Managing Systems Data

ESTIMATED TIME
40 min.

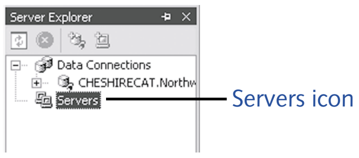
In this additional lesson, you will learn how to:

- ✔ *Read and write Microsoft Windows event logs.*
- ✔ *Monitor Windows performance counters and sample performance counter values over a period of time.*
- ✔ *Determine the status of a Windows service.*
- ✔ *Start and stop Windows services.*

The chapters in Part 5 of this book have shown you how to access data held in a database by using Microsoft ADO.NET and how to process XML data by using Microsoft's implementation of the XML Domain Object Model. In this lesson, you'll learn how to access and manage Windows systems data. Specifically, you will see how to read and write Windows event logs, monitor Windows performance counters, and start and stop Windows services.

Using Server Explorer

You have already used Server Explorer in databases through the Data Connections node. You may have noticed a Servers node as well. You can use this node to gain access to Windows systems data and services on your computer or on other computers if you have permission.



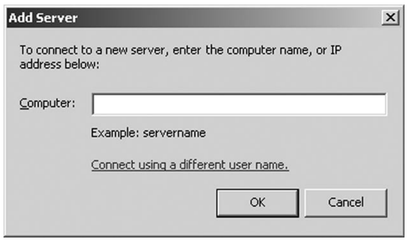
Connecting to a Server

Before you can use a server, you must connect to it. In the following exercises, you will create a connection to your own computer and examine the systems information that is available for your applications.

Create a connection

1. Start Microsoft Visual Studios.NET, and display Server Explorer (on the View menu, click Server Explorer).
2. In Server Explorer, right-click the Servers node. On the menu, click Add Server.

   The Add Server dialog box opens.

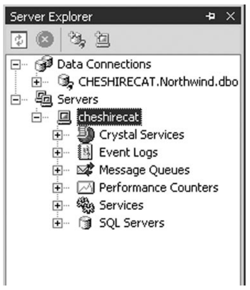
   
   Add Server
   To connect to a new server, enter the computer name, or IP address below:
   Computer:
   Example: servername
   Connect using a different user name.
   OK
   Cancel

3. In the Computer text box, type the short name of your computer, and then click OK.

tip

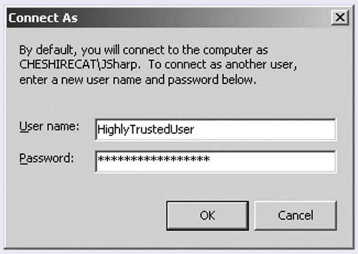
To find out the name of your computer, right-click the My Computer icon on the Start menu (or on your desktop if you are using Windows 2000) and then click Properties. In the System Properties dialog box, click the Network Identification tab. You will see the name of your computer on this page.

4. Expand the Servers node. Your computer is added to the list of servers in Server Explorer.

Server Security

To read and write systems information in Windows, you must have the appropriate security rights—some of the information you can gain access to might be sensitive. If you do not have sufficient privileges, you will need to ask your administrator for the name and password of an account you can use. You can then log on as this user by clicking Connect Using A Different User Name in the Add Server dialog box. You will be prompted for the user name and password.


Connect As

By default, you will connect to the computer as CHESHIRECAT\JSharp. To connect as another user, enter a new user name and password below.

User name: HighlyTrustedUser
Password: ******************

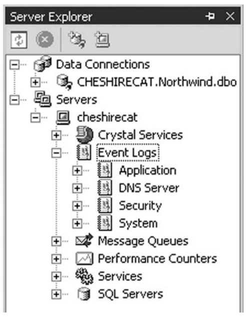
Type in the user name and password, and click OK to connect to the server.

Examine the server

The nodes and data available to your server will vary, depending on the software that is installed on your computer and how the software is configured. In this lesson, you will use the Event Logs, Performance Counters, and Services nodes. These nodes will always be available, and you will examine them now.

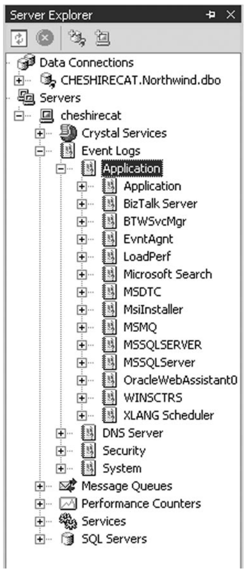
1 In Server Explorer, expand the node named after your server and then expand the Event Logs node.

The different types of event logs available on your computer appear, including application, security, and system logs. If your computer is a domain controller or if you have installed additional services, there might be other types of event logs included in this list as well.



2 In Server Explorer, under Event Logs, expand Application.

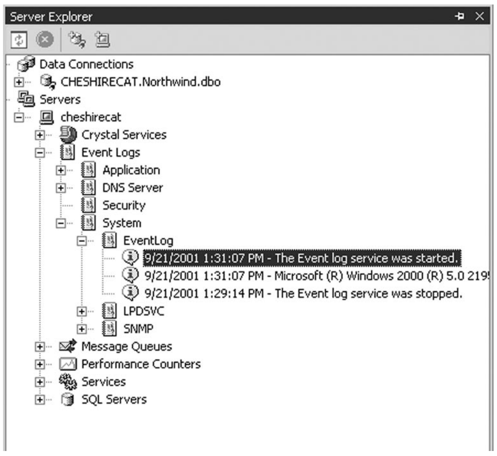
A list of all the event sources that have written to the log is displayed. The event logs are organized according to the source of the events (your list might be different from that shown in the following image).



```
Server Explorer
Data Connections
  CHESHIRECAT.Northwind.dbo
Servers
  cheshirecat
    Crystal Services
    Event Logs
      Application
        Application
        BizTalk Server
        BTWSvcMgr
        EvntAgnt
        LoadPerf
        Microsoft Search
        MSDTC
        MsiInstaller
        MSMQ
        MSSQLSERVER
        MSSQLServer
        OracleWebAssistant0
        WINSCTRS
        XLANG Scheduler
      DNS Server
      Security
      System
    Message Queues
    Performance Counters
    Services
    SQL Servers
```

3 In Server Explorer, under Event Logs, collapse the Application log and expand the System log. Expand the EventLog entry.

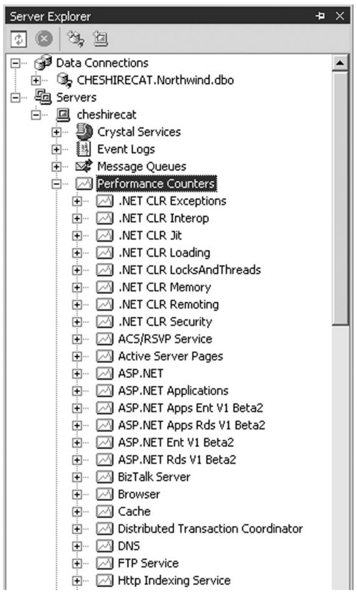
Information related to the EventLog service appears.



```
Server Explorer
Data Connections
  CHESHIRECAT.Northwind.dbo
Servers
  cheshirecat
    Crystal Services
    Event Logs
      Application
      DNS Server
      Security
      System
        EventLog
          9/21/2001 1:31:07 PM - The Event log service was started.
          9/21/2001 1:31:07 PM - Microsoft (R) Windows 2000 (R) 5.0 219
          9/21/2001 1:29:14 PM - The Event log service was stopped.
        LPDSVC
        SNMP
    Message Queues
    Performance Counters
    Services
    SQL Servers
```

4 In Server Explorer, collapse the Event Logs node, and then expand the Performance Counters node for your computer.

A list of performance objects is displayed (again, your list might vary).



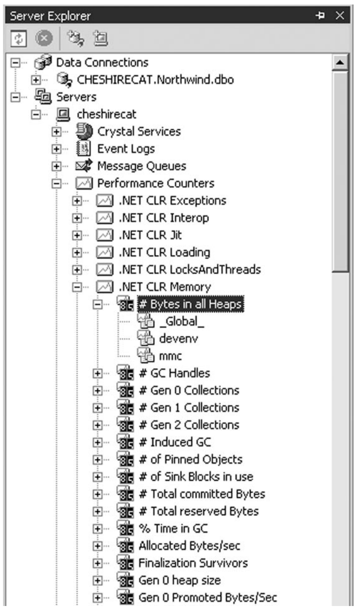
Server Explorer

Data Connections
CHESHIRECAT.Northwind.dbo
Servers
cheshirecat
Crystal Services
Event Logs
Message Queues
Performance Counters
.NET CLR Exceptions
.NET CLR Interop
.NET CLR Jit
.NET CLR Loading
.NET CLR LocksAndThreads
.NET CLR Memory
.NET CLR Remoting
.NET CLR Security
ACS/RSVP Service
Active Server Pages
ASP.NET
ASP.NET Applications
ASP.NET Apps Ent V1 Beta2
ASP.NET Apps Rds V1 Beta2
ASP.NET Ent V1 Beta2
ASP.NET Rds V1 Beta2
BizTalk Server
Browser
Cache
Distributed Transaction Coordinator
DNS
FTP Service
Http Indexing Service

5 In Server Explorer, under Performance Counters, expand .NET CLR Memory.

This object contains counters for monitoring the heap memory managed by the common language runtime.

6 In Server Explorer, expand # Bytes In All Heaps to view the instances available.

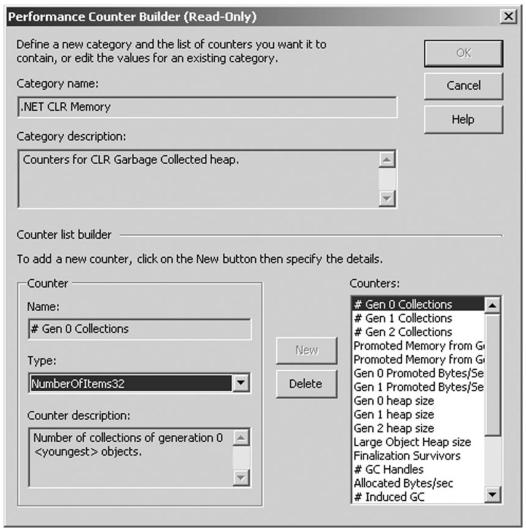


Server Explorer

Data Connections
CHESHIRECAT.Northwind.dbo
Servers
cheshirecat
Crystal Services
Event Logs
Message Queues
Performance Counters
.NET CLR Exceptions
.NET CLR Interop
.NET CLR Jit
.NET CLR Loading
.NET CLR LocksAndThreads
.NET CLR Memory
Bytes in all Heaps
Global
devenv
mmc
GC Handles
Gen 0 Collections
Gen 1 Collections
Gen 2 Collections
Induced GC
# of Pinned Objects
# of Sink Blocks in use
Total committed Bytes
Total reserved Bytes
% Time in GC
Allocated Bytes/sec
Finalization Survivors
Gen 0 heap size
Gen 0 Promoted Bytes/Sec

7 In Server Explorer, right-click .NET CLR Memory, and then click View Category on the pop-up menu.

The Performance Counter Builder dialog box opens and displays the counters for the object.



Define a new category and the list of counters you want it to contain, or edit the values for an existing category.

Category name: .NET CLR Memory

Category description: Counters for CLR Garbage Collected heap.

Counter list builder

To add a new counter, click on the New button then specify the details.

Counter

Name: # Gen 0 Collections

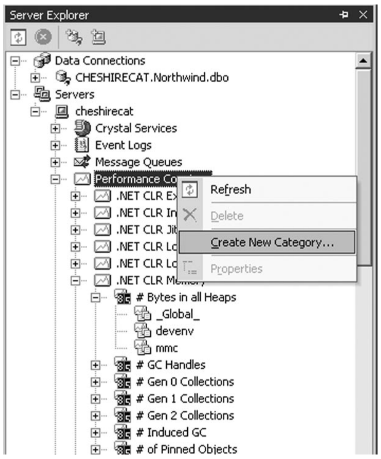
Type: NumberOfItems32

Counter description: Number of collections of generation 0 <youngest> objects.

Counters:
Gen 0 Collections
Gen 1 Collections
Gen 2 Collections
Promoted Memory from G
Promoted Memory from G
Gen 0 Promoted Bytes/Se
Gen 1 Promoted Bytes/Se
Gen 0 heap size
Gen 1 heap size
Gen 2 heap size
Large Object Heap size
Finalization Survivors
GC Handles
Allocated Bytes/sec
Induced GC

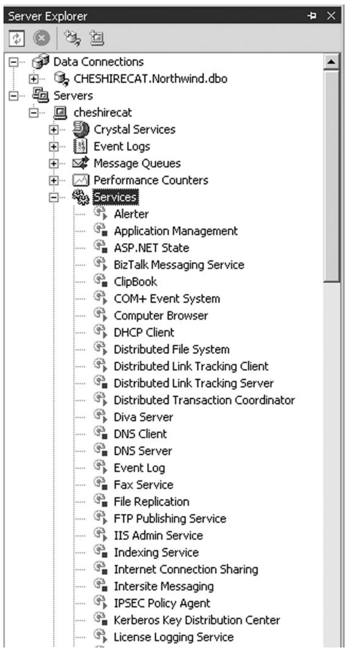
Under some circumstances, you can use this dialog box for adding new categories and counters. For now, the dialog box is in read-only mode—you cannot change any of the information shown. Click Cancel to close the Performance Counter Builder dialog box.

8 In Server Explorer, right-click Performance Counters, and then click Create New Category on the pop-up menu. The Performance Counter Builder dialog box appears again—this time, you can create your own custom category if you want. Close the Performance Counter Builder dialog box.



9 In Server Explorer, collapse Performance Counters and expand the Services node.

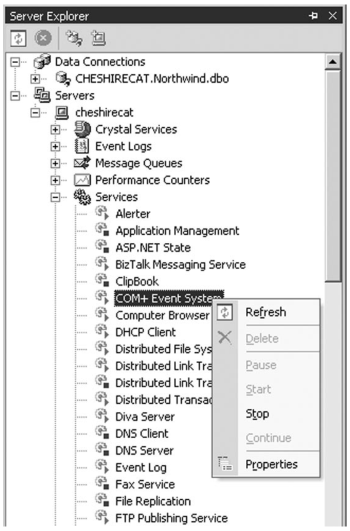
A list of the services installed on your computer is displayed.



```
Server Explorer
Data Connections
    CHESHIRECAT.Northwind.dbo
Servers
    cheshirecat
        Crystal Services
        Event Logs
        Message Queues
        Performance Counters
        Services
            Alerter
            Application Management
            ASP.NET State
            BizTalk Messaging Service
            ClipBook
            COM+ Event System
            Computer Browser
            DHCP Client
            Distributed File System
            Distributed Link Tracking Client
            Distributed Link Tracking Server
            Distributed Transaction Coordinator
            Diva Server
            DNS Client
            DNS Server
            Event Log
            Fax Service
            File Replication
            FTP Publishing Service
            IIS Admin Service
            Indexing Service
            Internet Connection Sharing
            Intersite Messaging
            IPSEC Policy Agent
            Kerberos Key Distribution Center
            License Logging Service
```

10 In Server Explorer, right-click COM+ Event System.

The pop-up menu allows you to start, stop, pause, and continue the service (assuming you have sufficient rights). The same menu is available to the other services. Do not click any of the commands.



```
Server Explorer
Data Connections
    CHESHIRECAT.Northwind.dbo
Servers
    cheshirecat
        Crystal Services
        Event Logs
        Message Queues
        Performance Counters
        Services
            Alerter
            Application Management
            ASP.NET State
            BizTalk Messaging Service
            ClipBook
            COM+ Event System
            Computer Browser
            DHCP Client
            Distributed File Sys
            Distributed Link Tra
            Distributed Link Tra
            Distributed Transac
            Diva Server
            DNS Client
            DNS Server
            Event Log
            Fax Service
            File Replication
            FTP Publishing Service
```

Accessing Systems Data

You can access each of the different nodes (Event Logs, Performance Counters, and Services) in your own programs. The Microsoft .NET Framework exposes classes in the *System.Diagnostics* namespace for handling event logs and performance counters. The *System.ServiceProcess* namespace contains classes for manipulating Windows services.

Using the Windows Event Log

In the following exercises, you will learn how to write to the Windows event log and also monitor the event log for changes made by other applications.

Write to the application event log

1. In Microsoft Visual Studio .NET, create a new project called LogWriter in the \Additional Lessons\Managing Systems Data by using the Windows Application template.

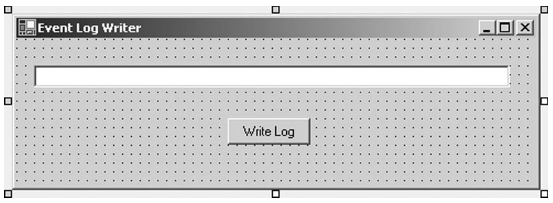
   A blank form called Form1 appears in the Design View window.

2. In the Toolbox, drag a *TextBox* control and a *Button* control onto Form1. (If the Toolbox isn't visible, click Toolbox on the View menu.)

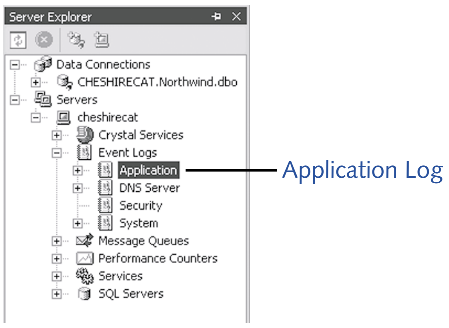
3. In the Properties window, set the properties of the form and controls to the values in the following table.

Control	Property	Value
Form1	*(Name)*	EventLogForm
	Size	480, 160
	Text	Event Log Writer
textBox1	*(Name)*	logEntry
	Location	16, 24
	Size	432, 20
	Text	*Leave blank*
button1	*(Name)*	writeLog
	Location	192, 72
	Text	Write Log

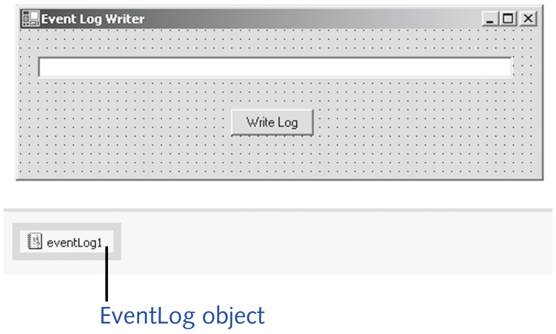
The form should look like the following image.



4 Display Server Explorer. In Server Explorer, make sure that the Servers node and the computer name node are expanded. Expand Event Logs, and then click and drag the Application node anywhere onto the form.



An *EventLog* object called *eventLog1* is added under the form.



5 Select the *writeLog* button. In the Properties window, click Events. Create an event handler for the *Click* event called *writeLogClick*.

A method called *writeLogClick* appears in the Code and Text Editor window.

6 In the Code and Text Editor window, add the following *using* statement to the top of the Form1.cs file:

```
using System.Diagnostics;
```

The *System.Diagnostics* namespace contains classes for handling event logs.

7 Locate the *Main* method. Replace the *Application.Run* statement that you see with the following statement:

```
Application.Run(new EventLogForm());
```

8 Locate the *writeLogClick* method that you created earlier. Type the following statements in the *writeLogClick* method:

```
eventLog1.Source = "LogWriter";
eventLog1.WriteEntry(logEntry.Text,
EventLogEntryType.Information);
```

The first statement sets the name of the event log to write to. If the log doesn't exist, it will be created when the application writes to the event log. The second statement actually writes to the event log. You can specify several different types of log entry according to the severity and type of the action creating the log entry—Error, FailureAudit, Information, SuccessAudit, and Warning. This example creates an Information entry.

9 Compile and run the program. Type **Recorded by Event Log Writer application** in the text box, and click the Write Log button.

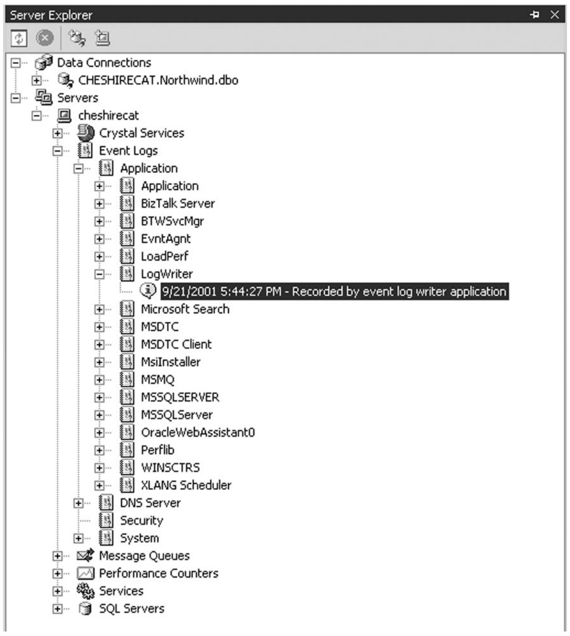
10 Close the program, and return to the Visual Studio .NET programming environment.

11 In Server Explorer, make sure that the Servers node and the computer name node are expanded. Expand Event Logs, and then expand Application. Right-click Application, and then click Refresh.

The *LogWriter* event source appears in the list.

12 Expand LogWriter.

The entry that you created in step 9 appears in the list.



```
Data Connections
    CHESHIRECAT.Northwind.dbo
Servers
    cheshirecat
        Crystal Services
        Event Logs
            Application
                Application
                BizTalk Server
                BTWSvcMgr
                EvntAgnt
                LoadPerf
                LogWriter
                    9/21/2001 5:44:27 PM - Recorded by event log writer application
                Microsoft Search
                MSDTC
                MSDTC Client
                MsiInstaller
                MSMQ
                MSSQLSERVER
                MSSQLServer
                OracleWebAssistant0
                Perflib
                WINSCTRS
                XLANG Scheduler
            DNS Server
            Security
            System
        Message Queues
        Performance Counters
        Services
        SQL Servers
```

Monitor the application event log

The *EventLog* class publishes an event called *EntryWritten*. You can use this event to monitor an event log and receive an alert when an application writes an entry to it.

1. In Visual Studios.NET, create a new project called LogReader and save it in the \Additional Lessons\Managing Systems Data folder by using the Windows Application template.
2. In the Toolbox, drag a *TextBox* control onto the form.
3. In the Properties window, set the properties of the form and the text box to the values specified in the following table.

Control	Property	Value
Form1	*(Name)*	EventLogForm
	Size	480, 96
	Text	Event Log Monitor
textBox1	*(Name)*	logEntry
	Location	16, 24
	Size	432, 20
	Text	*Leave blank*

4 Switch to Code View. In the Code and Text Editor, replace the *Application.Run* statement in the *Main* method with the following statement:

```
Application.Run(new EventLogForm());
```

Return to Design View window. Using Server Explorer, click and drag the Application event log and drop it on the form as you did in the previous exercise. An *EventLog* object called eventLog1 appears below the form.

5 Select eventLog1. In the Properties window, set the *EnableRaisingEvent* property to *True*.

This property is used to specify whether the object will be alerted when an entry is written to the event log.

6 In the Properties window, click Events. Create an event handler for the *EntryWritten* event called *logEntryWritten*.

A method called *writeLogClick* appears in the Code and Text Editor.

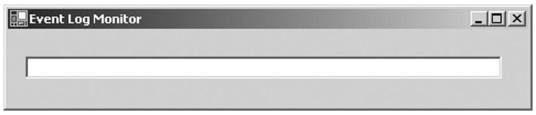
7 In the Code and Text Editor, add the following statement to the *logEntryWritten* method:

```
this.logEntry.Text = e.Entry.Message;
```

The *logEntryWritten* method takes an *EntryWrittenEventArgs* parameter named e. This type contains a property called *Entry*, which contains all the information written to the event log that is also accessible as properties. The *Message* property contains the text of the message written to the event log.

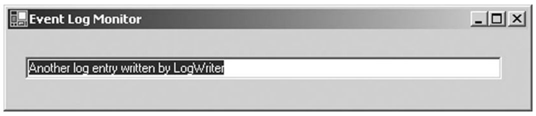
8 Build and execute the application.

The *EventLogMonitor* form appears, waiting for an event to occur.



9 In Microsoft Windows Explorer, navigate to the \Additional Lessons\Managing Systems Data\LogWriter\bin\Debug folder. Double-click LogWriter.exe to run the log writer application you created earlier. Type **Another log entry written by LogWriter** and then click the Write Log button. Switch back to the *EventLogMonitor* form.

The message is displayed in the text box on the form.

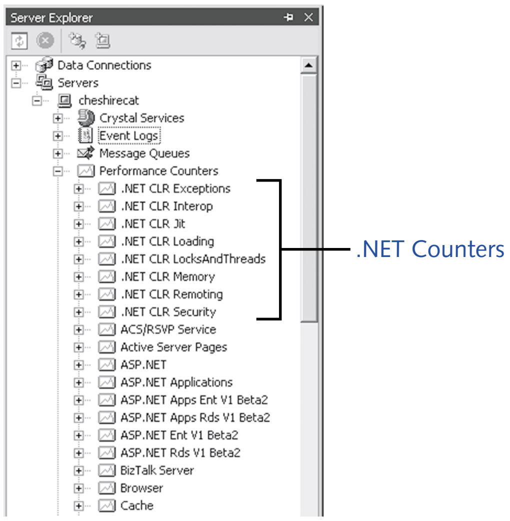


Another log entry written by LogWriter

10 Close both applications, and return to the Visual Studio .NET programming environment.

Monitoring Performance Counters

Performance counters are an invaluable tool for monitoring your computer and maintaining throughput. Numerous performance counters for monitoring disk I/O, memory use, CPU utilization, network load, and so on are installed with, and can be accessed through, the Windows operating system. Many applications and services also install their own performance counters—for example, Microsoft SQL Server supplies counters for measuring the performance of its databases and replication service. Similarly, the .NET Framework has counters for monitoring the performance of the common language runtime, including the .NET CLR Exceptions counter. You can see all of the available counters by expanding the Performance Counters node for your computer in Server Explorer, as shown in the following illustration.



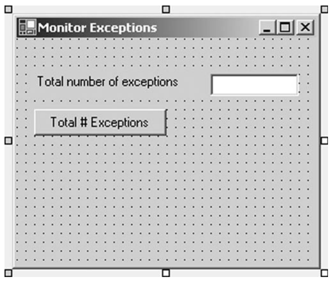
Performance counters can supply absolute values or sample data over a period of time (a rate) and generate a calculated value. In the following exercises, you will use the .NET performance counters to monitor the volume of CLR exceptions raised by applications running on your computer, both as an absolute value and as a rate.

Monitor the number of CLR exceptions

1. Create a new project called ExceptionMonitor in the \Additional Lessons\Managing Systems Data folder by using the Windows Application template.
2. In the Toolbox, drag a *Label* control, a *TextBox* control, and a *Button* control onto Form1.
3. In the Properties window, set the properties of the form and the controls to the values shown in the following table.

Control	Property	Value
Form1	*(Name)*	ExceptionForm
	Size	280, 232
	Text	Monitor Exceptions
label1	*Location*	16, 32
	Size	160, 23
	Text	Total number of exceptions
textBox1	*(Name)*	counterText
	Location	176, 32
	Size	80, 20
	Text	*Leave blank*
button1	*(Name)*	monitorTotalButton
	Location	16, 64
	Size	120, 23
	Text	Total # Exceptions

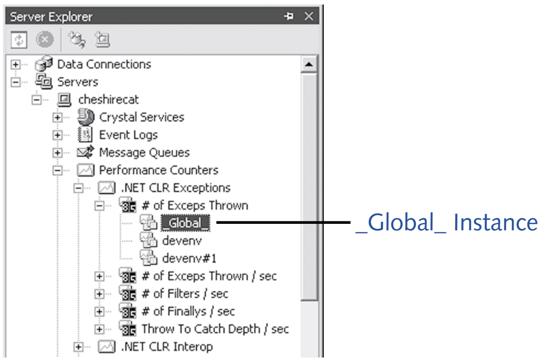
The form should look like the following image.



4. Switch to the Code and Text Editor window. Locate the *Main* method. Replace the *Application.Run* statement that you see with the following statement:

```
Application.Run(new ExceptionForm());
```

5 Switch to Design View. In Server Explorer, expand your computer name, expand Performance Counters, expand .NET CLR Exceptions, and then expand # Of Exceps Thrown. Click the _Global_ instance, and then drag it onto the form. This instance will track the total number of .NET exceptions thrown by all applications running on your computer.



A *PerformanceCounter* object called *performanceCounter1* appears under the form.

6 Select the *performanceCounter1* object, and in the Properties window, change the *(Name)* property to *exceptionCounterTotal*.

7 Select the *monitorTotalButton* button. In the Properties window, click Events. Create an event handler for the *Click* event called *totalButtonClick*.

A method called *totalButtonClick* appears in the Code and Text Editor window.

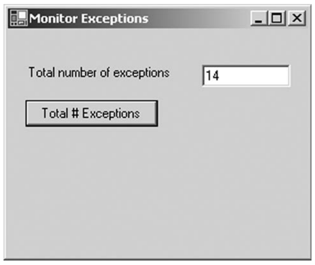
8 Add the following statement to the *totalButtonClick* method:

```
counterText.Text =
exceptionCounterTotal.NextValue().ToString();
```

The *NextValue* method retrieves the latest absolute value of this instance of the counter—in this case, the total number of .NET CLR exceptions that have occurred on your computer.

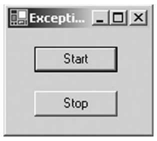
9 Compile and run the application. When the form appears, click Total # Exceptions.

The text box displays the number of exceptions that have occurred (it might be zero if you have not made any coding mistakes recently).



10 Leave the Monitor Exceptions form running, and then use Windows Explorer to run the ExceptionGenerator program located in the \Additional Lessons\Managing Systems Data\bin\Debug folder.

This application generates a steady stream of exceptions when you click Start (the source code is available in the \Additional Lessons\Managing Systems Data\ExceptionGenerator folder). When you click Stop, it stops generating exceptions.



11 Click Start, and then return to the Monitor Exceptions form and click Total # Exceptions again.

Notice that the number of exceptions reported has increased sharply.

12 Click Total # Exceptions again.

The number of exceptions reported should be even greater.

13 Return to the Exception Generator form, and click Stop. Close the Monitor Exceptions form (leave the Exception Generator form running), and return to the Visual Studio .NET programming environment.

Sample the rate of exceptions

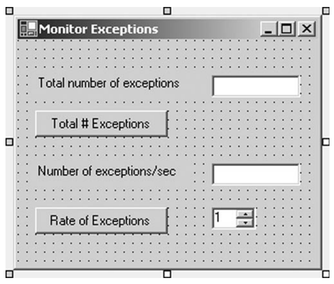
Performance counters let you capture a sample of data, which tracks the number of events that have occurred, together with a time stamp indicating when the sample was taken. If you take a second sample, you can determine the number of events that have occurred in the intervening period and the duration of that period, and thus calculate the rate at which events occurred in that time period. By using this technique, as well as reporting the absolute value of the number of .NET CLR exceptions generated, you can report the rate at which exceptions occur over a given time period.

1. Return to the ExceptionMonitor project, and display the *ExceptionForm* form in the Design View window.

2. In the Toolbox, add a *Label* control, a *TextBox* control, a *Button* control, and a *NumericUpDown* control to ExceptionForm. Set the properties of these controls to the values shown in the following table.

Control	Property	Value
label2	*Location*	16, 112
	Size	144, 23
	Text	Number of exceptions/sec
textBox1	*(Name)*	rateText
	Location	176, 112
	Size	80, 20
	Text	*Leave blank*
button1	*(Name)*	monitorRateButton
	Location	16, 152
	Size	120, 23
	Text	Rate of Exceptions
numericUpDown1	*(Name)*	numberSecs
	Location	176, 152
	Maximum	100
	Minimum	1
	Size	40, 20

The form should now look like the following image.



3. In Server Explorer, expand your computer name, expand Performance Counters, expand .NET CLR Exceptions, and then expand # Of Exceps Thrown. Click _Global_ instance, and then drag another instance onto the form. A second *PerformanceCounter* object called *performanceCounter1* is added under the form.

4 In the Properties window, change the *performanceCounter1* object's *(Name)* property to *exceptionCounterRate*.

5 Switch to the Code and Text Editor window. Type the following *using* statements at the top of Form1.cs:

```
using System.Diagnostics;
using System.Threading;
```

The *System.Diagnostics* namespace contains the classes used for handling performance counters. You will also use the *Sleep* method of the *Thread* class in the *System.Threading* namespace later.

6 Return to the Design View window and click the *monitorRateButton* button. In the Properties window, click Events. Create an event handler for the *Click* event called *rateButtonClick*.

A method called *rateButtonClick* appears in the Code and Text Editor window.

7 Type the following statement in the *rateButtonClick* method:

```
CounterSample firstSample =
exceptionCounterRate.NextSample();
```

This statement captures a time-stamped sample of the number of .NET CLR exceptions raised.

8 Add the following block of code in the *rateButtonClick* method:

```
Cursor curr = Cursor.Current;
Cursor.Current = Cursors.WaitCursor;
Thread.Sleep(Int32.Parse(numberSecs.Text) * 1000);
Cursor.Current = curr;
CounterSample secondSample =
exceptionCounterRate.NextSample();
```

This code changes the cursor to an hourglass and then sleeps for the number of seconds specified by the user in the *numberSecs NumericUpDown* control—the value entered by the user is multiplied by 1000 because the parameter to *Thread.Sleep* specifies a period of time in milliseconds. When the waiting period is over, the cursor is restored and a second sample of values is taken.

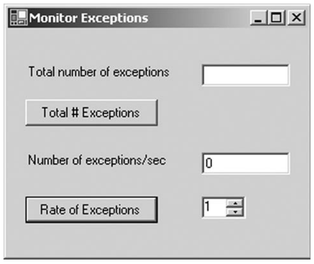
9 Append the following two statements to the *rateButtonClick* method:

```
float rate = CounterSample.Calculate(firstSample,
secondSample);
rateText.Text = rate.ToString();
```

The static *Calculate* method of the *CounterSample* structure takes two samples of data and then evaluates the number of times the sampled event (the number of exceptions) occurred per second. The result is displayed in the *rateText* text box.

10 Build and run the program. When the form appears, click Rate Of Exceptions.

The cursor changes to an hourglass for 1 second (the default period) and then changes back. If no exceptions have been raised during that time, you will see the value 0 in the Number Of Exceptions/Sec text box.



Total number of exceptions	
Number of exceptions/sec	0
Rate of Exceptions	1

11 Leave the form running, and run the ExceptionGenerator program (located in the \Additional Lessons\Managing Systems Data\bin\Debug folder). The *Exceptions* form appears. On the *Exceptions* form, click Start.

12 Switch back to the *Monitor Exceptions* form. On the *Monitor Exceptions* form, click Rate of Exceptions again.

After 1 second, you will see the rate at which the *Exception Generator* form is generating exceptions.

13 On the *Monitor Exceptions* form, change the interval to 5 seconds, and then click Rate of Exceptions.

You should see a similar rate of exceptions reported.

14 On the *Monitor Exceptions* form, click Rate of Exceptions for a final time, but before the 5-second period has expired, switch to the *Exceptions* form (the ExceptionGenerator program). On the *Exceptions* form, click Stop.

After 5 seconds, you will notice that the rate of exceptions has dropped.

15 Close both forms, and return to the Visual Studio .NET programming environment.

Managing Services

A Windows service is a process that runs in the background and provides access to some system function or data. Windows services are a fundamental part of the Windows operating system. One example is the Event Log service, which writes records to the event logs on behalf of applications (you used this service when you executed the *WriteEntry* method of an *EventLog* object earlier in this lesson). Another example is the MSSQLSERVER service—this is the SQL Server database server process. Services are usually configured to start when the operating system starts, but if you have the appropriate rights, you can start and stop them manually by using the Services tool in the Administrative Tools folder. You can also start and stop them programmatically, which is what you are going to do in the following exercise.

Examine the status of the ASP.NET State service

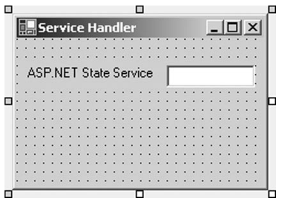
The ASP.NET State service is used by Microsoft ASP.NET to cache session state information for Web applications. It is installed as part of Visual Studio .NET. In this exercise, you will see how to examine the state of a service to determine if it is currently running.

1. Create a new project called ServiceHandler in the \Additional Lessons\Managing Systems Data folder by using the Windows Application template.

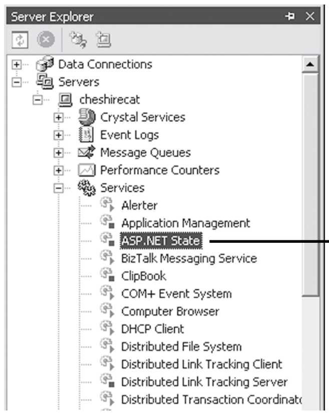
2. Add a *Label* control and a *TextBox* control to the form. Use the following table to set the properties of the form and the controls.

Control	Property	Value
Form1	*(Name)*	ServiceForm
	Size	232, 160
	Text	Service Handler
label1	*Location*	16, 24
	Size	124, 23
	Text	ASP.NET State Service
textBox1	*(Name)*	state
	Location	136, 24
	Size	80, 20
	Text	*Leave blank*

The form should look like the following image.



3. In Server Explorer, expand Services for your computer. Drag the ASP.NET State service onto your form.



A *ServiceController* object called *serviceController1* is added below the form. While *serviceController1* is still selected, change its name to *aspStateController*.

4 Switch to the Code and Text Editor window. Locate the *Main* method. Replace the *Application.Run* statement that you see with the following statement:

```
Application.Run(new ServiceForm());
```

5 Add the following *using* statement to the list of *using* statements at the top of the form:

```
using System.ServiceProcess;
```

The *System.ServiceProcess* namespace contains the classes, enumerations, and structures used for manipulating Windows services.

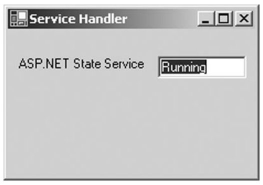
6 Create an event handler for the *Load* event of *ServiceForm* called *serviceFormLoad*. Type the following statements in the *serviceFormLoad* method:

```
ServiceControllerStatus status = aspStateController.Status;
state.Text = status.ToString();
```

This code queries the status of the ASP.NET State service and displays it in the text box on the form. *ServiceControllerStatus* is an enumeration that can contain the values *Running*, *Stopped*, *Paused*, and several others.

7 Build and run the application.

If the ASP.NET State service is running, you will see the status *Running*. If not, you will see the status *Stopped*.



ASP.NET State Service Running

8 Close the form, and return to the Visual Studio.NET programming environment.

Start and stop the ASP.NET State service

1 Add two *Button* controls to ServiceForm. Set their properties to the values in the following table.

Control	Property	Value
button1	*Name*	startButton
	Location	24, 80
	Text	Start
button2	*Name*	stopButton
	Location	120, 80
	Text	Stop

2 Create a *Click* event handler for the *Start* button called *startButtonClick*. Type the following block of code in the *startButtonClick* method:

```
ServiceControllerStatus status = aspStateController.Status;
if (status == ServiceControllerStatus.Stopped)
{
   Cursor curs = Cursor.Current;
   Cursor.Current = Cursors.WaitCursor;
   aspStateController.Start();
   aspStateController.WaitForStatus(ServiceControllerStatus.Running);
   status = aspStateController.Status;
   state.Text = status.ToString();
   Cursor.Current = curs;
}
```

You can only start a service only if it is currently in the Stopped state. The first two lines of code test the status of the service. If it is stopped, the cursor is changed to an hourglass, and the *Start* method is called to start the service running. The *Start* method only initiates the service; it does not wait for it to actually start. The *WaitForStatus* method sleeps until the service is in the specified state (Running). The status is displayed in the State text box, and the cursor is restored.

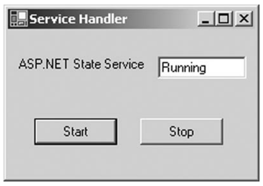
3 Create an event handler for the *Click* event called *stopButtonClick*. Add the following code to the *stopButtonClick* method:

```
if (aspStateController.CanStop)
{
   Cursor curs = Cursor.Current;
   Cursor.Current = Cursors.WaitCursor;
   aspStateController.Stop();
   aspStateController.WaitForStatus(ServiceControllerStatus.Stopped);
   ServiceControllerStatus status = aspStateController.Status;
   state.Text = status.ToString();
   Cursor.Current = curs;
}
```

The first statement checks to make sure that the service can be stopped. Not all services allow you to stop them—for example, the Event Log service—which is why you should use this method rather than checking the state as you did when starting the service.

The code inside the *if* statement changes the cursor to an hourglass, sends a stop signal to the service, waits for the service to stop, updates the status that is displayed, and then restores the cursor.

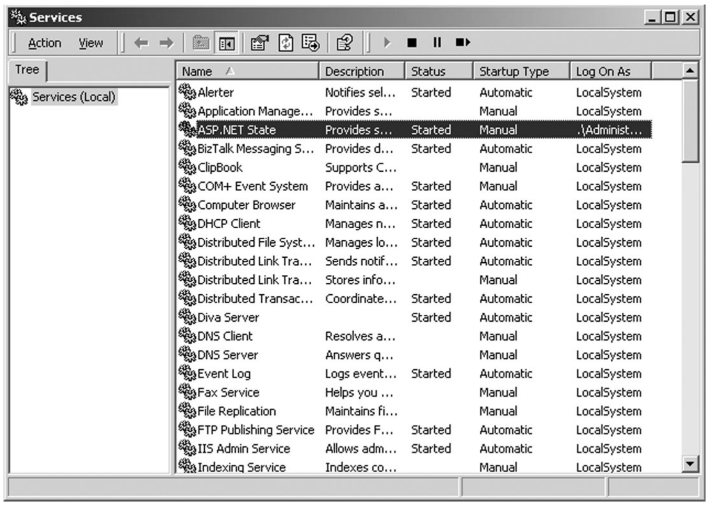
4 Build and run the application. If the status of the ASP.NET State service is Stopped, click Start.



ASP.NET State Service Running

5 On the taskbar, click Start, and click Control Panel. In Control Panel,1 click Performance and Maintenance, and then click Administrative Tools. Double-click Services.

The Services tool opens, displaying the services installed on your computer. The status of the ASP.NET State service should be Started.



Name	Description	Status	Startup Type	Log On As
Alerter	Notifies sel...	Started	Automatic	LocalSystem
Application Manage...	Provides s...		Manual	LocalSystem
ASP.NET State	Provides s...	Started	Manual	.\Administ...
BizTalk Messaging S...	Provides d...	Started	Automatic	LocalSystem
ClipBook	Supports C...		Manual	LocalSystem
COM+ Event System	Provides a...	Started	Manual	LocalSystem
Computer Browser	Maintains a...	Started	Automatic	LocalSystem
DHCP Client	Manages n...	Started	Automatic	LocalSystem
Distributed File Syst...	Manages lo...	Started	Automatic	LocalSystem
Distributed Link Tra...	Sends notif...	Started	Automatic	LocalSystem
Distributed Link Tra...	Stores info...		Manual	LocalSystem
Distributed Transac...	Coordinate...	Started	Automatic	LocalSystem
Diva Server		Started	Automatic	LocalSystem
DNS Client	Resolves a...		Manual	LocalSystem
DNS Server	Answers q...		Manual	LocalSystem
Event Log	Logs event...	Started	Automatic	LocalSystem
Fax Service	Helps you ...		Manual	LocalSystem
File Replication	Maintains fi...		Manual	LocalSystem
FTP Publishing Service	Provides F...	Started	Automatic	LocalSystem
IIS Admin Service	Allows adm...	Started	Automatic	LocalSystem
Indexing Service	Indexes co...		Manual	LocalSystem

6 Switch to the *Service Handler* form, and then click Stop. When the service has stopped and the status has changed, return to the Services tool. Press F5 to refresh the display.

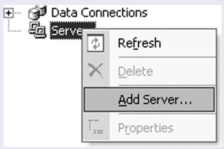
You will see that the status is now blank, which means Stopped.

7 On the *Service Handler* form, click Start again. When the service is running, switch to the Services tool and press F5.

The status of the ASP.NET State service is Started.

8 Close the Services tool, and then close the *Service Handler* form.

Quick Reference

To	Do this	Use this
Connect to a server	Use Server Explorer. Right-click Servers, and then click Add Server. Provide the details of the server you want to use.	 Data Connections Serve Refresh Delete Add Server... Properties
View event logs in Server Explorer	Expand Event Logs for your server.	
View performance counters in Server Explorer	Expand Performance Counters for your server.	
View services in Server Explorer	Expand Services for your server.	

To	Do this	Use this

Write a record to an event log	Instantiate an *EventLog* object. Set the *Source* property, and then execute the *WriteEntry* method. For example:
	`eventLog1.Source = "My App";`
	`eventLog1.WriteEntry("My Message",`
	`EventLogEntryType.Information);`	

Monitor an event log for changes	Subscribe to the *EntryWritten* event for the event log. The *EntryWrittenEventArgs* parameter to this event contains a property called *Entry*, which holds the details of the new record.	
Read the absolute value of a performance counter	Use the *NextValue* method of the *PerformanceCounter* object.	

Sample performance counter values of a period of time	Generate a *CounterSample* object containing time-stamped data by using the *NextSample* method. Wait for the defined period of time. Generate another *CounterSample* object. Call the static *CounterSample.Calculate* method passing in the two *CounterSample* objects as parameters to return the rate at which events occurred. For example:
	`CounterSample firstSample =`
	`exceptionCounterRate.NextSample();`
	`Thread.Sleep(...); CounterSample`
	`secondSample exception`
	`CounterRate.NextSample();`
	`float rate = CounterSample.Calculate`
	`(firstSample, secondSample);`	

Determine the status of a service	Read the *Status* property of the *ServiceController* object.	

(continued)

(continued)

To	Do this	Use this

Start a service	Verify that the service is stopped. Call the *Start* method on the *ServiceController* object. Wait for the service to start by using the *WaitForStatus* method. For example:
	`ServiceControllerStatus status =`
	`aspStateController.Status; if`
	`(status == ServiceControllerStatus.Stopped)`
	`{`
	` aspStateController.Start();`
	` aspStateController.WaitForStatus`
	`(ServiceControllerStatus.Running);`
	`}`	

Stop a service	Verify that the service can be stopped by querying the *CanStop* property of the *ServiceController* object. If the service can be stopped, call the *Stop* method and then wait for the service to stop by using the *WaitForStatus* method. For example:
	`if (aspStateController.CanStop)`
	`{`
	` aspStateController.Stop();`
	`aspStateController.WaitForStatus`
	`(ServiceControllerStatus.Stopped);`
	`}`	