QL Server 2005 - SQL Server Integration Services - Part 11 - SSIS Events and Event Handlers

One of the concepts introduced in the previous article of our series discussing SQL Server 2005 Integration Services were events. While we initially looked at them in the context of Windows Instrumentation Management (which, in turn, serve as the basis of operations for SSIS WMI Event Watcher Task), we also mentioned that equivalent functionality has been implemented as an independent feature within SSIS packages. SSIS events are generated by all of the executable components (including tasks, containers, and packages themselves) at various stages of their lifetime. If desired, it is possible to define a set of actions that will be triggered in response to any of them and incorporate such actions into package components known as event handlers. In this article, we will focus on the characteristics of SSIS events and their handlers, and provide some examples demonstrating their use.

SQL Server 2005 Integration Services provide the ability to handle practically any type of event associated with execution of its task and container (through the ability to configure corresponding handlers). The following list contains more significant and commonly monitored types of events (you might be able to spot some of them in the Output window during package execution in Debug mode):

- OnError - generated as the result of an error condition. It falls into the category of the most frequently implemented types of event handler. Its purpose can be, for example, to gracefully terminate failed execution of a task or container, to provide additional information simplifying troubleshooting, or to notify about a problem and need for remediation.
- OnWarning - similar to the OnError event, it is raised in response to a problem (although not as significant in terms of severity).
- OnInformation - produces reporting information relating to the outcome of either validation or execution of a task or container (other than warning or error).
- OnTaskFailed - signals the failure of a task and typically follows OnError event.
- OnPreExecute - indicates that an executable component is about to be launched.
- OnPreValidate - marks the beginning of the component validation stage, following the OnPreExecute event. The main purpose of validation is detection of potential problems that might prevent execution from completing successfully.
- OnPostValidate - occurs as soon as the validation process of the component is completed (following OnPreValidate event),
- OnPostExecute - takes place after an executable component finishes running.
- OnVariableValueChanged - allows you to detect changes to variables. The scope of the variable determines which executable will raise the event. In addition, in order for the event to take place, the variable's ChangeEvent property must be set to True (the default is False).
- OnProgress - raised at the point where measurable progress is made by the executable (for example, when running Execute SQL Task). This can be evaluated by monitoring the values of the system variables associated with the OnProgress event handler, such as ProgressComplete, ProgressCountLow, and Progress CountHigh.

Some events are limited to specific scenarios (for example, OnQueryCancel is appropriate in cases where it is feasible to cancel task execution during its

processing), others are component-specific (for example, WMIEventWatcher task, described in our previous article, provides the ability to configure handlers for OnWMIEventWatcherEventOccurred and OnWMIEventWatcherEventTimedout events, in addition to the ones listed above).

Most of the event categories (with the exception of the component-specific ones) exist for each executable (although there are exceptions), which means that within a package, you can have multiple event handlers, invoked by the same type of event. In addition, events raised by a subcomponent are by default processed not only by its own handler (if it exists), but also by a handler for the same event assigned to its parent and grandparent(s) - until the top container - i.e. package - is reached. This event processing "hierarchy" is beneficial, since it allows you to minimize the number of handlers that you need to create in order to respond properly to events that you are concerned about (in the simplest case, you can assign them only on the package level). On the other hand, you can prevent processing of events occurring on the subcomponent level by their parents' handlers through assigning a False value to the Propagate system variable (of Boolean data type) of a subcomponent's handler (by default, its value is set to True). This way, you can ensure that a particular event for a specific executable is handled in a unique fashion. Another possibility for modifying the default event processing behavior involves the use of the DisableEventHandlers property of an executable, which prevents any of its handlers from being triggered in case of an associated event - however, the notification about the event is passed up the hierarchy, which activates equivalent handlers defined in the parent and grandparent(s) containers.

Event handlers share a number of System Variables, which can be used to obtain more information about them and their underlying events (such as EventHandlerStartTime, LocaleID, SourceDescription, SourceID, and SourceName) or change their characteristics (such as earlier described Propagate). There are also variables which are event handler specific (for example, ErrorCode and ErrorDescription, associated with the OnError event handler or already mentioned variables implemented as part of the OnProgress event handler).

Event handlers resemble packages, since both share the same set of features, including control and data flow as well as support for variables for which they provide scope. While you can build and configure them programmatically, it is much more straightforward and intuitive to use the Event Handlers tab of the package designer interface within the SQL Server 2005 Business Intelligence Development Studio for this purpose. The tab contains two listboxes - one listing all package executables, the other displaying corresponding events for each - and a link placed underneath them, which creates an empty handler for the currently selected executable and event. Once you click on the link, you will be presented with the handler interface onto which you can add control flow elements (including the Data Flow task) by dragging them from the Toolbox tab (the same way you would populate the Control Flow of a package). With Event Handlers tab active, you also have access to the Connection Managers area, with its server and data sources.

Let's take a look at a simple example demonstrating some of the features described so far. In particular, we will experiment with the OnError event handlers defined on the task and package level and their interaction, depending on the value of the DisableEventHandlers task property and Propagate system variable of the task's event handler. We can trigger an error condition using the WMI Event Watcher-based package, presented in the previous article of this series (refer to it in order to determine steps required to create the package). It does not really matter which one of our sample WQL Queries you specify as the WQLQuerySource option (assuming that you use "Direct input" as the WQLQuerySourceType), since our goal

is not to satisfy the condition specified there, but rather trigger the task failure. This can be easily accomplished by setting the AfterTimeout option to "Return with failure," assigning a value of 1 to Timeout, and changing ActionAtTimeout to "Log the time-out and fire the SSIS event."

Now switch to the Event Handlers tab of the Designer interface. Choose the WMI Event Watcher Task in the Executable listbox and select OnError as its Event handler. Click on the link below to create the OnError event handler. Drag and drop the Script Task from the Toolbox onto the empty area of the designer (as you can see, this process is identical to creating a regular package). Right-click on the newly added component and activate the Edit option from its context sensitive menu. After switching to the Script option, type `SourceName` as the value of ReadOnlyVariables and `Propagate` as the value of ReadWriteVariables. These are system variables of the OnError event handler, which we will use to extract additional information about the way events are handled and to modify their default behavior. Click on the "Design Script..." button and enter the following as the content of the Public Sub Main():

```
Public Sub Main()
 '
 ' Add your code here
 '
 Dts.Variables("Propagate").Value = True
 MsgBox("Event Source = " & CType(Dts.Variables("SourceName").Value, String) &
vbLf & _
  "Propagate = " & CType(Dts.Variables("Propagate").Value, String),
MsgBoxStyle.OKOnly, _
  "Script Task in OnError Event Handler of WMI Event Watcher Task")
 Dts.TaskResult = Dts.Results.Success
End Sub
```

This code assigns a value of True to the Propagate system variable of the OnError event handler and displays the message box, with descriptive header, informing about the executable, which was the source of the error as well as the current value of the Propagate variable.

Close the Microsoft Visual Studio for Applications window and click on OK in the Script Task Editor interface to return to the Event Handlers tab of the designer interface. Click on the down arrow symbol of the Executable listbox and select the Package entry. Using the same procedure as before, create an event handler for the OnError event containing the Script Task. In its Editor dialog box, set ReadOnlyVariables to `SourceName` and type in the following code after displaying the Microsoft Visual Studio for Applications window by clicking on the "Design Script..." command button:

```
Public Sub Main()
 '
 ' Add your code here
 '
 MsgBox("Event Source = " & CType(Dts.Variables("SourceName").Value, String), _
  MsgBoxStyle.OKOnly, "Script Task in OnError Event Handler of Package")
 Dts.TaskResult = Dts.Results.Success
End Sub
```

As before, close the Microsoft Visual Studio for Applications and Script Task Editor windows to return to the Event Handlers tab of the designer interface. Switch to the

Control Flow tab and launch the execution of the package by pressing the F5 key (or selecting the Start Debugging item from the Debug menu). Shortly after (following 1 second of execution), the WMI task should time out and trigger failure. This, in turn, invokes the OnError event handler we have defined for the WMI Event Watcher Task, which generates the message box, listing the WMI Event Watcher Task as the Event Source and the current value of the Propagate system variable (True). Once you click on OK, you will be presented with another message box, generated by the OnError event handler on the package level (listing WMI Event Watcher Task as the Event Source). In addition, you will notice that both the task and the package failed. This happens because the value of the Propagate system variable is set to True. If you set it to False by changing the first line of code in the Sub Main procedure of the Script Task in OnError event handler for WMI Event Watcher Task to:

```
Dts.Variables("Propagate").Value = False
```

after stopping the execution, and re-launch the package again, you will see only the message box generated by the OnError event handler for the WMI Event Watcher Task (you can verify this by checking the message box header, which should contain "Script Task in OnError Event Handler of WMI Event Watcher Task" text). At the same time, even though our task fails, the package finishes successfully (this behavior can be altered by changing the values of the FailPackageOnFailure or FailParentOnFailure task properties from their default False to True).

Now revert to our original first line in the OnError event handler in the WMI Event Watcher Task (setting the Propagate system variable back to True). Select the WMI Event Watcher Task on the Control Flow area of the designer interface and set its DisableEventHandlers property to True (in the Properties window). Once you launch the package, you will notice that this time only the OnError event handler defined on the Package level is triggered (which, at the same time, results in the package failure).

Keep in mind that this is barely a short introduction into the rich event-related functionality in SQL Server 2005 Integration Services. As you can see, they function in essence as individual packages, giving you the ability to configure complex workflows, accommodating a variety of dynamically changing criteria.