Event Observer Service

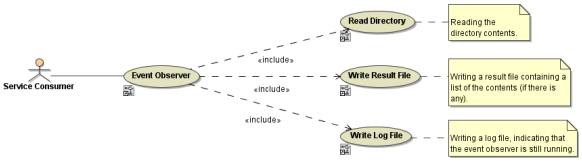
A class stereotyped <<E2EEventObserver>> models the Event Observer. The difference between Timer and Event Observer is how the corresponding event handlers are modeled. An activity assigned to the Timer class models the Timer event handler. For the Event Observer, the event handler activity is not assigned directly to the Event Observer class. Instead, the activity assigned to the Event Observer class calls explicitly the event handler, because UML 2.0 allows us to model time events in activities explicitly (see activity **Modeling the Behaviour of an Event Observer**).

On this Page:

- Event Observer Behavior
- Event Observer Components

The following use case diagram shows the use cases that illustrate the behavior of an Event Observer. The complete model is found in the example file.

Figure: Event Observer Use Cases



The next chapter provides an overview of the components required to build an Event Observer service.

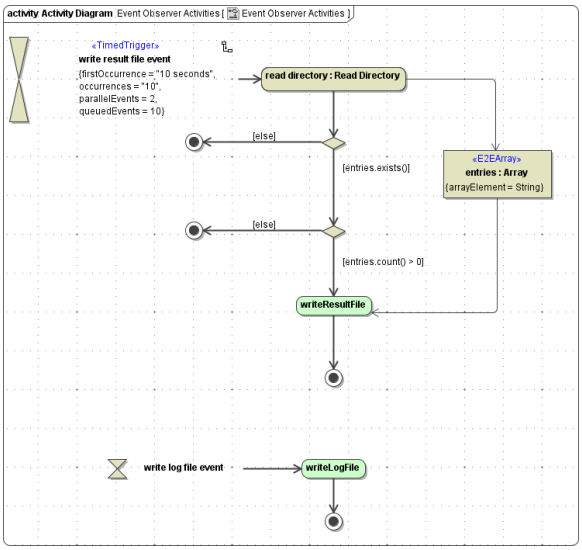
Example File (Builder project Add-ons/Scheduling):

<your example path>\Add-ons\Scheduling\uml\eventObserver.xml

Event Observer Behavior

The following activity (**Event Observer Activities**) was assigned to the **EventObserver** class. In UML, this implies that this activity diagram governs the behavior of this class.

Figure: Modeling the Behavior of an Event Observer

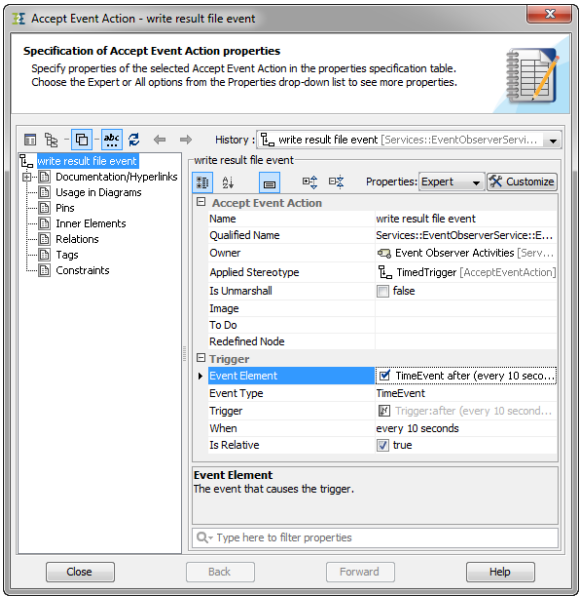


In contrast to UML 1.x, UML 2.0 offers to model time events in activities using time-triggered transitions . In order to draw such a transition, you first need to draw a time event symbol and then a transition to an action state or a sub-activity state starting from the time event symbol.

Figure: Transition from Time Event to Action Node

The hour glass symbolizes time events. To configure this symbol, open the time event specification dialog, select whether the time event is absolute or relative, and enter a time expression such as every 10 seconds.

Figure: Configuring Time Events for the Event Observer

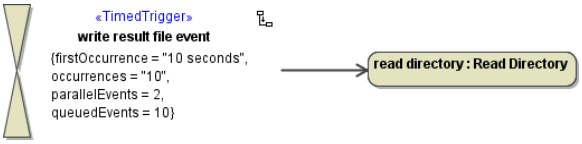


The term every 10 seconds defines that a time event is triggered every 10 seconds. every is a keyword. The expression every 10 seconds is equivalent to setting the tagged value **repeatInterval** to 10 seconds when using a Timer. The correct syntax of Event Observer time expressions is:

```
every <TimeDurationExpression>
```

For the syntax of `<TimeDurationExpression>` refer to Time Durations. In addition to a time duration expression, the time-triggered transition can be configured by using all tagged values defined for Timer instances (see Timer Deployment).The only exception is the tagged value **repeatInterval**, which is covered by the time expression. For example:

Figure: Example of Configuring a Time-Triggered Transition with Tagged Values



To reverse the position of the hour glass and the configuration information for a time event, select the time event on the activity, position the cursor over the time event, and click the Rotate State button. The button appears only when the cursor is positioned over the time event.

The activity **Modeling the Behavior of an Event Observer** models the behavior of an Event Observer class by defining the time events and the handlers of the time events. The activity has no initial state. Instead, the targets of both time-triggered transitions define the start points of the event handler. An event handler start point can be any activity, such as the sub-activity **Read Directory** or a call operation action calling an operation of the Event Observer class (for instance **writeLogFile**). In **Modeling the Behavior of an Event Observer**, one time event triggers a lookup in a directory. If this directory contains files, the activity calls the Event Observer operation **writeResultFile**. The implementation of this sub-activity operation call is not shown here. Refer to the example file.

Sometimes it is reasonable for one Event Observer to trigger more than one time event because the observed events are associated logically . This can be done by drawing another hour glass symbolizing another time event (see **Modeling the Behavior of an Event Observer**).

> The Bridge supports no synchronization mechanism between parallel running event handlers. By definition, the event handlers are independent. Therefore, if event handlers have common dependencies, such as shared resources, the modeler must consider this issue.

Running the xUML service, the service may write the following warning to the xUML service standard log:

```
[Warning][External][TIMADLM][13][Event dropped, queue is full. Request
not being executed: <internal activity diagram ID>
```

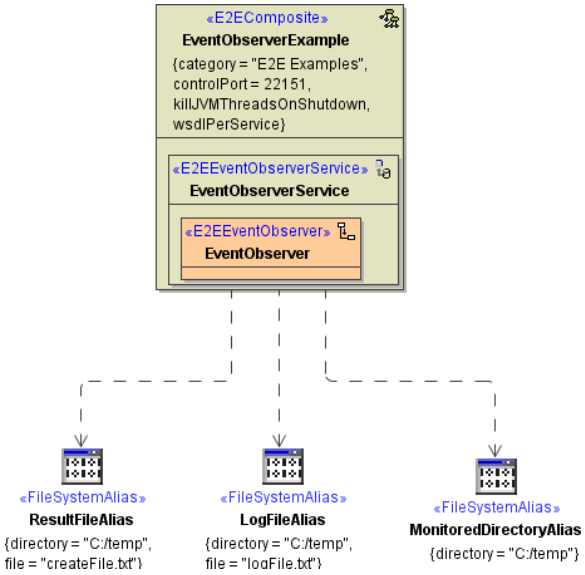
The Bridge provides a mechanism for viewing the xUML service standard log. Troubleshooting Timer and Event Observer Services describes the reason for, and solution to, this problem.

Event Observer Components

A service component stereotyped <<E2EEventObserverService>> models an Event Observer service (**EventObserverService**) . The class **EventObserver** (<<E2EEventObserver>>) represents the actual Event Observer. An activity assigned to the class contains the calls to the event handlers. The class can have operations that model the behavior of this class. They may contain the event handler.

Also, the component diagram contains the **FileSystemService** because, in this example, the Event Observer service uses the local file system.

Figure: Event Observer Component Diagram



The **EventObserverExample** composite contains an Event Observer service artifact(<<E2EEventObserverService>>). The Event Observer service contains the event observer class (**EventObserver**).

The backend service alias **FileSystemServiceAlias**, shown in the component diagram above, is required because the activities access the file system. For more details, refer to the example file.