SQL Server 2005 - SQL Server Integration Services - Part 10 - WMI Event Task

In the previous article of our series dedicated to new features in SQL Server 2005 Integrated Services, we started exploring methods of incorporating Windows Management Instrumentation (WMI) based data into the package flow control. As explained, this functionality has been implemented in the form of two components - WMI Data Task and WMI Event Task. The former (which we already have covered) provides access to data describing a managed environment, collected through WMI as soon as it is launched. This allows you, for example, to establish the status of a computer system at an arbitrarily chosen date and time, to ensure that parameters such as disk space, CPU utlization, or free memory are within limits required in order to continue package execution. Similarly, you can gather inventory of event log entries, installed applications, or running processes, if other tasks depend on them in some way. Note that in all these examples, you are the one who decides (through either interactive or scheduled execution) the point in time at which values of WMI object properties are extracted. In addition, use of WMI Data Task implies that you are primarily interested in actual values. However, there might be cases in which you are more concerned about detecting object creation, deletion, or modification (e.g. when an arbitrary threshold is reached) in order to trigger an appropriate action at that time. This is where WMI Event Task comes into play.

In order to explore the way WMI Event Task operates, we need first to explain how WMI handles this type of scenario. As you might recall from our discussion about principles of WMI, management features are implemented through classes. While the classes we have mentioned so far deal with the state of hardware and software components, there are also others which represent their dynamic character. Depending on the type of changes, known in WMI jargon as events (to be exact, changes to WMI classes that we are interested in are called intrinsic events), there are three distinct classes:

- __InstanceCreationEvent class, representing creation of a new object of a WMI class,
- __InstanceDeletionEvent class, representing deletion of an existing object of a WMI class,
- __InstanceModificationEvent class, representing modification to an existing object of a WMI class.

Note that these classes are designed with the intention of being paired up with another WMI class, providing the ability to monitor changes to its objects. Furthermore, WMI Query Language, which we briefly introduced before (and which we will be using to extract data about instances of event classes), has some syntactical peculiarities that you will need to be familiar with. More specifically, the WHERE clause (mandatory in this case) includes the ISA keyword followed by the name of a monitored class (which also applies the query to all of its subclasses - as we mentioned in the previous article, classes form a multi-level hierarchy, where children inherit properties and methods of their parents). The WITHIN keyword (in the context presented in our examples) sets the polling interval, which determines the maximum amount of time between the event taking place and its notification being delivered to the WMI-based application (WMI Event Task, in this case). You might want to set this interval to a small integer for the duration of testing, but remember to increase it afterwards, since low values tend to have negative impact on system performance. Another common characteristic of WMI Event queries is the use of the TargetInstance property (which exists in all three of the intrinsic event classes listed above) representing an object that was the subject of the monitored event (created, modified, or deleted). This provides the ability to further narrow down the scope of your queries by specifying criteria identifying exact instances that you are interested in. __InstanceModificationEvent class has, additionally, PreviousInstance property, representing the object with its properties prior to the event, which allows comparing "old" values against the "new" ones assigned to TargetInstance properties.

To make these specifications a bit easier to understand, let's look at some of their practical applications. Most commonly, we will be testing for creation of a particular object (class instance), which means that we will be running the WQL SELECT statement against the __InstanceCreationEvent class. For example, in order to detect newly launched processes (and ensure that the notification will be returned in no more than 10 seconds), you would execute the following query:

```
SELECT * FROM __InstanceCreationEvent WITHIN 10
 WHERE TargetInstance ISA "Win32_process"
```

If you are interested only in a specific process (e.g. iexplore.exe), you can add to the WHERE clause check for matching value of the Name property of the object represented by the TargetInstance property:

```
SELECT * FROM __InstanceCreationEvent WITHIN 10
 WHERE TargetInstance ISA "Win32_process"
 AND TargetInstance.Name = "iexplore.exe"
```

Similarly, you can monitor Windows Event Viewer for a particular entry. TargetInstance properties that might be helpful in assigning desired selection criteria - besides the Logfile (which is included in our example below and can take on values of System, Application, or Security) - include SourceName (identifying the component responsible for generating the entry), or EventCode (a unique number assigned to every event type):

```
SELECT * FROM __InstanceCreationEvent WITHIN 10
 WHERE TargetInstance ISA "Win32_NTLogEvent"
 AND TargetInstance.Logfile = "System"
```

As mentioned before, __InstanceModificationEvent queries, frequently take advantage of the PreviousInstance property. The sample query listed below monitors the insertion of a disk into the CD-ROM drive (by checking values of MediaLoaded property of Win32_CDROMDrive class instance before and after the modification WMI event):

```
SELECT * FROM __InstanceModificationEvent WITHIN 10
 WHERE TargetInstance ISA "Win32_CDROMDrive"
 AND TargetInstance.MediaLoaded = TRUE
 AND PreviousInstance.MediaLoaded = FALSE
```

Probably the most common data processing scenario in which WMI events could be helpful, involve launching a sequence of tasks activities as soon as data to be processed becomes available (e.g. by being copied to target computer). This can be accomplished by employing a SELECT query against __InstanceCreationEvent class in combination with another class, for which creation of a new instance is equivalent to creation of a new file on a managed computer. The best match in this case (from the performance point of view) is the CIM_DirectoryContainsFile class (this class forms an association between CIM_Directory and CIM_File classes, which are referenced as its GroupComponent and PartComponent properties, respectively). In order to detect a new file appearing in a specific directory (in our example c:\Data), you would run the following query (note that the last entry contains extra backslashes as so-called escape characters, which ensure proper string parsing. Such characters need to be applied whenever you want the next character to be treated literally. For example, by placing it after `"Win32_Directory.Name=\`, you ensure that the following double quote does not indicate the end of this string, but rather the beginning of the path to target folder c:\Data. Similarly, we added it in front of each backslash appearing in the folder path):

```
SELECT * FROM __InstanceCreationEvent WITHIN 10
 WHERE TargetInstance ISA "CIM_DirectoryContainsFile"
 AND TargetInstance.GroupComponent = "Win32_Directory.Name=\"c:\\\\Data\""
```

Now it is time to look into incorporating this functionality into SSIS packages using the WMI Event Watcher Task. Start by creating a new project using SQL Server Business Intelligence Development Studio. Drag the WMI Event Watcher Task from the toolbox onto the Control Flow area of the package designer interface. Right-click on it and select Edit option from the context sensitive menu. In the resulting WMI Event Watcher Task Editor window, switch to the WMI Options entry. This will allow you to set the following parameters:

- WmiConnection - this option contains the name of the target system and WMI namespace (which, by default is, root\cimv2), as well as the authentication method applied when connecting to it (for more detailed information, refer to the previous article, where we covered this topic in more detail).
- WqlQuerySourceType - as in the case of the WMI Data Task, this option can be assigned Direct input, File connection, or Variable.
- WqlQuerySource - the way this entry is populated depends on the WqlQuerySourceType. If you decide to test our examples above, ensure that you remove all extraneous formatting characters (tabs, carriage returns), which we included in order to make them more readable.
- ActionAtEvent - two available choices are "Log the event" or "Log the event and fire SSIS event." We will be covering SSIS events and package execution logging capabilities in our future articles, but for now, in order to be able to test our examples, let's review briefly the way to verify their successful completion through entries recorded in logs. Start by right-clicking on the unused space of the Control Flow area of the designer interface and selecting the Logging... item from the context sensitive menu. You will be presented with the Configure SSIS Logs:Package window. Click on the Package label in the list of Containers on the left hand side of the window. On the Providers and Logs tab on the right hand side, ensure that "SSIS log provider for Text Files" is selected and use the Add... button to make the corresponding entry to appear in the "Select the logs to use for the container" section. In the list of Containers on the left hand side, mark the checkbox next to the WMI Event Watcher Task and click on its label (leaving the Package checkbox empty). You should see the "SSIS log provider for Text Files" under the Providers and Logs tab. Click on the list box in the Configuration column and select the <New connection...> entry. This will display the File Connection Manager Editor dialog box, from which you need to specify the Usage type (Create file, Existing file, Create folder, or Existing folder) as well as the file name and location. All logging actions resulting from events associated with WMI Event Watcher Task will be recorded in this file. Once you have completed configuring connection manager, confirm your choices with the OK button. After you return to the Configure SSIS Logs:Package window, switch to the Details tab. Here you

can choose events that you want to be logged. For the purpose of our exercise, enable the WMIEventWatcherWatchingForWMIEvents entry. Click on OK to return to the SSIS designer interface. From this point on, designated events will be recorded in the log file you specified during each package execution. You can review it to verify the outcome.

- ActionAtTimeout - determines an action that is triggered if a monitored WMI event does not take place before the period specified in the Timeout option (described later) passes. This includes logging the time-out or logging the time-out and firing an SSIS event (as we already mentioned, we will be reviewing SSIS events in our next article). If you intend to test the timeout functionality, for now, use the "Log the time-out" entry and mark the WMIEventWatcherEventTimedout entry on the Details tab of the Configure SSIS Logs:Package dialog box.
- AfterTimeout - allows the task to either Return with failure, Return with success, or Watch for the event again. Your choice will likely depend on the other components within the Control Flow - in the case of our simple examples, this setting is not relevant.
- NumberOfEvents - is an integer indicating the number of events that the task will capture before completing.
- Timeout - specifies the number of seconds before the AfterTimeout action is launched (assuming that number of monitored WMI events has not reached NumberOfEvents value before this time has elapsed).

To test the examples listed above, configure logging, choose the Direct input as the WqlQuerySource, and copy the WQL queries to the WqlQuerySource window (you might want to modify the WITHIN value to 1, to see the results immediately). Use the default Timeout value of 0, which keeps it disabled. Launch the package and pay attention to changes in color of the WMI Event Watcher Task, which should initially turn yellow. You can force each of the WMI events by launching a new instance of Internet Explorer, generating an event in the Windows System Event Log (e.g. by stopping or starting a non-critical service), putting a disk in the CD-ROM drive, or creating a new file in c:\Data directory. Shortly afterwards, you should see the color of the WMI Event Watcher task becoming green. In addition, you can verify successful task completion by checking content of the log file.