1. Walkthrough: Creating a Windows Service Application in the Component Designer

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](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#BK_CreateProject) by using the **Windows Service** project template, and configure it. This template creates a class for you that inherits from [System.ServiceProcess.ServiceBase](https://docs.microsoft.com/en-us/dotnet/api/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](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#BK_WriteCode) for the [OnStart](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstart) and [OnStop](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstop) procedures, and override any other methods that you want to redefine.
3. [Setting Service Status](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#BK_SetStatus). By default, services created with [System.ServiceProcess.ServiceBase](https://docs.microsoft.com/en-us/dotnet/api/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](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#BK_AddInstallers) for your service application.
5. (Optional) [Set Startup Parameters](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#BK_StartupParameters), specify default startup arguments, and enable users to override default settings when they start your service manually.
6. [Building the Service](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#BK_Build).
7. [Installing the Service](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#BK_Install) on the local machine.
8. Access the Windows Service Control Manager and [Starting and Running the Service](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#BK_StartService).
9. [Uninstalling a Windows Service](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#BK_Uninstall).

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](https://docs.microsoft.com/en-us/visualstudio/ide/personalizing-the-visual-studio-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.

1. 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](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase).

1. 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).
2. In the **Properties** window for **Service1.cs [Design]** or **Service1.vb [Design]**, set the [ServiceName](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.servicename) and the **(Name)** property for Service1 to **MyNewService**, if it's not already set.
3. 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](https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.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](https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.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#Copy

private System.ComponentModel.IContainer components;

private System.Diagnostics.EventLog eventLog1;

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

C#Copy

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";

}

2

To define what occurs when the service starts

* In the Code Editor, locate the [OnStart](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.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#Copy

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](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstart) method. However, [OnStart](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstart) doesn’t actually do the monitoring. The [OnStart](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.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](https://docs.microsoft.com/en-us/dotnet/api/system.timers.timer) component as follows: In the [OnStart](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstart) method, set parameters on the component, and then set the [Enabled](https://docs.microsoft.com/en-us/dotnet/api/system.timers.timer.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#Copy

// 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 a member variable to the class. It will contain the identifier of the next event to write into the event log.

C#Copy

private int eventId = 1;

Add code to handle the timer event:

C#Copy

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](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase) reference page.

1

To define what occurs when the service is stopped

* Replace the code for the [OnStop](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstop) method with the following. This adds an entry to the event log when the service is stopped:

C#Copy

protected override void OnStop()

{

eventLog1.WriteEntry("In onStop.");

}

In the next section, you can override the [OnPause](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onpause), [OnContinue](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.oncontinue), and [OnShutdown](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.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](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.oncontinue) method:

C#Copy

protected override void OnContinue()

{

eventLog1.WriteEntry("In OnContinue.");

}

1

Some custom actions have to occur when a Windows service is installed by the [Installer](https://docs.microsoft.com/en-us/dotnet/api/system.configuration.install.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](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.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](http://msdn.microsoft.com/library/windows/desktop/ms686241.aspx).

To implement service pending status

1. Add a using statement or Imports declaration to the [System.Runtime.InteropServices](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices) namespace in the MyNewService.cs or MyNewService.vb file:

C#Copy

using System.Runtime.InteropServices;

1. 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#Copy

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 int dwServiceType;

public ServiceState dwCurrentState;

public int dwControlsAccepted;

public int dwWin32ExitCode;

public int dwServiceSpecificExitCode;

public int dwCheckPoint;

public int dwWaitHint;

};

1. Now, in the MyNewService class, declare the [SetServiceStatus function](http://msdn.microsoft.com/library/windows/desktop/ms686241.aspx) by using platform invoke:

C#Copy

[DllImport("advapi32.dll", SetLastError=true)]

private static extern bool SetServiceStatus(IntPtr handle, ref ServiceStatus serviceStatus);

1. To implement the Start Pending status, add the following code to the beginning of the [OnStart](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstart) method:

C#Copy

// 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);

1. Add code to set the status to Running at the end of the [OnStart](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstart) method.

C#Copy

// Update the service state to Running.

serviceStatus.dwCurrentState = ServiceState.SERVICE\_RUNNING;

SetServiceStatus(this.ServiceHandle, ref serviceStatus);

1. (Optional) Repeat this procedure for the [OnStop](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstop) method.

1

Caution

The [Service Control Manager](http://msdn.microsoft.com/library/windows/desktop/ms685150.aspx) uses the dwWaitHint and dwCheckpoint members of the [SERVICE\_STATUS structure](http://msdn.microsoft.com/library/windows/desktop/ms685996.aspx) to determine how much time to wait for a Windows Service to start or shut down. If your [OnStart](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstart) and [OnStop](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase.onstop) methods run long, your service can request more time by calling [SetServiceStatus](http://msdn.microsoft.com/library/windows/desktop/ms686241.aspx) 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.1

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.

1. In **Design** view for **ProjectInstaller**, choose **serviceInstaller1** for a Visual C# project, or **ServiceInstaller1** for a Visual Basic project.
2. In the **Properties** window, make sure the [ServiceName](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.serviceinstaller.servicename) property is set to **MyNewService**.
3. 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.
4. Set the [DisplayName](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.serviceinstaller.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](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.serviceinstaller.servicename) property, which is the name used by the system (for example, when you use the net start command to start your service).
5. Set the [StartType](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.serviceinstaller.starttype) property to [Automatic](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicestartmode#System_ServiceProcess_ServiceStartMode_Automatic).
6. In the designer, choose **serviceProcessInstaller1** for a Visual C# project, or **ServiceProcessInstaller1** for a Visual Basic project. Set the [Account](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.serviceprocessinstaller.account) property to [LocalSystem](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.serviceaccount" \l "System_ServiceProcess_ServiceAccount_LocalSystem). This will cause the service to be installed and to run on a local service account.

Important

The [LocalSystem](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.serviceaccount" \l "System_ServiceProcess_ServiceAccount_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](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.serviceaccount" \l "System_ServiceProcess_ServiceAccount_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](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.serviceaccount" \l "System_ServiceProcess_ServiceAccount_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](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/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](https://docs.microsoft.com/en-us/dotnet/api/system.configuration.configurationmanager.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#Copy

static void Main(string[] args)

{

ServiceBase[] ServicesToRun = new ServiceBase[] { new MyNewService(args) };

ServiceBase.Run(ServicesToRun);

}

1. Change the MyNewService constructor as follows:

C#Copy

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.

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

C#Copy

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:

Copy

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](https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.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](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/how-to-install-and-uninstall-services).

1

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.

1. In the **Services** window, open the shortcut menu for your service, and then choose **Start**.
2. Open the shortcut menu for the service, and then choose **Stop**.
3. (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.

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:

Copy

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](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/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](https://docs.microsoft.com/en-us/visualstudio/deployment/installshield-limited-edition). You can also use the [Windows Installer XML Toolset](http://go.microsoft.com/fwlink/?LinkId=249067) to create an installer for a Windows service.

You might explore the use of a [ServiceController](https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.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](https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.eventloginstaller)reference page.

See Also

[Windows Service Applications](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/index)  
[Introduction to Windows Service Applications](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/introduction-to-windows-service-applications)  
[How to: Debug Windows Service Applications](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/how-to-debug-windows-service-applications)  
[Services (Windows)](http://msdn.microsoft.com/library/windows/desktop/ms685141.aspx)

Note

The feedback system for this content will be changing soon. Old comments will not be carried over. If content within a comment thread is important to you, please save a copy. For more information on the upcoming change, [we invite you to read our blog post](https://docs.microsoft.com/teamblog/a-new-feedback-system-is-coming-to-docs).

**24 Comments**

Sign in

**99 people listening**

[mairacw avatarAlbertHerd avatar](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

+ Follow

Post comment as...

[Avatar for Daniel_Wells](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Daniel\_Wells 9 days ago

Starting the service returns the error 1053 due to timeout immediately. The only reference to this issue I've found relates to a hotfix for Server 2003. I've tried this on 2008 and 2012 servers and my own Windows 10 machine. I've removed all my added code with the same result. Any suggestions?

Update: Removing all references to the Event Log works. That is not ideal of course. Any thoughts?

LikeReply

[Avatar for Eduard_Florescu](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Eduard\_Florescu Jan 13, 2018

Hi! I would like to know if there's any way to add an installer to a service that I already have installed on my computer. Thanks in advance.

LikeReply

[Avatar for Akshay_Kokil](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Akshay\_Kokil Dec 14, 2017

Very comprehensive walkthrough. Thank You!

LikeReply

LarGriff Nov 4, 2017

An exception occurred during the Install phase.  
System.Security.SecurityException: The source was not found, but some or all event logs could not be searched.  Inaccessible logs: Security.

LikeReply

[Avatar for Mike_Dean](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Mike\_Dean Dec 11, 2017

@LarGriff I'm having the same problem. Any luck?

LikeReply

[Avatar for Akshay_Kokil](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Akshay\_Kokil Dec 14, 2017

@Mike\_Dean @LarGriff Install service from Administrator acoount command prompt.

LikeReply

[Avatar for baio77](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Baio77 Sep 21, 2017

Hello. I don't have the Template > Visual C# > Windows Desktp > Windows Service.

How can get it?

LikeReply

Thura\_Soe Sep 24, 2017

@baio77  Just type service in search box.

1LikeReply

[Avatar for Uyen_Luu_Dinh](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Uyen\_Luu\_Dinh Sep 6, 2017

Please tell me know what is eventId? I don't see that it defined in this project.

LikeReply

MichaelMccormick Sep 6, 2017

@Uyen\_Luu\_Dinh  For me, using Visual Studio 2012, when you are viewing the code for MyNewService.cs, you can find eventID in the drop-down list at the upper right of the code screen. When you select it, it will open a new code window for MyNewService.Designer.cs with the declaration of eventID.

LikeReply

Scott\_Winterrowd Aug 31, 2017

This was a good introduction to building a windows service for me. Which I had never done. Please check my sidenotes to clarify the article further. Additionally, I got a cryptic error about a security violation when using the install utility at the command prompt. This was because I failed to run the command prompt as an "Administrator". In addition, upon executing the uninstall, the event log file "MyEventLog1" was still visible in the "Applications and Service Logs" folder. After some research, I figured out how to delete it by running the command "Remove-EventLog -LogName "MyEventLog1""  using Powershell as an Administrator. Overall an informative article but needs a little more work in my opinion.

LikeReply

[Avatar for JanineMoore](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

JanineMoore Aug 10, 2017

So I'm easily confused.  #4 Add a declaration for the eventLog object in the MyNewService class, right after the line that declares the components variable:  The class doesn't have a components declaration so I'm unsure where I'm suppose to insert the next part.

1LikeReply

MichaelMccormick Aug 18, 2017

@JanineMoore  I was stuck here as well. In the Solution Explorer I expanded MyNewService under MyNewService.cs. Below that I saw a list of objects including components : IContainer. I right-clicked that object and chose Is Used By from the contextual menu. There were three references to MyNewServiceDesigner.cs, so I double clicked that to open the code. There I found the components declaration I believe that step #4 is referencing.

However, at the bottom of the code I found, inside of the code for InitializeComponent(), that there was already a declaration for eventLog1. So I left the code unchanged for now.

For reference, I am using Visual Studio 2012 and have .NET 4.5 installed.

LikeReply

[Avatar for JimKay](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

JimKay Dec 8, 2017

@JanineMoore I believe this article was written for an earlier version of the Service Template, where you had to add these variables. Later versions must have included them in some "code-behind" that is not immediately visible.

If you open the file MyNewService.Designer.cs the following lines appear around line number 5:

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

Sadly, this Walkthrough needs a technical review.

LikeReply

James\_MA Aug 8, 2017

The following error is prompted when trying to build the project:

SeverityCodeDescriptionProjectFileLineSuppression State

ErrorCS0103The name 'eventId' does not exist in the current contextMyNewService

It can be solved by adding the following code in MyNewService class.  
private int eventId = 0;  
  
But I don't know if the eventId should be initialized with any values under the services.

LikeReply

Brian\_thow Nov 1, 2017

@James\_MA I thought the same thing at first, but i found that the event Id is arbitrary.

what that means is that the id you use for an event should mean something to you. it can be whatever you like. maybe Monitoring the System is ID 1234 and on stop is 374. it's for readability. you could always declare consts for the IDs,

ex: private const int onstopID = 1234; and use the variable in your code instead

sources:

[**https://stackoverflow.com/questions/3970220/window-services-event-id**](https://stackoverflow.com/questions/3970220/window-services-event-id)  
[**https://support.microsoft.com/en-us/help/307024/how-to-write-to-an-event-log-by-using-visual-c**](https://support.microsoft.com/en-us/help/307024/how-to-write-to-an-event-log-by-using-visual-c)

LikeReply

[Avatar for sinanaltaii](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Sinanaltaii Jul 15, 2017

Can i download this sample application? would be nice to have the sample and compare it to my failing code

1LikeReply

[Avatar for Rasmus_Tilljander](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Rasmus\_Tilljander Jul 4, 2017

At section "Adding Installers to the Service" number 9 there are some inconsistency and I'm unsure how to interpret the text. The text tells the reader to use "Service Account" and then proceeds to warn about the rights of a "Service Account" and finally recommends the user to instead use another account type, which is "Service Account". I cannot wrap my head to understand this logic and I assume that at least one of the three statements are incorrect. I might be wrong, however, could someone clarify this a bit? Many thanks

1LikeReply

[Avatar for jwliechty](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Jwliechty Jul 6, 2017

@Rasmus\_Tilljander I think the document is rendering incorrectly. The "ServiceAccount" is the enum type. My guess is that it at one time it stated \*which\* enum value to use, but something happened, and no longer does. I just set mine to "LocalSystem".

LikeReply

[Avatar for mairacw](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Mairacw moderatorJul 6, 2017

@jwliechty @Rasmus\_Tilljander we had some migration issues that I thought were completely solved by now, but clearly aren't. I'll fix this. And yes, it should say "LocalSystem". Thanks for bringing this to our attention and I apologize for the inconvenience.

LikeReply

[Avatar for mairacw](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Mairacw moderatorJul 6, 2017

@jwliechty @Rasmus\_Tilljander You can see the PR for the fix here: [**https://github.com/dotnet/docs/pull/2559**](https://github.com/dotnet/docs/pull/2559)

LikeReply

[Avatar for DovMiller](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

DovMiller Jul 3, 2017

How do I navigate to the folder that contains my project's output in the Command Prompt window?

LikeReply

Robbie\_Faulks Jul 5, 2017

@DovMiller the path will by default be in your %project%\bin\Debug or %project%\bin\Release folders. Standard cmd prompt navigation should get you there e.g. cd c:\mycode\myproject\bin\Debug

Then run the command as defined in the walkthrough

LikeReply

[Avatar for Sebastian_Scholle](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)

Sebastian\_Scholle Jun 26, 2017

Getting a build error on  "new System.Timers.ElapsedEventHandler(this.OnTimer)" - "No overload 'OnTimer' matches delegate 'EleapsedEventHandler'"

LikeReply

* [Feedback](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer#comments-container)
* [Edit](https://github.com/dotnet/docs/blob/master/docs/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer.md)
* [Share](https://docs.microsoft.com/en-us/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer)
* Language    
* Theme    