation of a Specific Scope

The ability to explicitly invoke the `compensate` activity is the underpinning of the application-controlled error-handling framework of WS-BPEL. This activity can be used only in

- a fault handler
- the compensation handler
- the termination handler

of the scope that immediately encloses the scope for which compensation is to be performed.

Example:

```
<compensate scope="RecordPayment"/>
```

All scopes and activities directly nested in a scope MUST be uniquely named. If the value of the scope attribute specified on a compensate activity does not resolve to a unique scope or activity name in the same scope as the compensate activity, the BPEL definition MUST be rejected from processing. This requirement MUST be statically enforced.

If the explicit compensation handler for a scope is absent, the default compensation handler for the scope is invoked.

An invocation of a compensate activity occurs in a fault, compensation, or termination handler (in the rest of this paragraph jointly called “handlers”) and is used to compensate the behavior of a successfully completed scope nested immediately inside the scope associated with the handler. However, handlers may themselves contain scopes, and therefore it is necessary to be more precise. The invocation of a compensate activity is interpreted based on the *nearest* enclosing handler and is used to compensate the behavior of a successfully completed scope nested immediately inside the scope associated with that handler. There is therefore no way to use a compensate activity to compensate the scopes nested immediately inside a handler . The value of the scope attribute on a compensate activity MUST NOT resolve to the name of a scope nested immediately inside a handler. This rule MUST be statically enforced.

Section 13.3.4 “Compensation for Scopes in Repeatable Constructs or Handlers” and section 13.3.5 “Compensation Handler Instance Group” describe how scope-specific compensation logic happens when a scope is executed repetitively.

Section 13.3.6 “Compensation within Handlers” ~~Section 13.3.4. Compensation for Scopes in Repeatable Constructs or Handlers~~ describes the compensation behavior for scopes nested inside of handlers in detail.

13.3.3.2. Default Compensation Behavior

The `<compensate/>` form, in which the scope name is omitted in a `compensate` activity, causes all child scopes to be compensated in the default order. This is useful when an enclosing fault or compensation handler needs to perform additional work, such as updating variables or sending external notifications, in addition to performing default compensation for the associated scope. Note that the `<compensate/>` activity in a fault or compensation handler attached to scope S causes the default-order invocation of compensation handlers for completed scopes directly nested within S. The use of this activity can be mixed with any other user-specified behavior except the explicit invocation of `<compensate scope="Sx"/>` for scope Sx nested directly within S.

Explicit invocation of compensation for such a scope nested within S disables the availability of default-order compensation, as expected.

Section 13.3.5 "Compensation Handler Instance Group" describes how to handle a fault or termination during execution of a default compensation.

13.3.4. Compensation for Scopes in Repeatable Constructs or Handlers

Multiple instances of a scope may exist for scopes nested inside of repeatable activities or event handlers, one for each repetition or event handler instantiation, respectively. When a scope-specific compensation activity is executed with such a scope as the target, all installed instances will be treated as one unit called Compensation Handler Instance Group. More details are available in the section below.

13.3.5 Compensation Handler Instance Group

When `<compensate/>` activity (i.e. default compensation) is used or `<compensateScope>` activity (i.e. scope specific compensation) is used on a scope contained by a ~~***Definition*: Compensation Handler Instance Group**. For scopes nested inside of~~ repeatable construct~~s~~, such as a loop~~s~~ or an event handler~~s~~, the compensation activity will attempt to run a set of installed compensation handler instances and hence will cause the corresponding set of child scope instances to be compensated. ~~the~~ The set of all installed compensation handler instantiations ~~of a particular scope~~ is called a ~~*c*~~*Compensation* ~~*h*~~*Handler* ~~*i*~~*Instance* ~~*g*~~*Group*. For the case of default compensation, the Compensation Handler Instance Group contains the compensation handler of all child scope instances that complete successfully. For the case of scope specific compensation, the Compensation Handler Instance Group contains the installed compensation handler instances of a particular target scope that is executed within a repeatable construct. For the purpose of this definition, in nested groups without intervening scopes, all of the compensation instances are treated as a single flat group and not as a nested group.[SRT18]

Compensation handler instance groups are treated as a unit. If ~~a~~an uncaught fault occurs in any compensation handler instance of the group or termination of compensation activities occurs, then all running instances are terminated following standard BPEL activity termination semantics. All the compensation work done by finished compensation instances in the group is kept. Finally, all compensation handler instances of the group and compensation handler instance groups of nested scopes are uninstalled[SRT22].

If a scope being compensated by name was nested in a while, repeatUntil, or non-parallel forEach loop, the installed instances of the compensation handlers in the successive iterations are invoked in reverse completion order.

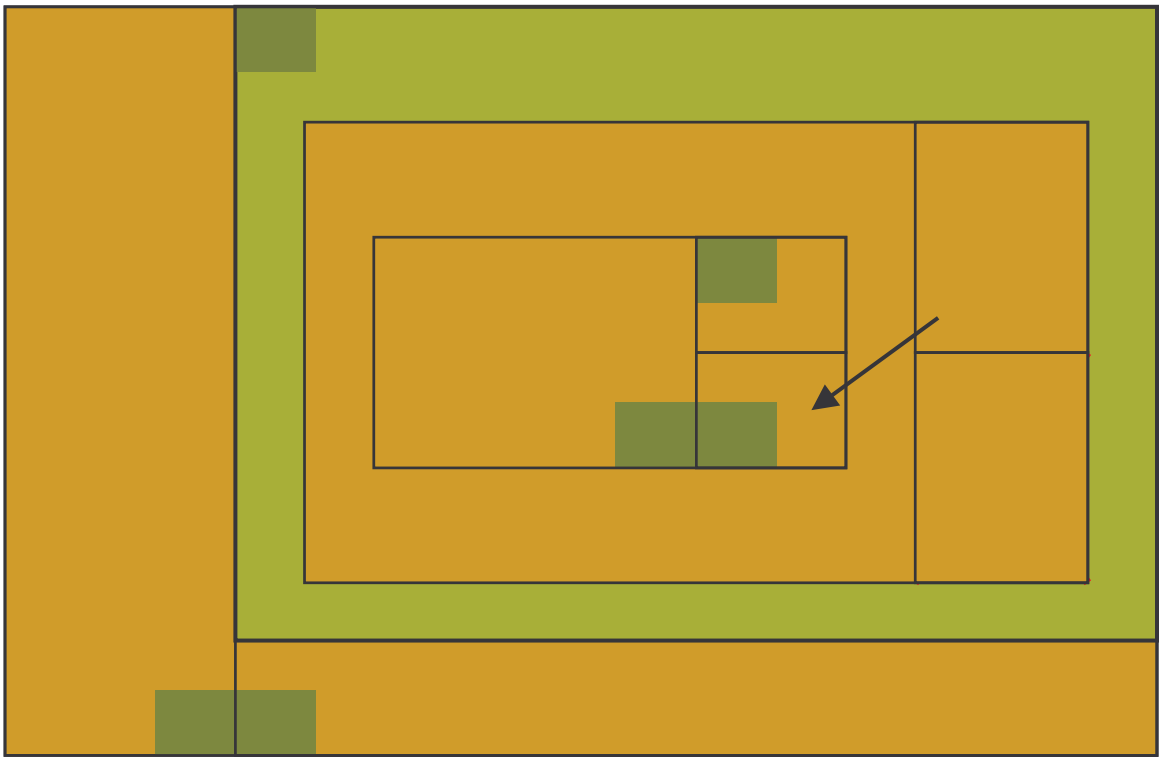
For parallel forEach and event handlers, compensation of the associated scope will cause all installed compensation handler instances to be run for iterations of parallel forEach and instances of the event handler, respectively. No ordering requirement is imposed.

13.3.6 Compensation within Handlers

13.3.46.1. Compensation for Scopes nested within Fault Handlers

If a scope is nested inside of a fault handler then the scope's compensation handler is available only for the lifetime of the enclosing fault handler. If the fault handler is left then any installed compensation handlers within it are uninstalled. Especially, a root scope nested inside a fault handler cannot have a compensation handler associated because it is not reachable at all from anywhere within the process. Therefore, the root scope inside a fault handler MUST not have a compensation handler. This rule MUST be statically enforced.

The picture below shows a fault handler F1 that contains a scope S2. Scope S2 cannot have a compensation handler as this handler would be unreachable, but it may have a fault handler F2 that is allowed to compensate an inner scope S3.



A[SRT27] fault in a fault handler causes all running contained activities to be terminated as specified in section 13.4.4. Semantics of Activity Termination. All compensation handlers contained in the fault handler are simply uninstalled (not run), and the fault is propagated to the parent scope.

13.3.64.2. Compensation for Scopes nested within Compensation Handlers

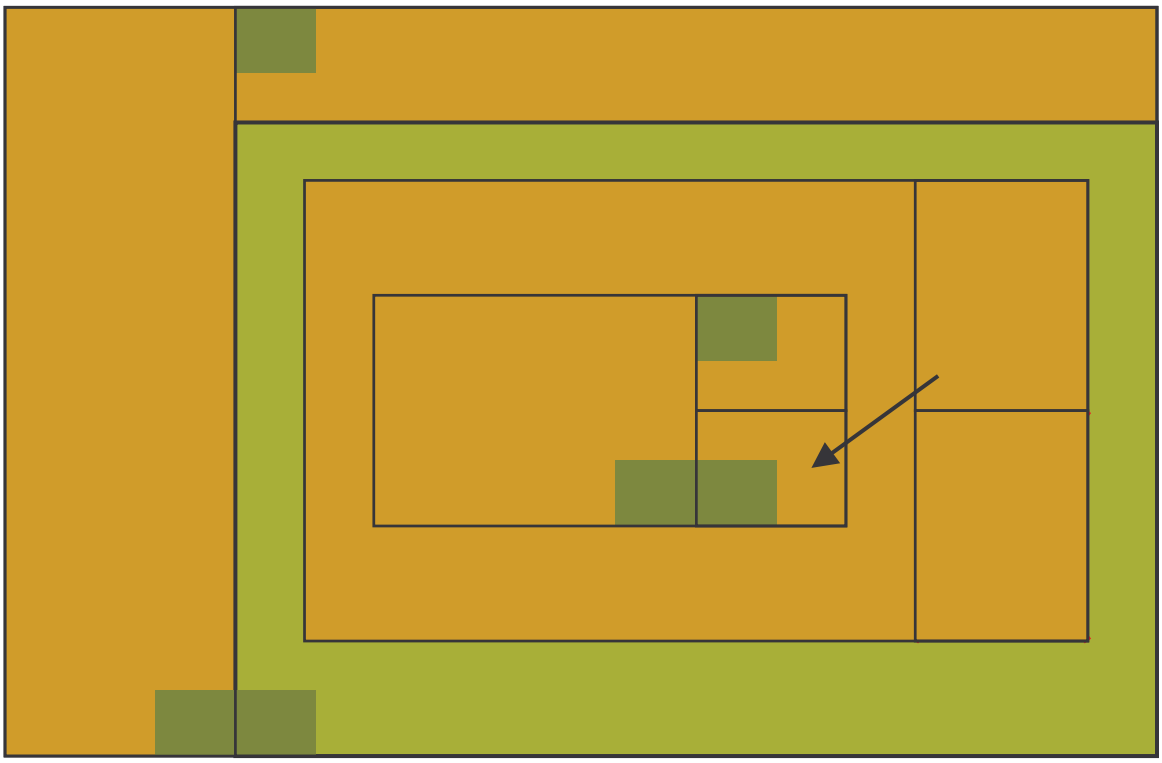
If a scope is nested inside a compensation handler then the scope's compensation handler is available only for the lifetime of the enclosing compensation handler. It can be used to

ensure “all or nothing” semantics for compensation handlers, but not for reversing the work of a successfully completed compensation handler. If the compensation handler completes successfully then any installed compensation handlers for scopes nested within it are uninstalled. Especially, a root scope nested inside a compensation handler cannot have a compensation handler associated because it is not reachable at all from anywhere within the process. Therefore, the root scope inside a compensation handler MUST not have a compensation handler. This rule MUST be statically enforced.

A compensation handler that faults (“internal fault”) will undo its partial work by compensating all child scopes to be compensated in the default order, similar to the “default compensation behavior” described above. This approach can be used to provide all or nothing semantics for compensation handlers. The compensation handler is then uninstalled, and the fault will propagate to the caller of the compensation handler. This caller may be a default handler or a custom handler containing a compensate activity.

If a running compensate activity is terminated due to a fault in the environment in which it was called (“external fault”) then the termination of the work of the compensate activity occurs analogous to the default termination for a scope and other WS-BPEL activities.

The picture below shows a compensation handler C1 that contains a scope S2. Again, S2 cannot have a compensation handler but may have a fault handler F2 that is allowed to compensate an inner scope S3.



13.3.46.3. Compensation for Scopes nested within Termination Handlers

A fault inside a termination handler is not propagated to the parent. Other than that all of the statements in 13.3.4.1 about fault handlers apply to termination handlers.

13.4. Fault Handlers

Fault handling in a business process can be thought of as a mode switch from the normal processing in a scope. Fault handling in WS-BPEL is always treated as "reverse work" in that its sole aim is to undo the partial and unsuccessful work of a scope in which a fault has occurred. The completion of the activity of a fault handler, even when it does not rethrow the fault handled, is never considered successful completion of the attached scope and compensation is never enabled for a scope that has had an associated fault handler invoked.

The optional fault handlers attached to a scope provide a way to define a set of custom fault-handling activities, syntactically defined as `catch` activities. Each `catch` activity is defined to intercept a specific kind of fault, defined by a globally unique fault QName and a variable for the data associated with the fault. If the fault name is missing, then the catch will intercept all faults with the right type of fault data. The fault variable is specified using the faultVariable attribute in a catch handler. The variable is deemed to be declared by virtue of being used as the value of this attribute and is local to the fault handler. It is not visible or usable outside the fault handler in which it is declared. The fault variable is optional because a fault might not have additional data associated with it.

A fault response to an invoke activity is one source of faults, with name and data aspects based on the definition of the fault in the WSDL operation (please refer to the Implementer’s Note in Section 11.3 for important details regarding access to fault information when faults are transmitted using SOAP). A programmatic `throw` activity is another source, again with explicitly given name and data. The core concepts and executable pattern extensions of WS-BPEL define several standard faults with their names and data, and there might be other platform-specific faults such as communication failures that can occur in a business process instance. A `catchAll` clause can be added to catch any fault not caught by a more specific fault handler.

```
<faultHandlers>?
      <!-- there must be at least one fault handler or default -->
     <catch faultName="qname"? faultVariable="ncname"?
                               faultMessageType="qname"?
                               faultElement="qname"?>*
        activity
     </catch>
     <catchAll>?
        activity
    </catchAll>
</faultHandlers>
```