Bonus Chapter A

Monitoring SSIS Performance with Custom Performance Counters

A number of useful *performance counters* ship with SQL Server Integration Services. You can use them for tracking some of the internal metrics of the SSIS Data Flow Task, such as data buffers spooling to disk. You can also use these counters along with Windows and SQL Server counters for disk or table read and write. When you do so, they can be powerful tools.

This chapter assumes that you are familiar with the use of the Windows System Monitor tool for performance. If you are, then you will probably already be using SSIS counters with your packages.

Sometimes, however, you may wish to measure specific aspects of your packages—aspects that standard counters do not cover. Fortunately, the .NET Framework includes classes to create your own counters. As you have seen in earlier chapters, these classes are easy to use with SSIS script tasks and components. As a result, it is also quite simple to create your own performance tools, directly within SSIS.

This chapter includes two simple examples. The first example is how to create and manage performance counters using a script task in the control flow. The second example is how to write to those counters using a script component in the data flow.

Creating and Managing Customer Performance Counters

Windows organizes counters in *performance counter categories*. If you are creating your own counters, you will most often also wish to create a new category for them. In addition to a category, each counter also has a *performance counter type*.

Performance Counter Types

The *type* of a counter determines important aspects of its behavior. The following list describes each type, along with some typical uses:

- **NumberOfItems32** — Use this counter simply to count items, storing the count as a 32-bit number. For example, you may wish to count the number of times some conditional code in a script executes. This can be useful to profile the execution of the script during the SSIS process. This is the default counter type.
- **NumberOfItems64** — This is a higher capacity version of the previous counter.
- **RateOfCountsPerSecond32** — Use this counter to track the rate at which an operation occurs. For example, you might wish to track the number of rows per second passing along a specific path in SSIS. In such a case, a script component incrementing a counter of this type could be useful.
- **RateOfCountsPerSecond64** — This is a higher capacity version for counting the rate at which an operation occurs.
- **AverageTimer32** — Use this counter to calculate the average time to perform a process.

Creating Performance Counters and Categories

Windows defines some categories for counters, but users cannot add custom counters to them. Therefore, you must create custom categories for your own use. However, even with custom categories, there are some important restrictions.

First, you cannot create counters and categories remotely, and to do so locally you must be an administrator on the computer. This restriction may not be surprising, but you must bear it in mind if creating counters or categories using a script task in SSIS. The SSIS package must run with administrator privileges in order to create the counter. In general, it is not good practice to run SSIS packages with full administrator privileges. However, performance counters and categories do persist until you delete them, so it is possible to create the categories and counters that you need in a one-off package that the administrator can execute. After running the administrator's package, the counters are ready for other packages to use.

Second, you cannot add new counters to existing categories, including existing customer categories.

If you need only one counter, you should create the counter object and pass it as a parameter when creating a category. However, in many cases you will wish to create several counters. In such cases, follow these steps to create counters:

1. Check that the category does not already exist. If it does, Windows will raise an exception when trying to create it again.
2. Create a **CounterCreationDataCollection** object for your counters.
3. Create your counters as **CounterCreationData** objects.
4. Set properties for your counters.
5. Add your counters to the collection.
6. Create the category using the **PerformanceCounterCategory** class. Pass the counter collection as a parameter.

This is quite simple to do in an SSIS script, as you shall see.

Creating Performance Counters in SSIS

Creating and managing counters for SSIS will nearly always be a control flow operation. In fact, I cannot readily think of cases where you would wish to create a counter in the data flow. Having said that, I am sure someone will e-mail with an example!

These examples explore a simple scenario. A data integration process includes some complex handling of exception rows for data quality. The process detects exception rows based on some business rules, perhaps using a conditional split. The exception rows flow down a specific path in the data flow. After detection, the data flow passes these rows through several transformations in an effort to fix them up. However, the exception handling may be quite complex. It may even slow down the entire process, if it finds too many exceptions. The administrator, then, has decided to set up two counters to monitor quality exceptions in real time. One measures the number of exception rows. The other measures the number of exception rows per second.

Listing A.1 shows how to create a category with counters in an SSIS script task. It follows the six steps described above.

```
Imports System
Imports System.Data
Imports System.Math
Imports Microsoft.SqlServer.Dts.Runtime
Imports System.Diagnostics

Public Class ScriptMain
   Public Sub Main()
      'Check that the counter category does not already exist
      If Not PerformanceCounterCategory.Exists("DQExceptionCounters") Then
         'Create a CounterCreationDataCollection object
         Dim CounterCollection As New CounterCreationDataCollection()

         'Create your counters as CounterCreationData objects
         Dim MyCounter1 As New CounterCreationData()
         Dim MyCounter2 As New CounterCreationData()

         'Set properties for your counters
         MyCounter1.CounterName = "DQ Exception Rows"
         MyCounter1.CounterHelp = "This counter tracks the number of data
quality exception rows in SSIS"
         MyCounter1.CounterType = PerformanceCounterType.NumberOfItems32
         MyCounter2.CounterName = "DQ Exception Rows/Sec"
         MyCounter2.CounterHelp = "This counter tracks the number of data
quality exceptions per second in SSIS"
         MyCounter2.CounterType = PerformanceCounterType.
RateOfCountsPerSecond32
```

```
         'Add your counters to the collection.
         CounterCollection.Add(MyCounter1)
         CounterCollection.Add(MyCounter2)

         'Create the category, passing the counter collection to it.
         PerformanceCounterCategory.Create("DQExceptionCounters", "Counters
for monitoring data quality in SSIS", CounterCollection)
         Dts.TaskResult = Dts.Results.Success
      Else
         Dts.Events.FireInformation(1, "", "Perf counter category exists", "",
0, True)
         Dts.TaskResult = Dts.Results.Failure
      End If
   End Sub
End Class
```

Listing A.1: Creating Performance Counters in an SSIS Script Task.

Note that the **System.Diagnostics** namespace is imported. As you can see, the main script first checks to see if category **DQExceptionCounters** exists. If this category does not exist, then the script continues, creating a counter collection. The script adds two counters, with a name, help string, and counter type for each. Next, we add the counters to the collection. Finally, we create the category, passing in the counter collection as a parameter. We also pass in a name and help string for the category. Note that if the category does exist, we use an event to return a useful message about the failure.

After creating this script task, execute the package. You can now run the Windows System Monitor performance tool. In this tool, you will see that you have created the new counters successfully. Figure A.1 shows these new counters in the System Monitor tool viewer.

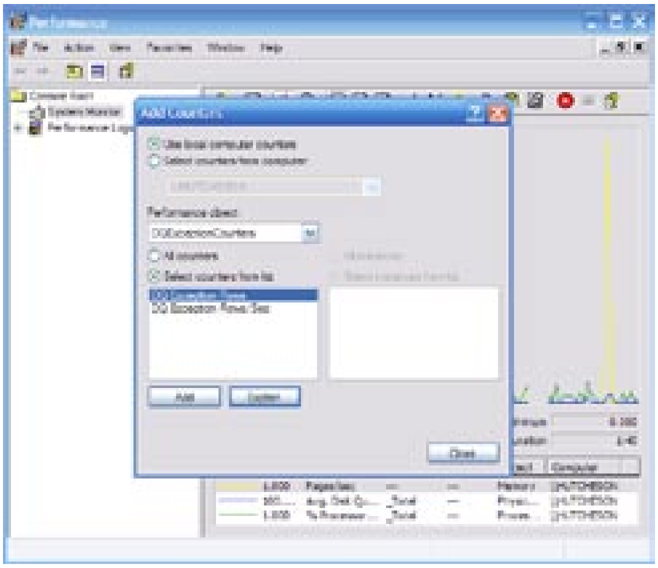

Figure A.1: New Performance Counters in the System Monitor Tool.

Deleting a Performance Counter in SSIS

At this point, for completeness, it will be useful to show how to delete a counter category in script. It is very easy. Listing A.2 shows how to delete a category and all the counters in it.

```
If PerformanceCounterCategory.Exists("DQExceptionCounters") Then
   PerformanceCounterCategory.Delete("DQExceptionCounters")

End If
```

Listing A.2: Deleting a Counter Category.

Writing to Performance Counters

Having created your counters, you will now want to write to them. It is certainly possible to write to a counter with a script task, but this will be a rare scenario. Users do not often need to monitor the performance of the SSIS Control Flow. However, you will find many uses for custom counters in the data flow. Therefore, in these examples you will learn how to write to a custom counter using an SSIS script component.

By now, you should be familiar with how to create an SSIS data flow, and how to add a script component to it. A script component to increment a counter can be rather unusual—it does not need to use any input columns. If your counter is handling row counts, you can create a simple synchronous component with no input columns. If your counter is to handle values within columns, then you will need to use input columns—just as for any other synchronous component. Script components that use performance counters will only very rarely be asynchronous.

You can use several methods to update counters. The ones you will find most useful are:

- **Decrement** — This method decrements the counter value by one. Use this for descending counters.
- **Increment** — This method increments the counter value by one. This is the most common method, used for most row-based counters.
- **IncrementBy** — Increments the counter value by a specified amount. You would typically use this to add the value of a column to a counter. For example, this method is useful when counting the number of line items passing through an ordering system.

Listing A.3 shows how to update the performance counters that you created using the **Increment** method.

```
Imports System
Imports System.Data
Imports System.Math
Imports Microsoft.SqlServer.Dts.Pipeline.Wrapper
Imports Microsoft.SqlServer.Dts.Runtime.Wrapper
Imports System.Diagnostics
Public Class ScriptMain
   Inherits UserComponent

   Dim myCounter1 As New PerformanceCounter("DQExceptionCounters", _
        "DQ Exception Rows", False)
   Dim myCounter2 As New PerformanceCounter("DQExceptionCounters", _
         "DQ Exception Rows/Sec", False)
   Public Overrides Sub Input0_ProcessInputRow(ByVal Row As Input0Buffer)
      myCounter1.Increment()
      myCounter2.Increment()
    End Sub
End Class
```

Listing A.3: Updating a Performance Counter Using a Script Component.

As you can see, this is a very simple script indeed. Remember to import the **System.Diagnostics** namespace for ease of use. Next, create two objects to access the counters. Note that you create these objects in the **ScriptMain** class. Be sure not to create these for each row. Otherwise, your performance counters will ruin your performance! For each row, simply call the **Increment** method for each counter. Figure A.2 shows these counters in use in the System Monitor tool.

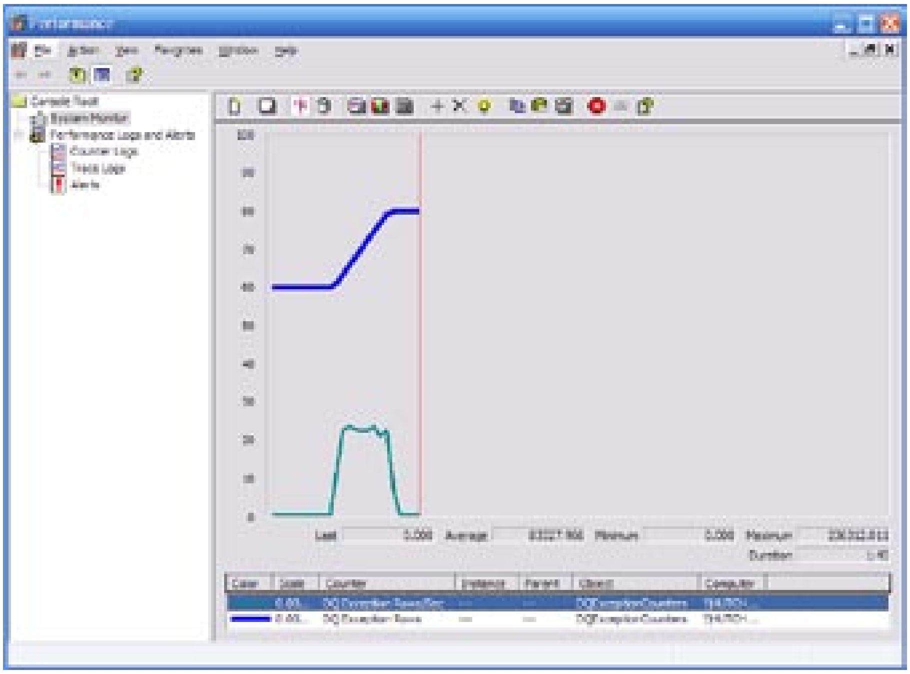

Figure A.2: New Performance Counters in Use.

The Performance of Performance Counters

As I hinted earlier, writing to your performance counters in the wrong way can have an impact on performance itself. Of course, even writing to a counter the correct way is not free. How expensive is it? On my laptop, I ran the example script with and without counters. The counters added only 0.8 seconds per million rows. For my needs, this is quite acceptable. For your scenarios, you will need to test on your own equipment. However, you are unlikely to find that the counters cause any serious issues.

Tech Tip:

This has been a brief introduction to performance counters. Read the .NET Framework documentation and MSDN for more features. For example, you can create multiple instances of the same counter to track several processes separately.

Summary

Custom performance counters give you very flexible monitoring of SSIS processes. Don't overdo them, but they can be extremely valuable when used with discretion. Familiarity with performance counters should be part of your everyday toolkit as an SSIS developer.