**Introduction**

While we were developing our Corporate Portal, I was asked for a functionality, which congratulates the birthday of our stuff sending an e-mail message. There was a congratulating message doing the same task at the main page of Intraweb but it was only for whom opens that. But we wanted that when they open their mailbox, they see a congratulating message of their own birthdays.   
There fore I realised that this cannot be accomplished with a web application and I decided to create a **windows** **Service** which will run on the server and search once in a day the stuffs birthdays and then send them the message if there is(are) a match(any matches.)  
Shorly, the task is to search database for the stuff table and if there is(are) birthday(s) matching on the day, send them a congratulation message (e-mail). I thought that it could be achieved best using a **windows** **service**.

The basics of **Windows** **service**s are given in this article. If you feel your knowledge enough, you can just skip this one and step on next article here. [**Windows Services in Action II**](http://www.codeproject.com/KB/cs/WindowsServicesInAction2.aspx).

**Background**

Some tasks can be solved with only **windows** **service**s effectively.So every developer should decide when to use them and how to use them. I have searched about **windows** **service**s because i needed in numerous cases. After all code work i decided to share it with other coders.

**Windows Services in .NET**

Firstly I will try to give you basic information about **Windows** **Service**s in .NET and show how we can use in a real-world task. Steps:

1. What are **windows** **service**s?

2. The Architecture

3. The Methods

4. The Components

5. Create a **windows** **service** application

6. Installation process

7. Monitoring and administration

**1. What are windows services?**

**Windows** **service**s are used to create long-running executable applications that run in their own **Windows** sessions in the background. **Windows** **service**s do not have a user interface because they are not meant to interact with users. They can be configured to start automatically when the computer boots, we can start them manually or some start when needed. They can be started, paused, restarted or stopped using the **Service Control Manager** which is the central utility provided to control them.   
They must be installed to system to run in a normal manner. We cannot run or debug Window **service**s without installation.We need a special component to install them or take special steps to have them ready to run.**Windows** **service** applications run in a different window station than the interactive station of the logged-on user. Because the station of the **Windows** **service** is not an interactive station they should be logged in the **Windows** event log instead of using a user interface.   
**Windows** **service** applications run in their own security context and are started before any users log on into the computer which they are installed.Some examples to **Windows** **service** applications:   
Network Connections, Print Spooler, Net Logon…You can see them typing **services.msc** on the Run menu item of **Windows** Start Menu if you have an NT Based OS like **Windows** 2000, XP, 2003 Server…

**2. The Architecture of Windows services in .NET**

* **ServiceBase Class** To create a new **service** class, it inherited from SerciveBase class. The methods of the class can be overridden to change their functionality if needed.
* **ServiceProcessInstaller**   
  A **windows** **service** must be represented using **Service**ProcessInstaller class in obtain to communicate and control the **service**.
* **ServiceInstaller**   
  A **windows** **service** must be expanded using **Service**Installer class to be able to use the standart .NET installation method.
* The namespace of these classes is **System.ServiceProcess** and their assembly is System.**Service**Process. So they can be found in **system.serviceprocess.dll**.

**3. The Methods**

There are several methods exposed by the **ServiceBase** class These methods can be overridden to add custom behavior.   
  
**OnStart:** This is called when the **service** starts running. So we can override it if we need to take an action in this step.   
**OnPause:** This is called when the **service** is paused.   
**OnStop:** This is called when the **service** is stopped.   
**OnContinue:** This is called when the **service** is resumed after being paused.   
**OnShutdown:** This is called when the **service** is just prior to your system shutting down.   
**OnCustomCommand:** This is called when your **service** receives a custom command.   
**OnPowerEvent:** This is called when a power management event is received.

**4. The Components**

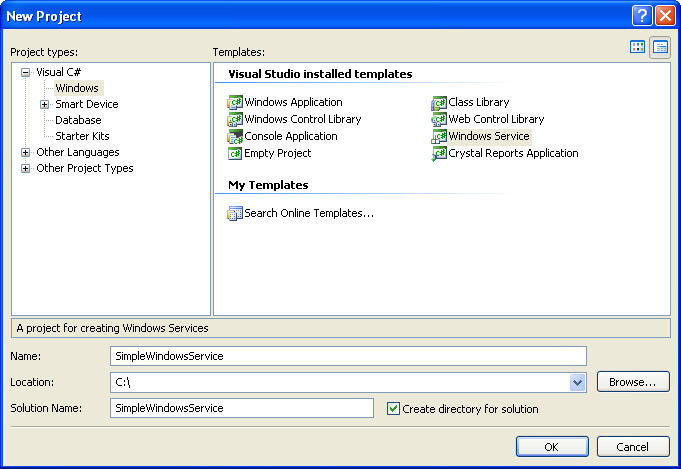
* First of all, the **service** has some properties like **CanStop**(true by default), **CanShutdown**, **CanPauseAndContinue**, **ServiceName**(**Service**1 by default)
* **EventLog** component is widely used in **windows** **service**s. Logs can be created by this component very simply. **Log** is the name of the log. **Source** property is to spesify the application name to use when writing to the eventlog. There will be some code examples related to eventlog component.
* **ServiceInstaller** and **ServiceProcessInstaller** components are necessary to make the **windows** **service** ready to use standard setup project. These two components are added automatically if we add installer to the **service**. **Service**Installer’s mostly set properties are **ServiceName** which is shown in the **services** tool and **StartType** which determines when the **service** is started. The most important property of **Service**ProcessInstaller component is account. Usually this property is set to **LocalService**.

**5. Create a windows service application**

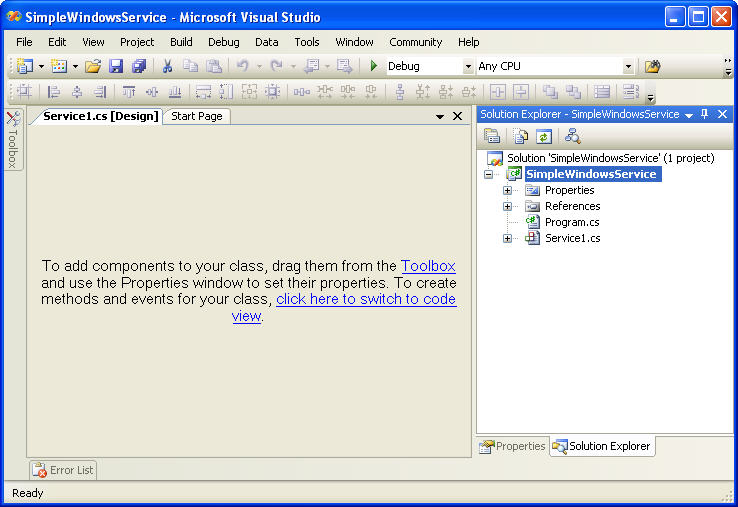
We can create a **windows** **service** writing code or using **Windows** **service** project template.

**Creating a windows service**

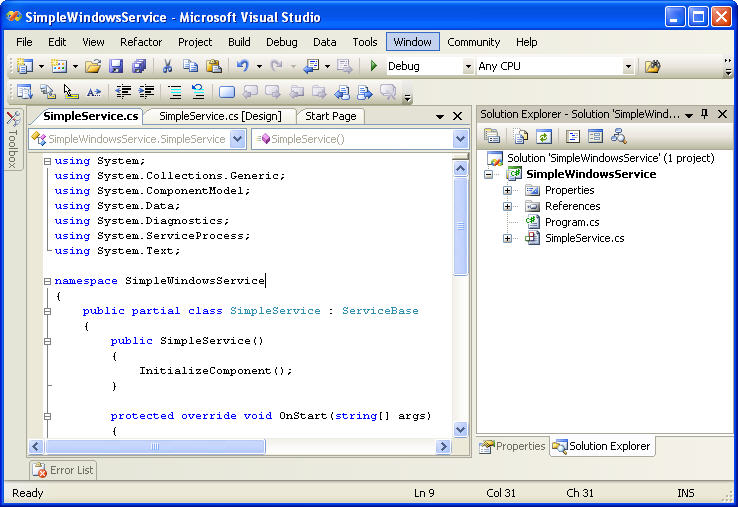
* Open Visual Studio 2005, click File, New and choose Project menu item. You will see the window below.



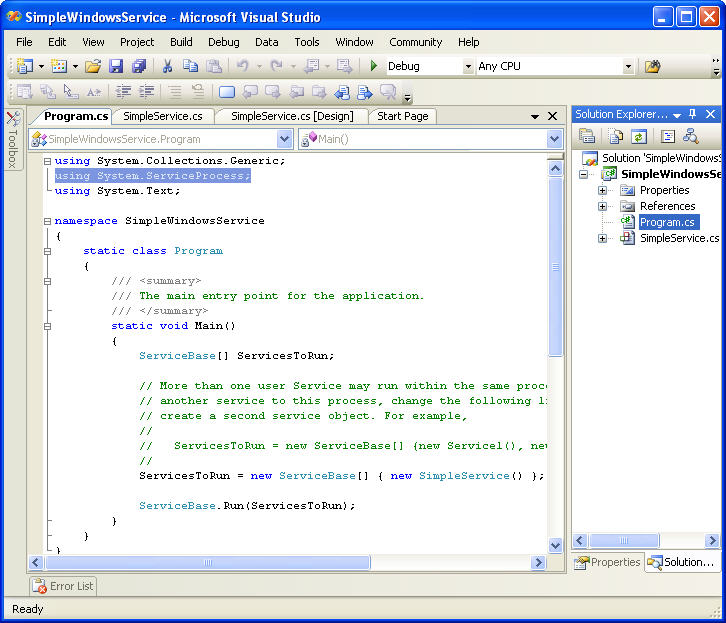
* Select **Windows** **Service** project template and rename project name as “Simple**WindowsService**” or whatever you want.
* Then VS will create all basic classes for you and you will see the environment like below.



* You can change the name of **service**1 as Simple**Service**. If you do this, a small window will apear and ask you whether you change related refences. Click yes.
* If you double click the workspace of Simple**Service**[Design], you can see the code page where you can work when necessary. Illustrated below.



* As you see, most of the codes and classes created automatically. Program.cs is where the **service** created and run. It uses **Service**Process. So you can see using System.**Service**Process; code at the top.
* Usually you don’t have to anything here.

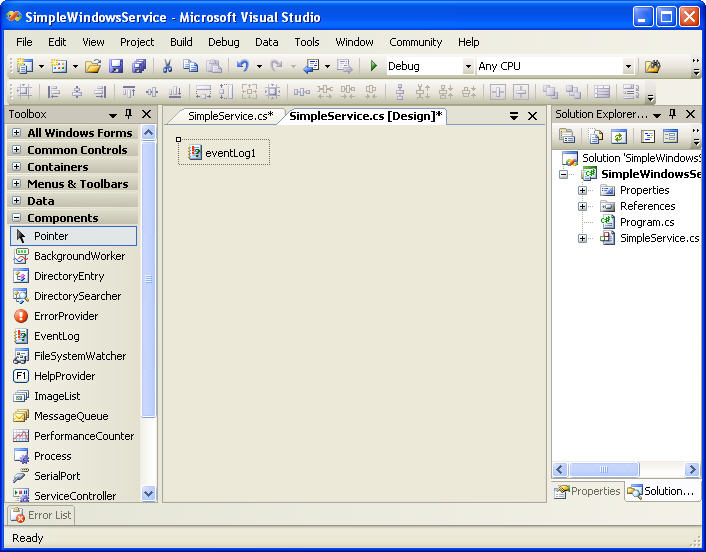


Now, our Simplew**WindowsService** is ready to go. You can build it (by pressing F6) But we cannot run it because it is not installed. And even it is installed it doesn’t have any functionlities except starting, stoping.

Therefore we will add some simple functonalities to our Simple**Service**.

**Adding some functionalities to a Windows Service**

* Open Simple**Service**.cs in Design mode.   
  The is task is to write into a log when the **service** starts and stops so that we will be simply informed using **Windows** standard Eventviewer.
* When in Design mode of Simple**Service**.cs Drop an eventLog component from the Components tab in the Toolbox to design area.   
  (To switch to Design mode press Shift+F7 and to switch to code view press F7)



* Rename eventLog1 component’s name as eventLogSimple · The eventlog component must be initialized in Simple**Service** method which is already created automatically in Simple**Service**.cs. After we modified the code, it will look like below:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/KB/cs/WindowsServicesInAction1.aspx)

public Simple**Service**()

{

InitializeComponent();

*// Initialize eventLogSimple*

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

System.Diagnostics.EventLog.CreateEventSource("SimpleSource","SimpleLog");

eventLogSimple.Source= "SimpleSource";

eventLogSimple.Log = "SimpleLog";

}

* Add the line below to OnStart Method which is already created.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/KB/cs/WindowsServicesInAction1.aspx)

eventLogSimple.WriteEntry("Hello world from Simple **Service**!");

* Add the line below to OnStart Method which is already created also.

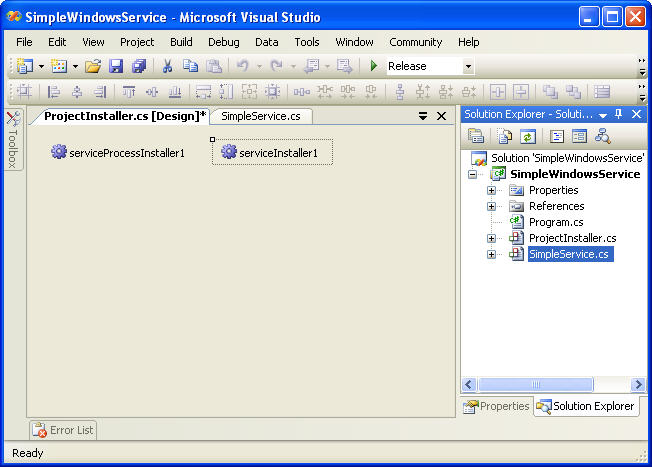
http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/KB/cs/WindowsServicesInAction1.aspx)

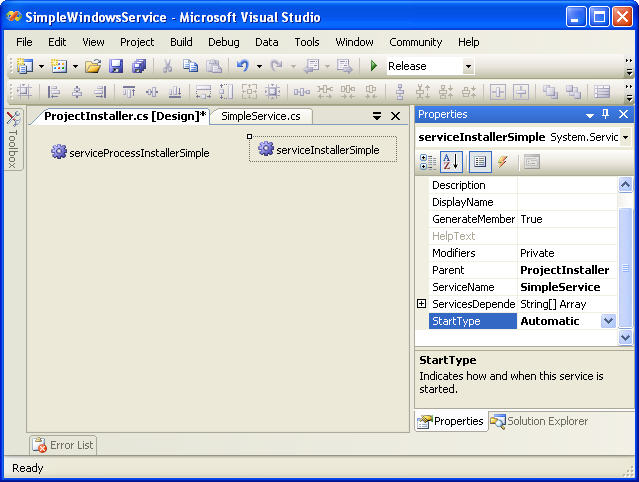
eventLogSimple.WriteEntry("Simple **Service** stopped!");

* Press F6 to build our **windows** **service** if you see “Build succeded” message at the bottom on the satus bar, you are done. Now, our **windows** **service** is ready to be installed.

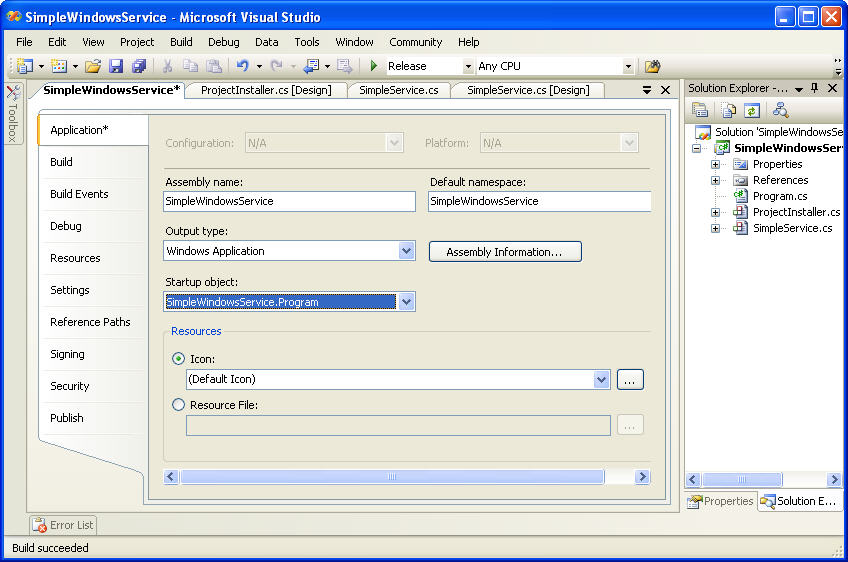
**6. Installation process**

**Install the Windows Service**   
The easiest way of installing a **windows** **service** is using the installer components and the Setup Project of Visual Studio 2005.

* Right-click Simple**Service**.cs in Solution Explorer and select View Designer.
* Right-click the design area and select Add Installer. Then ProjectInstaller.cs page will appear in design view. And there will be replaced two components in it: **service**ProcessInstaller1 and **service**Installer1 like below:   
    
  
* Rename them as **service**ProcessInstallerSimple and **service**InstallerSimple respectively. · Find the **Service**Name property of **service**InstallerSimple in Properties **windows**.
* Rename its value as Simple**Service**. It is **Service**1(default value). The **service** will be presented by this value. So it is better to set to a meaningful name.
* Another important property of **service**InstallerSimple component is StartType. It has three values: Manual, Automatic and Disabled. It is set to Manual by default. This means we must start the **Service** manually. Let’s set the property to Automatic so that it starts when **windows** starts.



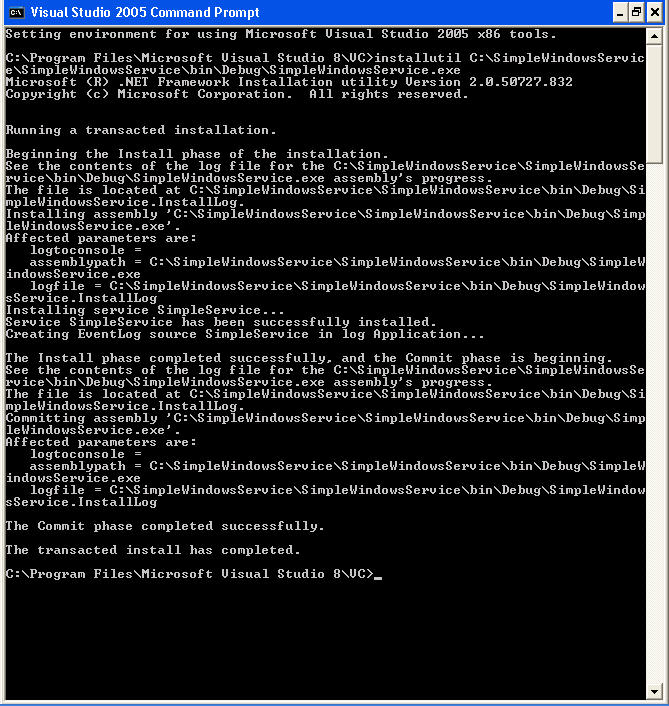
* After that, select **service**ProcessInstallerSimple in the designer and set the Account Property to Local**Service**. Therefore the **service** will be installed and run on a local account.
* In Solution explorer, select Simple**WindowsService** project, right-click it and select Properties.
* The Property Designer of the project will appear. Chose Simple**WindowsService**.Program from the Startup object on Application page.



* In Solution explorer, select Solution “Simple**WindowsService**” and right-click and select Build Solution. If everything went OK, you see “Build succeded” message at the bottom.

The **service** is now can be installed. There are two installation methods:

* **Windows** **service** can be on your system using InstallUtil.exe. The utility must be run on the Visual Studio 2005 Command Prompt (Usually found in "C:\Program Files\Microsoft Visual Studio 8\VC\ ". For instance, C:\Program Files\Microsoft Visual Studio 8\VC\installutil C:\Simple**WindowsService**\Simple**WindowsService**\bin\Debug\ Simple**WindowsService**.exe in our case.

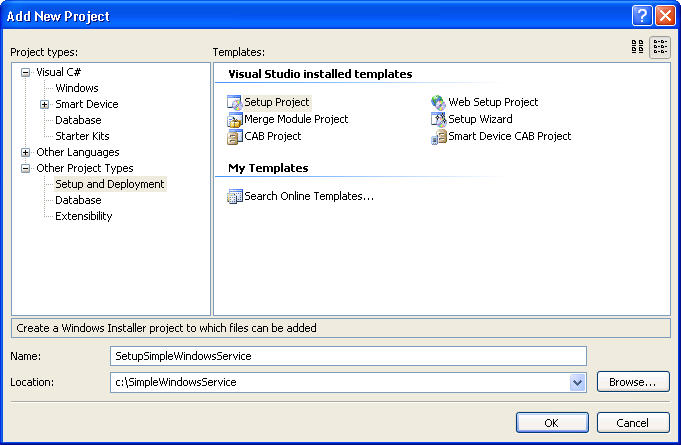


And to uninstall the **service** the utility must be used with –u parameter: C:\Program Files\Microsoft Visual Studio 8\VC\installutil -u C:\Simple**WindowsService**\Simple**WindowsService**\bin\Debug\ Simple**WindowsService**.exe

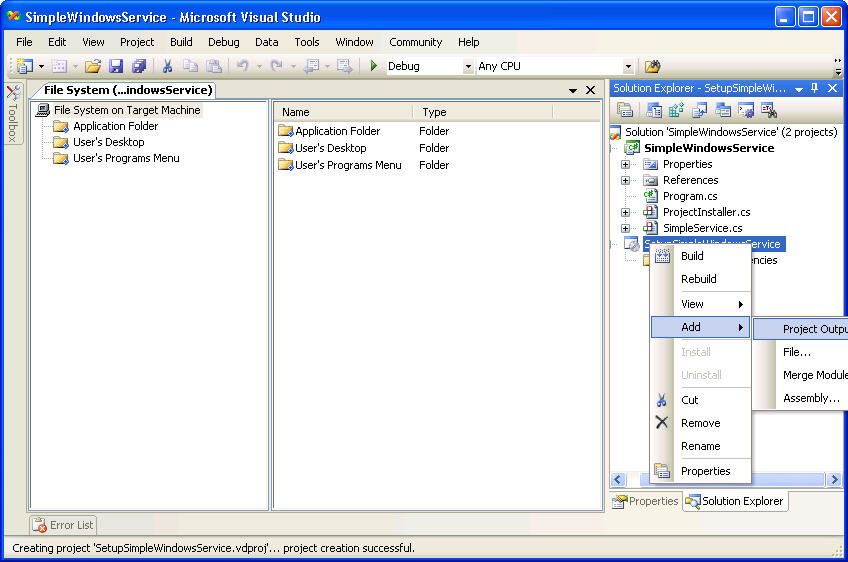
* Another method is to add a setup project to the solution and use the result package to install and uninstall the **windows** **service**. In this case, the Add or Remove programs tool in the Control Panel of the **windows** system can be used to uninstall it.

**Adding a setup project to our solution**

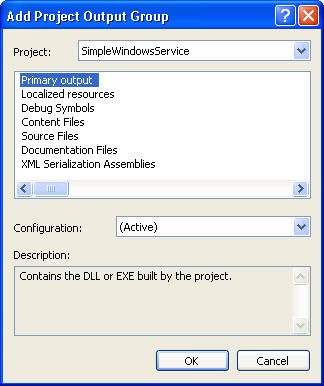
* Right-click Simple**WindowsService** solution in Solution Explorer, select Add, New Project and select Deployment project. Give a meaningful title to the project as seen below.



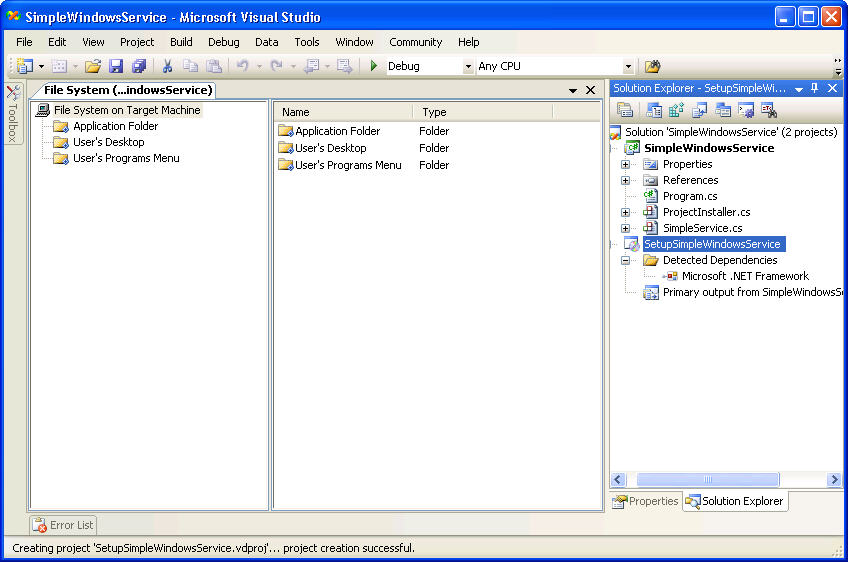
* Right-click SetupSimple**WindowsService** In Solution Explorer, select add, then choose Project Output.



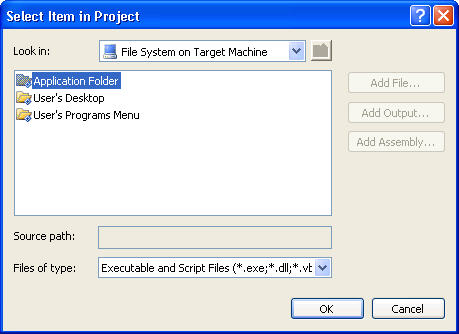
* From the list box, select Primary Output, and click OK.



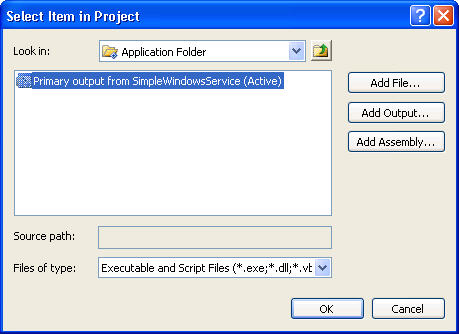
* To add a custom action right-click the setup project In Solution Explorer, select View, and then click Custom Actions.



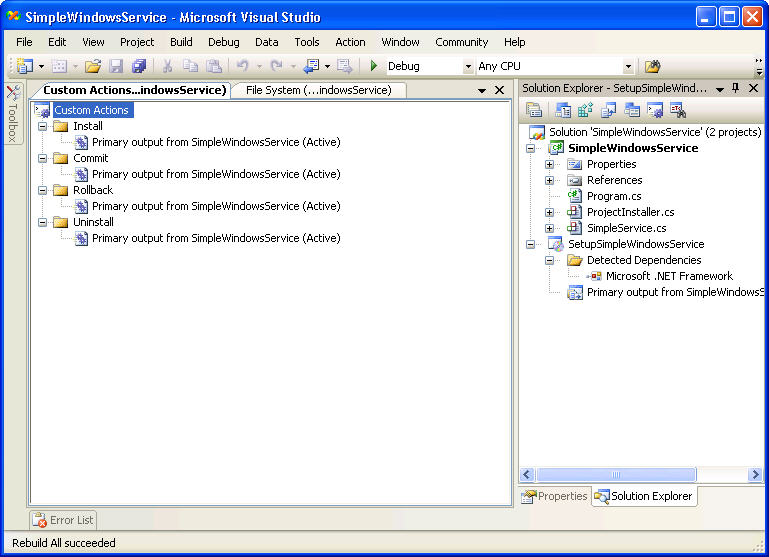
* Right-click the Custom Actions node and choose Add Custom Action.



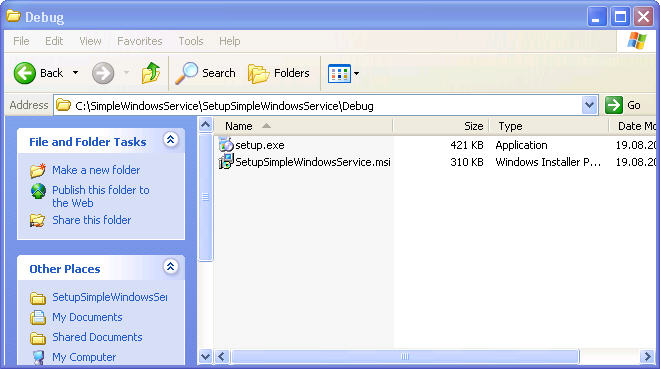
* Double-click the Application Folder in the list box to open it, select Primary Output from Simple**WindowsService** (Active), and click OK.



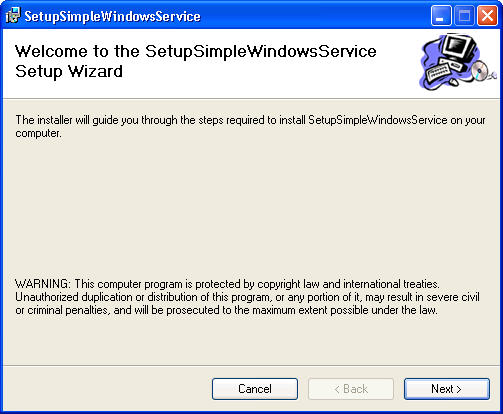
* The primary output is added to all four nodes of the custom actions — Install, Commit, Rollback, and Uninstall.



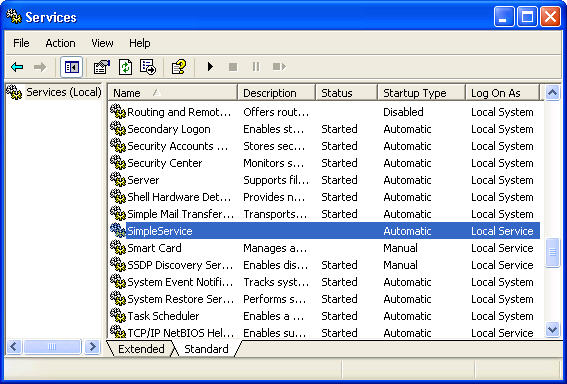
* In Solution Explorer, right-click the Simple**WindowsService**Setup project and click Build. After building the solution there can be found setup project SetupSimple**WindowsService**.msi in the setup folder. (C:\Simple**WindowsService**\SetupSimple**WindowsService**\Debug in our case.)



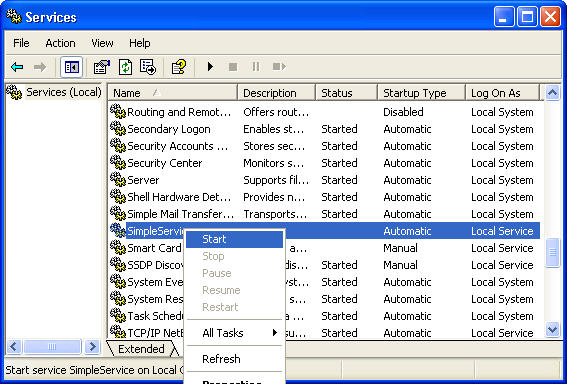
* It can be double-clicked to run the setup program and it is used to install, uninstall, commit and rollback. Finally the Setup Wizard of our solution be appeared and followed familiar setup steps.
* Or the setup project can be started by right-clicking the setup project in Solution Explorer and selecting Install.



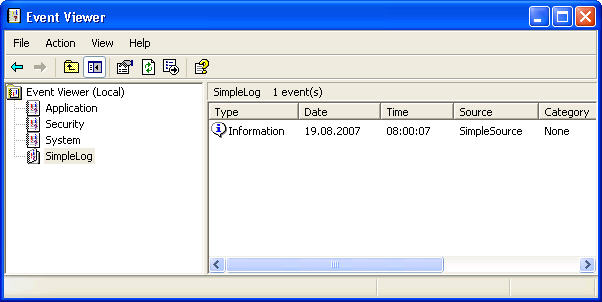
* After finished the installation, it can be seen in **Service**s on our **Windows** System.
* To view **Service**s, from **Control Panel** choose **Administrative Tools** and click **Services**.



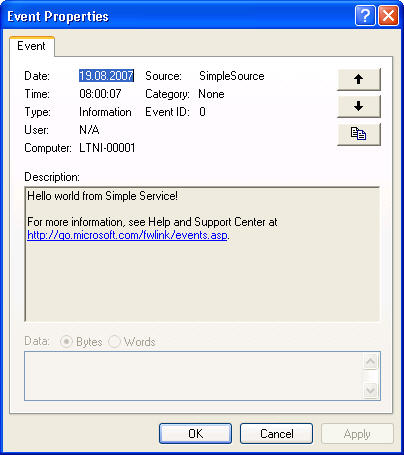
* To start the **service** right-click on it and select **Start**.



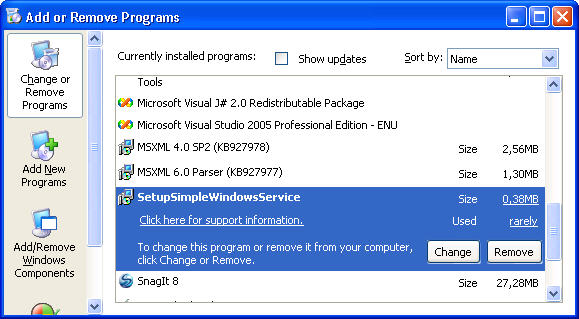
* To view the logs of our **windows** **service** click **Event Viewer** in **Administrative Tools**. You can see SimpleLog item in the treeview at the left which is created by our **service** and this item is where our **service**s logs are kept. If you started the **service**, you can see an event log at the right.



* Double-click on it to open then you can see the message "Hello world from Simple **Service**!" here.



* To uninstall it you can use **Add or Remove Programs** in **Control Panel**.



Ofcourse you can use the Setup Project to uninstall the **service** but be careful to use the same Build which is used for installation. If you built solution, installed it and then rebuilded it again the Setup Project changes and cannot uninstall the **service**. In this situation you can keep the Setup file which is used for installation or just use **Add or Remove Programs** tool.

**7. Monitoring and administration**

We said that **windows** **service**s don’t need a user interface. Therefore we have to manage them using other tools. There are three ways:   
1. Microsoft **Service**s : The **Service**s tool in Administrative Tools is also used to monitor and manage the **service**s.   
2. The console: **Service**s can be managed by using the command prompt: The line below is used to start the **service**:  
C:\net start Simple**WindowsService**  
3. Yes, we can manage them programmatically. I think this is out of the scope of this article.

**Conclusion**

I think, the solution is simple but of course should be improved to overcome your real-time problems. I am planning to improve my solution in the future and expect your help. I am planning to write another article to show you how a **windows** **service** can be used.