

Creating a Windows Service in C#

By John Andre (john@montgomerysoftware.com)

17 September 2004

As [written earlier](#), you could write a Windows (NT) service in Visual Basic 6 but it is prone to problems. Now, with C# and VB.Net it is simple to create a Windows service without those in-built limitations. That's right. No more need for a form (to host the required OCX). No more threading problems. The only surprise is that when you create a Windows service in .Net, the project is missing something very crucial - the InstallerClass.

The InstallerClass is a short, simple class that will prevent your service from working if it is missing. This article will show you what you need to do to make sure everything works properly.

First, let's get the basics of the service setup. When you create a new Windows service project, that project will include a class called **Service1.cs**. That class will have routines to handle when the Service Control Manager (SCM) starts the service and when the SCM stops the service (OnStart and OnStop, respectively).

Fill in the startup logic (open database connections, etc.) in the OnStart routine. Put your shutdown logic (close database connections, etc.) in the OnStop routine. If your service should do something at a regular interval (for example, check to see if there are any new records in a certain table), then you might want to add a timer to the service class (just drag and drop it from the toolbox in design view). Then, in the timer1_Elapsed event, add the logic that your service is being built to perform.

Now, you have to set a couple of properties on the service class (you can do this in the property window in design view):

- Name = can be anything you like, but like all control names in .Net, don't use spaces
- ServiceName = the name you would like to be displayed in the Services control panel and in the Event Log (can have spaces)

Logging Errors

Troubleshooting Windows services can be tricky, so it is a good idea to add event logging to your error handlers. Something like this should be adequate:

```
try
{
// your logic here
}
catch(Exception ex)
{
EventLog.WriteEntry(this.ServiceName,"Error: "+ ex.ToString(),
System.Diagnostics.EventLogEntryType.Error);
}
```

InstallerClass

Now, you're ready to add the missing Installer class. Just add a blank class file and

copy and past the code below.

You might be wondering why Microsoft doesn't automatically add this class when you create a Windows service project. I wonder it as well. However, if you don't add this class your service will simply not work. So, better to keep this code handy, you'll need it every time you create a new service.

```
using System.ComponentModel;
using System.ServiceProcess;

namespace YourNamespaceHere
{
    [RunInstallerAttribute(true)]
    public class InstallerClass: System.Configuration.Install.Installer
    {
        public InstallerClass()
        {
            ServiceInstaller si = new ServiceInstaller();
            ServiceProcessInstaller spi = new ServiceProcessInstaller();

            si.ServiceName = "DBNotifier";
            si.DisplayName = "DB Notifier";
            this.Installers.Add(si);

            spi.Account = System.ServiceProcess.ServiceAccount.LocalSystem;
            spi.Username=null;
            spi.Password=null;
            this.Installers.Add(spi);
        }
    }
}
```

Potential Problems

There are a few potential problems when implementing a Windows service in .Net. Here are some issues to watch out for:

Dependencies - If your service is dependent on other services, you need to make sure this dependency information is set properly in the registry. **Warning: improperly setting service dependency information in the registry could make a computer unbootable due to issues like circular dependencies - Service A cannot start before Service B and Service B cannot start before Service A - resulting in neither service ever starting. Be very careful here.**

Using RegEdt32 (RegEdit will not work here due to it's inability to add REG_MULTI_SZ types), go to HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\[YOUR SERVICE] and create a new value with a name of **DependOnService** and a data type of **REG_MULTI_SZ**. Set the data to the name of the key of the service your Windows service depends on. For example, if your Windows service depends on **MSMQ** the enter MSMQ since MSMQ is the name of the key in this services branch. If your Windows service depends on multiple services, add each service on its own line. If you are creating distribution media for your Windows service, you can include a .reg file which includes this information (easing deployment efforts).

Location of EXE - It is important that the executable you created for your Windows service be located on a local drive to the machine on which the service will run. If the executable is on a remote drive, the service will not run throwing **Error 5: Access is denied**.

Set User Account - Usually you will set the user account of your new Windows service after it is installed. However, you do have the option of setting the user account and password properties in the InstallerClass. If you do not set this information, the service will be set to use the Local System account. Make sure that the account you choose has the appropriate security permissions to perform the tasks that you are asking the service to perform. All processing, including calls out to ActiveX EXEs, will use the security context from your service.

Settings - If you will use the registry to store settings for your service, then you should use Microsoft.Win32.Registry.LocalMachine as opposed to Microsoft.Win32.Registry.CurrentUser, which might not work since there may be no user logged onto the computer where your service runs).

Installation

Windows Services need to be installed. I am not referring to setup media here but rather installing into the Service Control Manager (SCM). To install a service into the SCM you need to use InstallUtil.exe. This is quite simple. Just locate the InstallUtil.exe under your framework directory (e.g., C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322) and pass to it the full path to your new service (e.g., C:\DBNotifier\DBNotifier.exe). You can pass -u to uninstall the service from the SCM. You will want to do this from a command prompt in case you want to read the output from InstallUtil.exe.

Summary

Writing a Windows service in C# (or any .Net language) is much easier than it ever was before. Yes, you need to remember to include the missing InstallerClass but once you do that, it should be clear sailing.