Walkthrough: Creating a Windows Service Application in the Component Designer

.NET Framework (current version)

This article demonstrates how to create a simple Windows Service application in Visual Studio that writes messages to an event log. Here are the basic steps that you perform to create and use your service:

1. Creating a Service by using the **Windows Service** project template, and configure it. This template creates a class for you that inherits from System.ServiceProcess.ServiceBase and writes much of the basic service code, such as the code to start the service.

2. Adding Features to the Service for the OnStart and OnStop procedures, and override any other methods that you want to redefine.

3. Setting Service Status. By default, services created with System.ServiceProcess.ServiceBase implement only a subset of the available status flags. If your service takes a long time to start up, pause, or stop, you can implement status values such as Start Pending or Stop Pending to indicate that it's working on an operation.

4. Adding Installers to the Service for your service application.

5. (Optional) Set Startup Parameters, specify default startup arguments, and enable users to override default settings when they start your service manually.

6. Building the Service.

7. Installing the Service on the local machine.

8. Access the Windows Service Control Manager and Starting and Running the Service.

9. Uninstalling a Windows Service.

Warning

The Windows Services project template that is required for this walkthrough is not available in the Express edition of Visual Studio.

Note

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 Personalizing the IDE.

Creating a Service

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

To create and configure your service

1. In Visual Studio, on the menu bar, choose **File**, **New**, **Project**.

   The **New Project** dialog box opens.

2. In the list of Visual Basic or Visual C# project templates, choose **Windows Service**, and name the project **MyNewService**. Choose **OK**.

   The project template automatically adds a component class named `Service1` that inherits from System.ServiceProcess.ServiceBase.

3. On the **Edit** menu, choose **Find and Replace**, **Find in Files** (Keyboard: Ctrl+Shift+F). Change all occurrences of `Service1` to `MyNewService`. You'll find instances in Service1.cs, Program.cs, and Service1.Designer.cs (or their .vb equivalents).

4. In the **Properties** window for **Service1.cs [Design]** or **Service1.vb [Design]**, set the ServiceName and the **(Name)** property for `Service1` to **MyNewService**, if it's not already set.

5. In Solution Explorer, rename **Service1.cs** to **MyNewService.cs**, or **Service1.vb** to **MyNewService.vb**.

Adding Features to the Service

In this 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 component is used as an example of the type of component you could add to a Windows service.

To add custom event log functionality to your service

1. In **Solution Explorer**, open the context menu for **MyNewService.cs** or **MyNewService.vb**, and then choose **View Designer**.

2. From the **Components** section of the **Toolbox**, drag an EventLog component to the designer.

3. In **Solution Explorer**, open the context menu for **MyNewService.cs** or **MyNewService.vb**, and then choose **View Code**.

4. Add a declaration for the **eventLog** object in the `MyNewService` class, right after the line that declares the `components` variable:

C#

```
    private System.ComponentModel.IContainer components;
    private System.Diagnostics.EventLog eventLog1;
```

5. Add or edit the constructor to define a custom event log:

C#

```
            public MyNewService()
            {
                    InitializeComponent();
                    eventLog1 = new System.Diagnostics.EventLog();
                    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 method that was automatically overridden when you created the project, and replace the code with the following. This adds an entry to the event log when the service starts running:

C#

```
            protected override void OnStart(string[] args)
            {
                    eventLog1.WriteEntry("In OnStart");
            }
```

A service application is designed to be long-running, so it usually polls or monitors something in the system. The monitoring is set up in the OnStart method. However, OnStart doesn't actually do the monitoring. The OnStart 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 component as follows: In the OnStart method, set parameters on the component, and then set the Enabled property to `true`. The timer raises events in your code periodically, at which time your service could do its monitoring. You can use the following code to do this:

C#

```
// Set up a timer to trigger every minute.
System.Timers.Timer timer = new System.Timers.Timer();
timer.Interval = 60000; // 60 seconds
timer.Elapsed += new System.Timers.ElapsedEventHandler(this.OnTimer);
timer.Start();
```

Add code to handle the timer event:

C#

```
public void OnTimer(object sender, System.Timers.ElapsedEventArgs args)
{
    // TODO: Insert monitoring activities here.
    eventLog1.WriteEntry("Monitoring the System", EventLogEntryType.Information,
eventId++);
}
```

You might want to perform tasks by using background worker threads instead of running all your work on the main thread. For an example of this, see the System.ServiceProcess.ServiceBase reference page.

To define what occurs when the service is stopped

- Replace the code for the OnStop method with the following. This adds an entry to the event log when the service is stopped:

C#

```
        protected override void OnStop()
        {
                eventLog1.WriteEntry("In onStop.");
        }
```

In the next section, you can override the OnPause, OnContinue, and OnShutdown methods to define additional processing for your component.

To define other actions for the service

- Locate the method that you want to handle, and override it to define what you want to occur.

  The following code shows how you can override the OnContinue method:

C#

```csharp
        protected override void OnContinue()
        {
                eventLog1.WriteEntry("In OnContinue.");
        }
```

Some custom actions have to occur when a Windows service is installed by the Installer class. Visual Studio can create these installers specifically for a Windows service and add them to your project.

Setting Service Status

Services report their status to the Service Control Manager, so that users can tell whether a service is functioning correctly. By default, services that inherit from ServiceBase report a limited set of status settings, including Stopped, Paused, and Running. If a service takes a little while to start up, it might be helpful to report a Start Pending status. You can also implement the Start Pending and Stop Pending status settings by adding code that calls into the Windows SetServiceStatus function.

To implement service pending status

1. Add a `using` statement or `Imports` declaration to the System.Runtime.InteropServices namespace in the MyNewService.cs or MyNewService.vb file:

   C#

   ```csharp
   using System.Runtime.InteropServices;
   ```

2. Add the following code to MyNewService.cs to declare the `ServiceState` values and to add a structure for the status, which you'll use in a platform invoke call:

   C#

   ```csharp
   public enum ServiceState
     {
         SERVICE_STOPPED = 0x00000001,
         SERVICE_START_PENDING = 0x00000002,
         SERVICE_STOP_PENDING = 0x00000003,
         SERVICE_RUNNING = 0x00000004,
         SERVICE_CONTINUE_PENDING = 0x00000005,
         SERVICE_PAUSE_PENDING = 0x00000006,
         SERVICE_PAUSED = 0x00000007,
     }
   ```

```
    [StructLayout(LayoutKind.Sequential)]
    public struct ServiceStatus
    {
        public long dwServiceType;
        public ServiceState dwCurrentState;
        public long dwControlsAccepted;
        public long dwWin32ExitCode;
        public long dwServiceSpecificExitCode;
        public long dwCheckPoint;
        public long dwWaitHint;
    };
```

3. Now, in the `MyNewService` class, declare the SetServiceStatus function by using platform invoke:

C#

```
[DllImport("advapi32.dll", SetLastError=true)]
        private static extern bool SetServiceStatus(IntPtr handle, ref
ServiceStatus serviceStatus);
```

4. To implement the Start Pending status, add the following code to the beginning of the OnStart method:

C#

```
// Update the service state to Start Pending.
ServiceStatus serviceStatus = new ServiceStatus();
serviceStatus.dwCurrentState = ServiceState.SERVICE_START_PENDING;
serviceStatus.dwWaitHint = 100000;
SetServiceStatus(this.ServiceHandle, ref serviceStatus);
```

5. Add code to set the status to Running at the end of the OnStart method.

```
// Update the service state to Running.
serviceStatus.dwCurrentState = ServiceState.SERVICE_RUNNING;
SetServiceStatus(this.ServiceHandle, ref serviceStatus);
```

VB

```
' Update the service state to Running.
serviceStatus.dwCurrentState = ServiceState.SERVICE_RUNNING
```

```
SetServiceStatus(Me.ServiceHandle, serviceStatus)
```

6. (Optional) Repeat this procedure for the OnStop method.

> ⚠ **Caution**
>
> The Service Control Manager uses the `dwWaitHint` and `dwCheckpoint` members of the SERVICE_STATUS structure to determine how much time to wait for a Windows Service to start or shut down. If your OnStart and OnStop methods run long, your service can request more time by calling SetServiceStatus again with an incremented `dwCheckPoint` value.

Adding Installers to the Service

Before you can run a Windows Service, you need to install it, which registers it with the Service Control Manager. You can add installers to your project that handle the registration details.

To create the installers for your service

1. In **Solution Explorer**, open the context menu for **MyNewService.cs** or **MyNewService.vb**, and then choose **View Designer**.

2. Click the background of the designer to select the service itself, instead of any of its contents.

3. Open the context menu for the designer window (if you’re using a pointing device, right-click inside the window), and then choose **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.

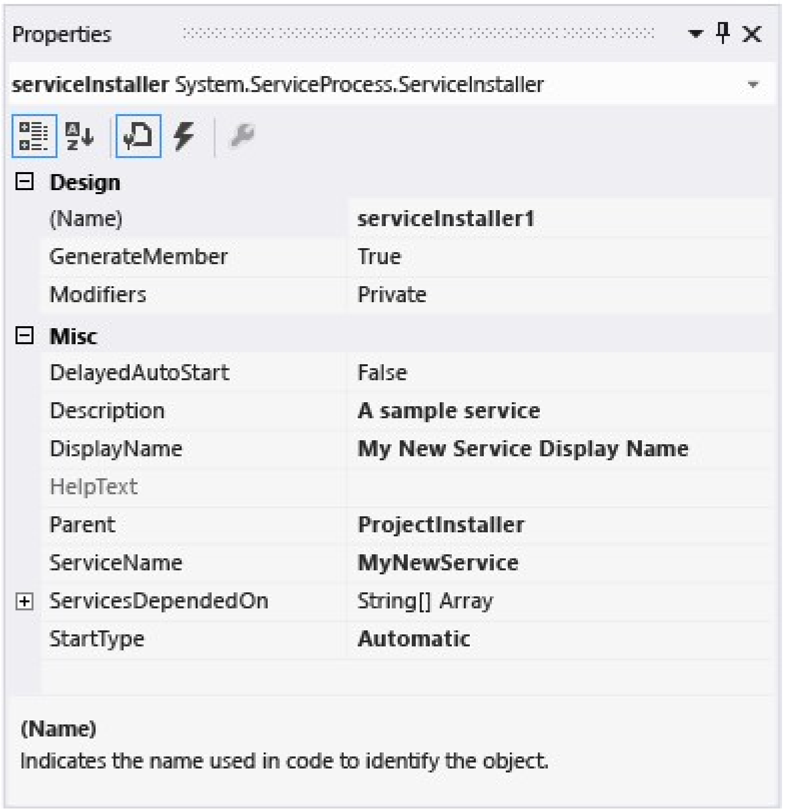
4. In **Design** view for **ProjectInstaller**, choose **serviceInstaller1** for a Visual C# project, or **ServiceInstaller1** for a Visual Basic project.

5. In the **Properties** window, make sure the ServiceName property is set to **MyNewService**.

6. Set the **Description** property to some text, such as "A sample service". This text appears in the Services window and helps the user identify the service and understand what it’s used for.

7. Set the DisplayName property to the text that you want to appear in the Services window in the **Name** column. For example, you can enter "MyNewService Display Name". This name can be different from the ServiceName property, which is the name used by the system (for example, when you use the `net start` command to start your service).

8. Set the StartType property to Automatic.


Properties

serviceInstaller System.ServiceProcess.ServiceInstaller

Design	
(Name)	serviceInstaller1
GenerateMember	True
Modifiers	Private
Misc	
DelayedAutoStart	False
Description	**A sample service**
DisplayName	**My New Service Display Name**
HelpText	
Parent	**ProjectInstaller**
ServiceName	**MyNewService**
ServicesDependedOn	String[] Array
StartType	**Automatic**

(Name)
Indicates the name used in code to identify the object.

9. In the designer, choose **serviceProcessInstaller1** for a Visual C# project, or **ServiceProcessInstaller1** for a Visual Basic project. Set the Account property to LocalSystem. This will cause the service to be installed and to run on a local service account.

Important

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. This example fails if you try to use the LocalService account, because it needs permission to write to the event log.

For more information about installers, see How to: Add Installers to Your Service Application.

Set Startup Parameters

A Windows Service, like any other executable, can accept command-line arguments, or startup parameters. When you add code to process startup parameters, users can start your service with their own custom startup parameters by using the Services window in the Windows Control Panel. However, these startup parameters are not persisted the next time the service starts. To set startup parameters permanently, you can set them in the registry, as shown in this procedure.

📝 **Note**

Before you decide to add startup parameters, consider whether that is the best way to pass information to your service. Although startup parameters are easy to use and to parse, and users can easily override them, they might be harder for users to discover and use without documentation. Generally, if your service requires more than just a few startup parameters, you should consider using the registry or a configuration file instead. Every Windows Service has an entry in the registry under HKLM\System\CurrentControlSet\services. Under the service's key, you can use the **Parameters** subkey to store information that your service can access. You can use application configuration files for a Windows Service the same way you do for other types of programs. For example code, see AppSettings.

Adding startup parameters

1. In the `Main` method in Program.cs or in MyNewService.Designer.vb, add an argument for the command line:

   C#

```
static void Main(string[] args)        {            ServiceBase[] ServicesToRun;
ServicesToRun = new ServiceBase[]            {                 new MyNewService
(args)              };            ServiceBase.Run(ServicesToRun);        }
```

2. Change the `MyNewService` constructor as follows:

   C#

```
public MyNewService(string[] args)        {            InitializeComponent
();            string eventSourceName = "MySource";            string logName =
"MyNewLog";            if (args.Count() > 0)            {
eventSourceName = args[0];            }            if (args.Count() > 1)
{                logName = args[1];            }            eventLog1 = new
System.Diagnostics.EventLog();            if (!
System.Diagnostics.EventLog.SourceExists(eventSourceName))            {
System.Diagnostics.EventLog.CreateEventSource(
eventSourceName, logName);            }            eventLog1.Source =
eventSourceName;            eventLog1.Log = logName;        }
```

   This code sets the event source and log name according to the supplied startup parameters, or uses default values if no arguments are supplied.

3. To specify the command-line arguments, add the following code to the `ProjectInstaller` class in ProjectInstaller.cs or ProjectInstaller.vb:

   C#

```
protected override void OnBeforeInstall(IDictionary savedState)
{             string parameter = "MySource1\" \"MyLogFile1";
Context.Parameters["assemblypath"] = "\"" + Context.Parameters["assemblypath"]
+ "\" \"" + parameter + "\"";            base.OnBeforeInstall(savedState);
}
```

This code modifies the **ImagePath** registry key, which typically contains the full path to the executable for the Windows Service, by adding the default parameter values. The quotation marks around the path (and around each individual parameter) are required for the service to start up correctly. To change the startup parameters for this Windows Service, users can change the parameters given in the **ImagePath** registry key, although the better way is to change it programmatically and expose the functionality to users in a friendly way (for example, in a management or configuration utility).

Building the Service

To build your service project

1. In **Solution Explorer**, open the context menu for your project, and then choose **Properties**. The property pages for your project appear.

2. On the Application tab, in the **Startup object** list, choose **MyNewService.Program**.

3. In **Solution Explorer**, open the context menu for your project, and then choose **Build** to build the project (Keyboard: Ctrl+Shift+B).

Installing the Service

Now that you've built the Windows service, you can install it. To install a Windows service, you must have administrative credentials on the computer on which you're installing it.

To install a Windows Service

1. In Windows 7 and Windows Server, open the **Developer Command Prompt** under **Visual Studio Tools** in the **Start** menu. In Windows 8 or Windows 8.1, choose the **Visual Studio Tools** tile on the **Start** screen, and then run Developer Command Prompt with administrative credentials. (If you’re using a mouse, right-click on **Developer Command Prompt**, and then choose **Run as Administrator**.)

2. In the Command Prompt window, navigate to the folder that contains your project's output. For example, under your My Documents folder, navigate to Visual Studio 2013\Projects\MyNewService\bin\Debug.

3. Enter the following command:

```
installutil.exe MyNewService.exe
```

If the service installs successfully, installutil.exe will report success. If the system could not find InstallUtil.exe, make sure that it exists on your computer. This tool is installed with the .NET Framework to the folder `%WINDIR%\Microsoft.NET\Framework[64]\`*framework_version*. For example, the default path for the 32-bit version of the .NET Framework 4, 4.5, 4.5.1, and 4.5.2 is `C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe`.

If the installutil.exe process reports failure, check the install log to find out why. By default the log is in the same folder as the service executable. The installation can fail if the RunInstallerAttribute Class is not present on the `ProjectInstaller` class, or else the attribute is not set to `true`, or else the `ProjectInstaller` class is not `public`.

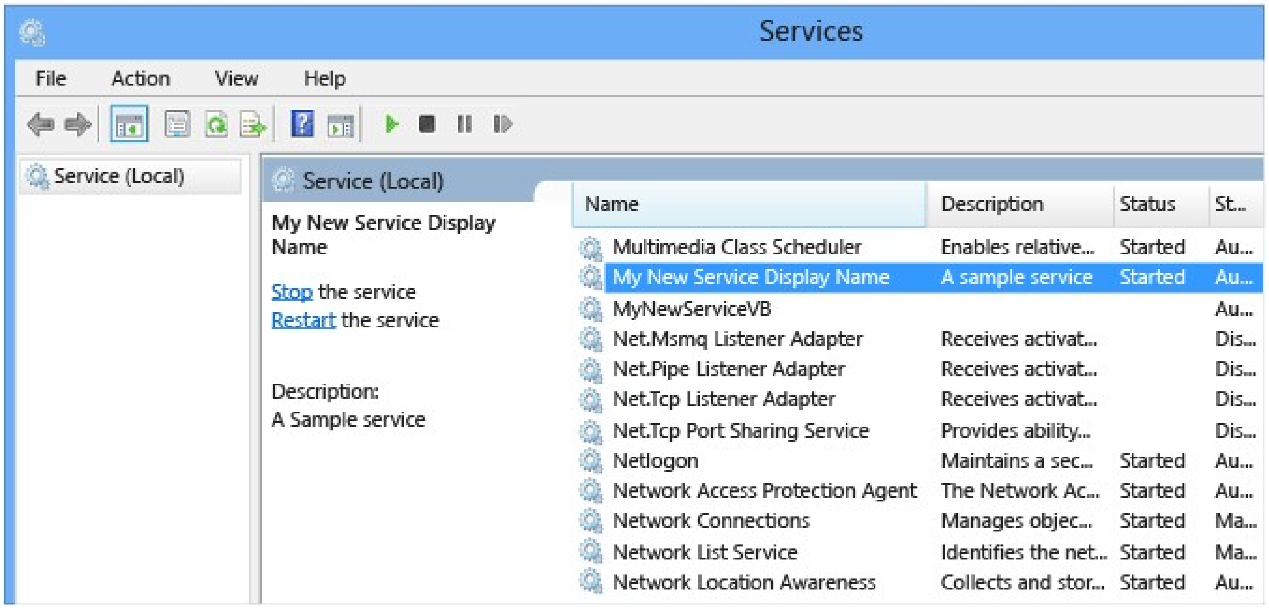
For more information, see How to: Install and Uninstall Services.

Starting and Running the Service

To start and stop your service

1. In Windows, open the **Start** screen or **Start** menu, and type `services.msc`.

   You should now see **MyNewService** listed in the **Services** window.



My New Service Display Name

Stop the service
Restart the service

Description:
A Sample service

Name	Description	Status	St...
Multimedia Class Scheduler	Enables relative...	Started	Au...
My New Service Display Name	A sample service	Started	Au...
MyNewServiceVB			Au...
Net.Msmq Listener Adapter	Receives activat...		Dis...
Net.Pipe Listener Adapter	Receives activat...		Dis...
Net.Tcp Listener Adapter	Receives activat...		Dis...
Net.Tcp Port Sharing Service	Provides ability...		Dis...
Netlogon	Maintains a sec...	Started	Au...
Network Access Protection Agent	The Network Ac...	Started	Au...
Network Connections	Manages objec...	Started	Ma...
Network List Service	Identifies the net...	Started	Ma...
Network Location Awareness	Collects and stor...	Started	Au...

2. In the **Services** window, open the shortcut menu for your service, and then choose **Start**.

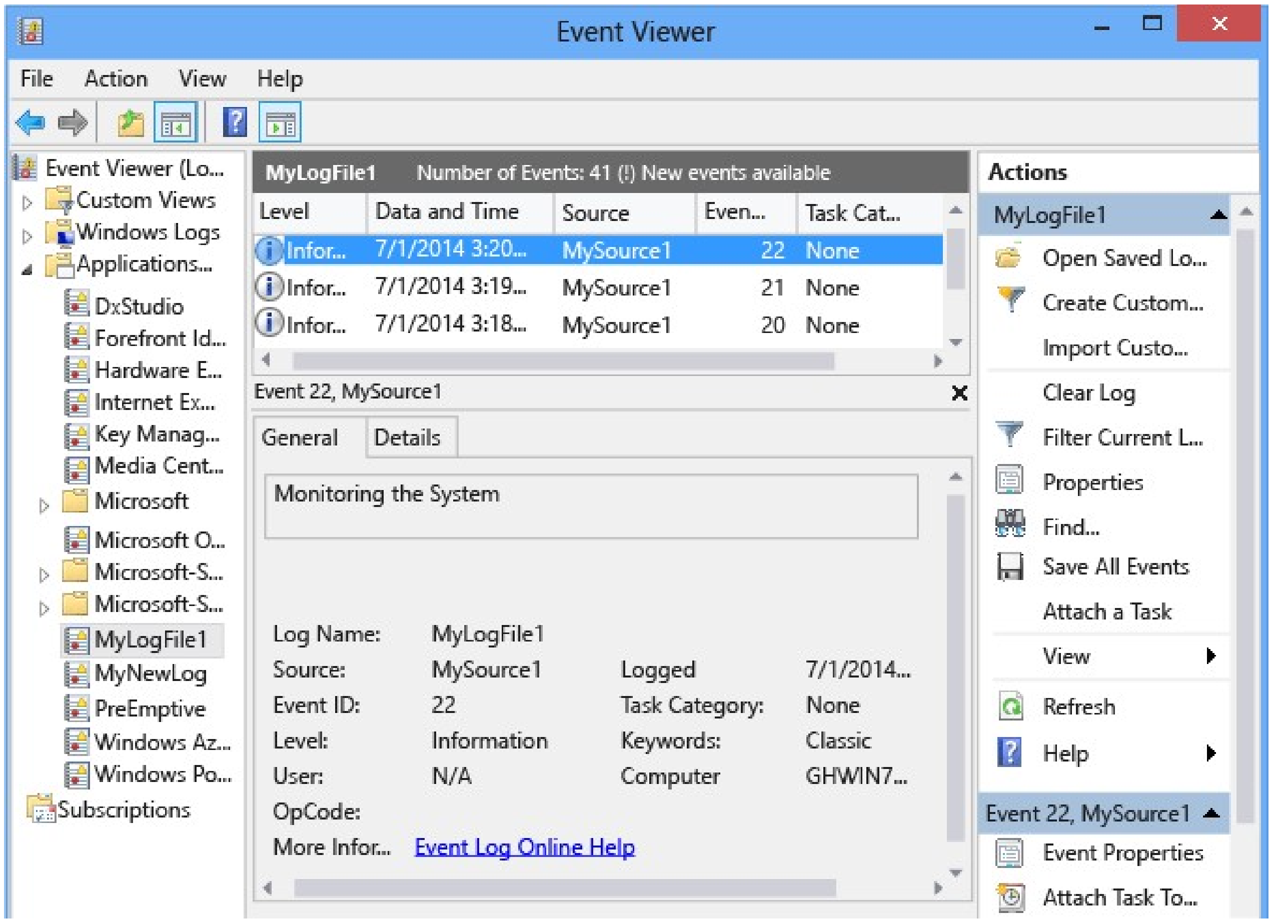
3. Open the shortcut menu for the service, and then choose **Stop**.

4. (Optional) From the command line, you can use the commands `net start``ServiceName` and `net stop``ServiceName` to start and stop your service.

To verify the event log output of your service

1. In Visual Studio, open **Server Explorer** (Keyboard: Ctrl+Alt+S), and access the **Event Logs** node for the local computer.

2. Locate the listing for **MyNewLog** (or **MyLogFile1**, if you used the optional procedure to add command-line arguments) and expand it. You should see entries for the two actions (start and stop) your service has performed.



MyLogFile1 Number of Events: 41 (!) New events available

Level	Data and Time	Source	Even...	Task Cat...
Infor...	7/1/2014 3:20...	MySource1	22	None
Infor...	7/1/2014 3:19...	MySource1	21	None
Infor...	7/1/2014 3:18...	MySource1	20	None

Event 22, MySource1

General Details

Monitoring the System

Log Name:	MyLogFile1		
Source:	MySource1	Logged	7/1/2014...
Event ID:	22	Task Category:	None
Level:	Information	Keywords:	Classic
User:	N/A	Computer	GHWIN7...
OpCode:			
More Infor...	Event Log Online Help		

Uninstalling a Windows Service

To uninstall your service

1. Open a developer command prompt with administrative credentials.

2. In the Command Prompt window, navigate to the folder that contains your project's output. For example, under your My Documents folder, navigate to Visual Studio 2013\Projects\MyNewService\bin\Debug.

3. Enter the following command:

```
installutil.exe /u MyNewService.exe
```

If the service uninstalls successfully, installutil.exe will report that your service was successfully removed. For more information, see How to: Install and Uninstall Services.

Next Steps

You can create a standalone setup program that others can use to install your Windows service, but it requires additional steps. ClickOnce doesn't support Windows services, so you can't use the Publish Wizard. You can use a full edition of InstallShield, which Microsoft doesn't provide. For more information about InstallShield, see InstallShield Limited Edition. You can also use the Windows Installer XML Toolset to create an installer for a Windows service.

You might explore the use of a ServiceController component, which enables you to send commands to the service you have installed.

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 the EventLogInstaller reference page.

See Also

Windows Service Applications
Introduction to Windows Service Applications
How to: Debug Windows Service Applications
Services (Windows)