Visual Studio 2010

**Introduction to Windows Service Applications**

Microsoft Windows services, formerly known as NT services, enable you to create long-running executable applications that run in their own Windows sessions. These services can be automatically started when the computer boots, can be paused and restarted, and do not show any user interface. These features make services ideal for use on a server or whenever you need long-running functionality that does not interfere with other users who are working on the same computer. You can also run services in the security context of a specific user account that is different from the logged-on user or the default computer account. For more information about services and Windows sessions, see the Platform SDK documentation in the MSDN Library.

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

You can easily create services by creating an application that is installed as a service. For example, suppose you want to monitor performance counter data and react to threshold values. You could write a Windows Service application that listens to the performance counter data, deploy the application, and begin collecting and analyzing data.

You create your service as a Microsoft Visual Studio project, defining code within it that controls what commands can be sent to the service and what actions should be taken when those commands are received. Commands that can be sent to a service include starting, pausing, resuming, and stopping the service; you can also execute custom commands.

After you create and build the application, you can install it by running the command-line utility InstallUtil.exe and passing the path to the service's executable file, or by using Visual Studio's deployment features. You can then use the **Services Control Manager** to start, stop, pause, resume, and configure your service. You can also accomplish many of these same tasks in the **Services** node in **Server Explorer** or by using the [ServiceController](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.aspx) class.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifService Applications vs. Other Visual Studio Applications

Service applications function differently from many other project types in several ways:

* The compiled executable file that a service application project creates must be installed on the server before the project can function in a meaningful way. You cannot debug or run a service application by pressing F5 or F11; you cannot immediately run a service or step into its code. Instead, you must install and start your service, and then attach a debugger to the service's process. For more information, see [How to: Debug Windows Service Applications](http://msdn.microsoft.com/en-us/library/7a50syb3.aspx).
* Unlike some types of projects, you must create installation components for service applications. The installation components install and register the service on the server and create an entry for your service with the Windows **Services Control Manager**. For more information, see [How to: Add Installers to Your Service Application](http://msdn.microsoft.com/en-us/library/ddhy0byf.aspx).
* The Main method for your service application must issue the Run command for the services your project contains. The Run method loads the services into the **Services Control Manager** on the appropriate server. If you use the **Windows Services** project template, this method is written for you automatically. Note that loading a service is not the same thing as starting the service. See "Service Lifetime" below for more information.
* Windows Service applications run in a different window station than the interactive station of the logged-on user. A window station is a secure object that contains a Clipboard, a set of global atoms, and a group of desktop objects. Because the station of the Windows service is not an interactive station, dialog boxes raised from within a Windows service application will not be seen and may cause your program to stop responding. Similarly, error messages should be logged in the Windows event log rather than raised in the user interface.

The Windows service classes supported by the .NET Framework do not support interaction with interactive stations, that is, the logged-on user. The .NET Framework also does not include classes that represent stations and desktops. If your Windows service must interact with other stations, you will need to access the unmanaged Windows API. For more information, see the Platform SDK documentation.

The interaction of the Windows service with the user or other stations must be carefully designed to include scenarios such as there being no logged on user, or the user having an unexpected set of desktop objects. In some cases, it may be more appropriate to write a Windows application that runs under the control of the user.

* Windows service applications run in their own security context and are started before the user logs into the Windows computer on which they are installed. You should plan carefully what user account to run the service within; a service running under the system account has more permissions and privileges than a user account.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifService Lifetime

A service goes through several internal states in its lifetime. First, the service is installed onto the system on which it will run. This process executes the installers for the service project and loads the service into the **Services Control Manager** for that computer. The **Services Control Manager** is the central utility provided by Windows to administer services.

After the service has been loaded, it must be started. Starting the service allows it to begin functioning. You can start a service from the **Services Control Manager**, from **Server Explorer**, or from code by calling the [Start](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.start.aspx) method. The [Start](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.start.aspx) method passes processing to the application's [OnStart](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstart.aspx) method and processes any code you have defined there.

A running service can exist in this state indefinitely until it is either stopped or paused or until the computer shuts down. A service can exist in one of three basic states: Running, Paused, or Stopped. The service can also report the state of a pending command: ContinuePending, PausePending, StartPending, or StopPending. These statuses indicate that a command has been issued, such as a command to pause a running service, but has not been carried out yet. You can query the [Status](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.status.aspx) to determine what state a service is in, or use the [WaitForStatus](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.waitforstatus.aspx) to carry out an action when any of these states occurs.

You can pause, stop, or resume a service from the **Services Control Manager**, from **Server Explorer**, or by calling methods in code. Each of these actions can call an associated procedure in the service ([OnStop](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onstop.aspx), [OnPause](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.onpause.aspx), or [OnContinue](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.oncontinue.aspx)), in which you can define additional processing to be performed when the service changes state.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifTypes of Services

There are two types of services you can create in Visual Studio using the .NET Framework. Services that are the only service in a process are assigned the type Win32OwnProcess. Services that share a process with another service are assigned the type Win32ShareProcess. You can retrieve the service type by querying the [ServiceType](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.servicetype.aspx) property.

You might occasionally see other service types if you query existing services that were not created in Visual Studio. For more information on these, see the [ServiceType](http://msdn.microsoft.com/en-us/library/tfdtdw0e.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifServices and the ServiceController Component

The [ServiceController](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.aspx) component is used to connect to an installed service and manipulate its state; using a [ServiceController](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.aspx) component, you can start and stop a service, pause and continue its functioning, and send custom commands to a service. However, you do not need to use a [ServiceController](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.aspx) component when you create a service application. In fact, in most cases your [ServiceController](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.aspx) component should exist in a separate application from the Windows service application that defines your service.

For more information, see [ServiceController](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicecontroller.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifDeploying and Installing Services

Visual Studio ships installation components that can install resources associated with your service applications. Installation components register an individual service on the system to which it is being installed and let the **Services Control Manager** know that the service exists.

After you add installers to your application, the next step is to create a setup project that will install the compiled project files and run the installers needed to install your service. To create a complete setup project, you must add the service project's output to the setup project and then add a custom action to have your service installed. For more information on setup projects, see [Setup and Deployment Projects](http://msdn.microsoft.com/en-us/library/wx3b589t.aspx). For more information on custom actions, see [Walkthrough: Creating a Custom Action](http://msdn.microsoft.com/en-us/library/d9k65z2d.aspx).

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

* Services must be created in a **Windows Service** application project or another .NET Framework–enabled project that creates an .exe file when built and inherits from the [ServiceBase](http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.aspx) class.
* Projects containing Windows services must have installation components for the project and its services. This can be easily accomplished from the **Properties** window. For more information, see [How to: Add Installers to Your Service Application](http://msdn.microsoft.com/en-us/library/ddhy0byf.aspx).