Visual Studio Programmer's Guide for the .NET Framework

**Walkthrough: Creating a Windows Service Application in the Component Designer**

Updated: February 2009

|  |
| --- |
| **Description: NoteNote:** |
| The **Windows Service** template and associated functionality is not available in the Standard Edition of Visual Studio. |

The procedures in this topic demonstrate creating a simple Windows Service application that writes messages to an event log. The basic steps that you perform to create and use your service include the following:

* Create a project by using the **Windows Service** application template. This template creates a class for you that inherits from [ServiceBase](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase(VS.90).aspx) and writes much of the basic service code, such as the code to start the service.
* Write the code for the [OnStart](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstart(VS.90).aspx) and [OnStop](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstop(VS.90).aspx) procedures, and override any other methods that you want to redefine.
* Add the necessary installers for your service application. By default, a class that contains two or more installers is added to your application when you click the **Add Installer** link: one to install the process, and one for each associated service that your project contains.
* Build your project.
* Create a setup project to install your service, and then install it.
* Access the Windows 2000 Services Control Manager and start your service.

To begin, you create the project and set values that are required for the service to function correctly.

|  |
| --- |
| **Description: NoteNote:** |
| Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Visual Studio Settings](http://msdn.microsoft.com/en-us/library/zbhkx167(VS.90).aspx). |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating a Service

**To create and configure your service**

1. On the **File** menu, click **New Project**.

The **New Project** dialog box opens.

1. Select the **Windows Service** project from the list of Visual Basic, Visual C#, or Visual C++ project templates, and name it **MyNewService**. Click **OK**.

|  |
| --- |
| **Description: NoteNote:** |
| The project template automatically adds a component class named Service1 that inherits from [System.ServiceProcess..::.ServiceBase](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase(VS.90).aspx). |

1. Click the designer to select Service1. Then, in the **Properties** window, set the [ServiceName](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.servicename(VS.90).aspx) and the **(Name)** property for Service1 to **MyNewService**.
2. Set the [AutoLog](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.autolog(VS.90).aspx) property to **true**.
3. On the **View** menu, click **Code** to open the Code Editor. Edit the Main method to create an instance of **MyNewService**. When you renamed the service in step 3, the class name was not modified in the Main method. In Visual C# applications, the Main method is located in the Program.cs file.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl04_code');" \o "Copy Code)

' To access the Main method in Visual Basic, select Main from the

' method name drop-down list. This expands the Component Designer

' generated code region.

Shared Sub Main()

Dim ServicesToRun() As System.ServiceProcess.ServiceBase

' Change the following line to match.

ServicesToRun = New System.ServiceProcess.ServiceBase() \_

{New MyNewService()}

System.ServiceProcess.ServiceBase.Run(ServicesToRun)

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl05_code');" \o "Copy Code)

static void Main()

{

System.ServiceProcess.ServiceBase[] ServicesToRun;

// Change the following line to match.

ServicesToRun = new System.ServiceProcess.ServiceBase[]

{ new MyNewService() };

System.ServiceProcess.ServiceBase.Run(ServicesToRun);

}

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAdding Features to the Service

In the next section, you add a custom event log to the Windows service. Event logs are not associated in any way with Windows services. Here the [EventLog](http://msdn.microsoft.com/en-us/library/system.diagnostics.eventlog(VS.90).aspx) component is used as an example of the type of component you could add to a Windows service. For more information about custom event logs, see [How to: Create and Remove Custom Event Logs](http://msdn.microsoft.com/en-us/library/49dwckkz(VS.90).aspx).

**To add custom event log functionality to your service**

1. In **Solution Explorer**, right-click **Service1.vb** or **Service1.cs** and select **View Designer**.
2. From the **Components** tab of the **Toolbox**, drag an [EventLog](http://msdn.microsoft.com/en-us/library/system.diagnostics.eventlog(VS.90).aspx) component to the designer.
3. In **Solution Explorer**, right-click **Service1.vb** or **Service1.cs** and select **View Code**.
4. Edit the constructor to define a custom event log.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl03_code');" \o "Copy Code)

' To access the constructor in Visual Basic, select New from the

' method name drop-down list.

Public Sub New()

MyBase.New()

InitializeComponent()

If Not System.Diagnostics.EventLog.SourceExists("MySource") Then

System.Diagnostics.EventLog.CreateEventSource("MySource", \_

"MyNewLog")

End If

EventLog1.Source = "MySource"

EventLog1.Log = "MyNewLog"

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl04_code');" \o "Copy Code)

public MyNewService()

{

InitializeComponent();

if (!System.Diagnostics.EventLog.SourceExists("MySource"))

{

System.Diagnostics.EventLog.CreateEventSource(

"MySource","MyNewLog");

}

eventLog1.Source = "MySource";

eventLog1.Log = "MyNewLog";

}

**To define what occurs when the service starts**

* In the Code Editor, locate the [OnStart](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstart(VS.90).aspx) method that was automatically overridden when you created the project, and write code to determine what occurs when the service starts running:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl06_code');" \o "Copy Code)

' To access the OnStart in Visual Basic, select OnStart from the

' method name drop-down list.

Protected Overrides Sub OnStart(ByVal args() As String)

EventLog1.WriteEntry("In OnStart")

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl07_code');" \o "Copy Code)

protected override void OnStart(string[] args)

{

eventLog1.WriteEntry("In OnStart");

}

|  |
| --- |
| **Description: NoteNote:** |
| A service application is designed to be long running. Therefore, it usually polls or monitors something in the system. The monitoring is set up in the [OnStart](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstart(VS.90).aspx) method. However, [OnStart](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstart(VS.90).aspx) does not actually do the monitoring. The [OnStart](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstart(VS.90).aspx) method must return to the operating system after the service's operation has begun. It must not loop forever or block. To set up a simple polling mechanism, you can use the [System.Timers..::.Timer](http://msdn.microsoft.com/en-us/library/system.timers.timer(VS.90).aspx) component. In the [OnStart](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstart(VS.90).aspx) method, you would set parameters on the component, and then you would set the [Enabled](http://msdn.microsoft.com/en-us/library/system.timers.timer.enabled(VS.90).aspx) property to **true**. The timer would then raise events in your code periodically, at which time your service could do its monitoring. |

**To define what occurs when the service is stopped**

* In the Code Editor, select the [OnStop](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstop(VS.90).aspx) procedure from the **Method Name** drop-down list, which was automatically overridden when you created the project. Write code to determine what occurs when the service is stopped:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl16_code');" \o "Copy Code)

Protected Overrides Sub OnStop()

EventLog1.WriteEntry("In OnStop.")

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl17_code');" \o "Copy Code)

protected override void OnStop()

{

eventLog1.WriteEntry("In onStop.");

}

You can also override the [OnPause](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onpause(VS.90).aspx), [OnContinue](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.oncontinue(VS.90).aspx), and [OnShutdown](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onshutdown(VS.90).aspx) methods to define additional processing for your component.

**To define other actions for the service**

* For the method that you want to handle, override the appropriate method and define what you want to occur.

The following code shows what it looks like if you override the [OnContinue](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.oncontinue(VS.90).aspx) method:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl22_code');" \o "Copy Code)

Protected Overrides Sub OnContinue()

EventLog1.WriteEntry("In OnContinue.")

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl23_code');" \o "Copy Code)

protected override void OnContinue()

{

eventLog1.WriteEntry("In OnContinue.");

}

Some custom actions have to occur when a Windows service is installed, which can be done by the [Installer](http://msdn.microsoft.com/en-us/library/system.configuration.install.installer(VS.90).aspx) class. Visual Studio can create these installers specifically for a Windows service and add them to your project.

**To create the installers for your service**

1. In **Solution Explorer**, right-click **Service1.vb** or **Service1.cs** and select **View Designer**.
2. Click the background of the designer to select the service itself, instead of any of its contents.
3. With the designer in focus, right-click, and then click **Add Installer**.

By default, a component class that contains two installers is added to your project. The component is named **ProjectInstaller**, and the installers it contains are the installer for your service and the installer for the service's associated process.

1. In **Design** view for **ProjectInstaller**, click **ServiceInstaller1** or **serviceInstaller1**.
2. In the **Properties** window, set the [ServiceName](http://msdn.microsoft.com/en-us/library/system.serviceprocess.serviceinstaller.servicename(VS.90).aspx) property to **MyNewService**.
3. Set the [StartType](http://msdn.microsoft.com/en-us/library/system.serviceprocess.serviceinstaller.starttype(VS.90).aspx) property to Automatic.
4. In the designer, click **ServiceProcessInstaller1** (for a Visual Basic project), or **serviceProcessInstaller1** (for a Visual C# project). Set the [Account](http://msdn.microsoft.com/en-us/library/system.serviceprocess.serviceprocessinstaller.account(VS.90).aspx) property to LocalSystem. This will cause the service to be installed and to run on a local service account.

|  |
| --- |
| **Description: Security noteSecurity Note:** |
| The LocalSystem account has broad permissions, including the ability to write to the event log. Use this account with caution, because it might increase your risk of attacks from malicious software. For other tasks, consider using the LocalService account, which acts as a non-privileged user on the local computer and presents anonymous credentials to any remote server. |

**To build your service project**

1. In **Solution Explorer**, right-click your project and then click **Properties**. The project's **Property Designer** appears.
2. On the Application page, from the **Startup object** list, click **MyNewService**.
3. Press CTRL+SHIFT+B to build the project.

Now that the project is built, it can be deployed. A setup project will install the compiled project files and run the installers that are required to run the Windows service. To create a complete setup project you will have to add the project output, **MyNewService.exe**, to the setup project and then add a custom action to have **MyNewService.exe** installed. For more information about setup projects, see [Setup Projects](http://msdn.microsoft.com/en-us/library/996a3fxs(VS.90).aspx). For more information about custom actions, see [Walkthrough: Creating a Custom Action](http://msdn.microsoft.com/en-us/library/d9k65z2d(VS.90).aspx).

**To create a setup project for your service**

1. In **Solution Explorer**, right-click to select your solution, point to **Add**, and then click **New Project**.
2. In the **Project Types** pane, select the **Setup and Deployment Projects** folder.
3. In the **Templates** pane, select **Setup Project**. Name the project **MyServiceSetup**. Click **OK**.

A setup project is added to the solution.

Next you will add the output from the Windows service project, **MyNewService.exe**, to the setup.

**To add MyNewService.exe to the setup project**

1. In **Solution Explorer**, right-click **MyServiceSetup**, point to **Add**, and then click **Project Output**.

The **Add Project Output Group** dialog box appears.

1. **MyNewService** is selected in the **Project** box.
2. From the list, select **Primary Output**, and click **OK**.

A project item for the primary output of **MyNewService** is added to the setup project.

Now add a custom action to install the MyNewService.exe file.

**To add a custom action to the setup project**

1. In **Solution Explorer**, right-click the setup project, point to **View**, and then click **Custom Actions**.

The **Custom Actions** editor appears.

1. In the **Custom Actions** editor, right-click the **Custom Actions** node and click **Add Custom Action**.

The **Select Item in Project** dialog box appears.

1. Double-click the **Application Folder** in the list to open it, select **Primary Output from MyNewService (Active)**, and click **OK**.

The primary output is added to all four nodes of the custom actions — **Install**, **Commit**, **Rollback**, and **Uninstall**.

1. In **Solution Explorer**, right-click the **MyServiceSetup** project and click **Build**.

**To install the Windows Service**

1. To install **MyNewService.exe**, right-click the setup project in **Solution Explorer** and select **Install**.
2. Follow the steps in the **Setup Wizard**. Build and save your solution.

**To start and stop your service**

1. Open the Services Control Manager by doing one of the following:
   * In Windows XP and 2000 Professional, right-click **My Computer** on the desktop, and then click **Manage**. In the **Computer Management** console, expand the **Services and Applications** node.

- or -

* + In Windows Server 2003 and Windows 2000 Server, click **Start**, point to **Programs**, click **Administrative Tools**, and then click **Services**.

|  |
| --- |
| **Description: NoteNote:** |
| In Windows NT version 4.0, you can open this dialog box from Control Panel. |

1. You should now see **MyNewService** listed in the **Services** section of the window.
2. Select your service in the list, right-click it, and then click **Start**.
3. Right-click the service, and then click **Stop**.

**To verify the event log output of your service**

1. Open **Server Explorer** and access the **Event Logs** node. For more information, see [How to: Work with Event Logs in Server Explorer](http://msdn.microsoft.com/en-us/library/f82akt63(VS.90).aspx).

|  |
| --- |
| **Description: NoteNote:** |
| The **Windows Service** template and associated functionality is not available in the Standard Edition of Visual Studio. |

1. Locate the listing for **MyNewLog** and expand it. You should see entries for the actions your service has performed.

**To uninstall your service**

1. On the **Start** menu, open **Control Panel** and click **Add or Remove Programs**, and then locate your service and click **Uninstall**.
2. You can also uninstall the program by right-clicking the program icon for the .msi file and selecting **Uninstall**.

|  |
| --- |
| **Description: NoteNote:** |
| If you installed the service on Windows 2000, you will have to restart the system before you can reinstall the service. In Windows 2000, services are not completely deleted until the system is rebooted. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifNext Steps

You might explore the use of a [ServiceController](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller(VS.90).aspx) component to enable you to send commands to the service you have installed. For more information about using the [ServiceController](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller(VS.90).aspx) component, see [Monitoring Windows Services](http://msdn.microsoft.com/en-us/library/t12x806t(VS.90).aspx).

You can use an installer to create an event log when the application is installed instead of creating the event log when the application runs. Additionally, the event log will be deleted by the installer when the application is uninstalled. For more information, see [Walkthrough: Installing an Event Log Component](http://msdn.microsoft.com/en-us/library/f5dcf6h3(VS.90).aspx).