Windows Services

Why do we need a Windows Service?

Whenever we want to run certain code without logging into the machine or without making a GUI for that code, we should create a Windows Service.

Look and Feel

The best method to start working on Windows Service is to check out the existing Windows Services. Let's open up the Services snap-in. Depending on your Windows OS, this can be done in one of the following ways:

1. Start → Control Panel → Administrative Tools → Services.
2. Start → Control Panel → Administrative Tools → Computer Management → Services and Applications on the left pane → Services.
3. Start → Control Panel → Performance and Management → Services
4. Start → Run → Services.msc

Here you will see all the services registered on your machine, with the details described in the following table.

Details of a Service

Property	Value	Description
Name	<Any>	Name of the service
Description	<Any>	Description of the service
Status	<Blank>	The service is not currently running.
	Started	The service is currently running.
	Paused	The service is currently paused.
Startup Type	Manual	The service will not start when the machine starts.
	Automatic	The service will start when the machine starts.
	Disabled	The service cannot be started.

Log On As	Local System	The most powerful account. It can work on any Operating System file of the machine. Will behave with a network in the same manner as the *Network Service* account.
	Network Service	An account with reduced privileges, equivalent to an authenticated local user account. Also, when the service tries to access resources over a network, it will provide the authentication details of the machine. So, the network will allow the access to its resources as it allows to that machine.
	Local Service	Same as *Network Service* account. But when the service tries to access resources over a network, it will provide no authentication details. So, the network will allow access to only those resources which are public.
	<Any other account>	The service will run with all the privileges of this account. The account can be a network account.

Right click on any service. You will see that you have the options Start, Stop, Pause, Resume, Restart. Some of these options will be disabled. For example, if the service status is Started, then the Start option will be disabled and if it is blank, then the Stop, Pause, Resume, Restart options will be disabled. Most of the services will have Pause and Resume options disabled. All these options are self-explanatory.

Configuring a service

Double click on any service. This will open up its properties with the following tabs: General, Log On, Recovery, Dependencies. The properties discussed till now can be set using the General or the Log On tabs. You should discover what can be done under the Recovery tab or what is seen under the Dependencies tab. The Remote Procedure Call (RPC) service is a very good starting point to discover these options.

A Windows Service Skeleton

Let's create a Windows Service and follow whatever comes our way, very carefully.

1. Create a new Windows Service Project. Let's call it `WinServ`.
2. As soon as you finish the last step, you will see that a service called `Service1` has already been added to the Project. `Service1` will be open

in the Design View. If you click on the main pane, the Properties window (opened by pressing F4, if not already open) will show a few properties. Let's concentrate on some Boolean properties:

Properties of `Service` class

Member	**Description**
AutoLog	If set to `true`, the normal operations on the service – started, stopped, paused - are reported to the EventLog of the System.
CanHandlePowerEvent	If set to `true`, the service can respond to change in the power mode – for example change from AC to battery or vice-versa, etc..
CanHandleSessionChangeEvent	If set to `true`, the service can respond to logging on or off by a user.
CanPauseAndContinue	If set to `true`, the service can be paused and then resumed.
CanShutdown	If set to `true`, the service can respond to the shutting down of the machine.
CanStop	If set to `true`, the service can be stopped.

3. Now, click on the link in the design view to go to the code view. You will see the following auto generated code:

```
public partial class Service1 : System.ServiceProcess.ServiceBase {
    public Service1() {
        InitializeComponent();
    }

    protected override void OnStart(string[] args) { }

    protected override void OnStop() { }
}
```

Analysis

1. The class `Service1` inherits from the class `System.ServiceProcess.ServiceBase`.
2. Whatever code is written in the `OnStart` method will be executed every time the service is started.
3. If the `CanStop` property of the service is set to `true`, the code in the `OnStop` method will execute, whenever the service is stopped. Similarly, you can override the methods `OnPause` and `OnContinue`, which will work only when the `CanPauseAndContinue` property of the service is set to `true`. Similarly, you can override the methods `OnSessionChange`, `OnShutDown` and `OnPowerEvent`, which work only when

the service properties `CanHandleSessionChangeEvent`, `CanShutdown` and `CanHandlePowerEvent`, respectively, are set to `true`.

Filling up the skeleton

Let us make `Service1` write the current time every minute, to a file.

Here is an approximate code that we will be tempted to write in the `OnStart` method:

```
StrmWtr = new StreamWriter("C:\\Temp\\MinuteStamp.txt");
while(true) {
    StrmWtr.Write(DateTime.Now.ToString());
    Thread.Sleep(60000);
}
```

Although the program will compile correctly, this will cause the service *to try* to start forever. Ultimately the operating system will stop trying and give the message: *Error 1053: The service did not respond to the start or control request in a timely fashion.*

Let us learn a concept here. When a service start is attempted, the OS creates a process for this service and runs the code written in the `OnStart` method. The service is *considered started only when* the code in the `OnStart` method finishes. In the above code, this never happens, so the error was generated.

Then how does a service work?

The technique is to initialize the variables in the `OnStart` method and then start a separate thread which actually does the work. Since most of the services do their tasks repeatedly, after some interval, instead of using a simple `Thread` object and creating a loop and using the `Thread.Sleep` method in it, we should use a `Timer`. If the code is to be run just one, then the technique of creating a thread is good.

Sample Code

```
using System;
using System.IO;

public partial class Service1 : System.ServiceProcess.ServiceBase
{
    StreamWriter StrmWtr;
    System.Timers.Timer tmr;

    public Service1() {
        InitializeComponent();
    }

    protected override void OnStart(string[] args) {
        this.StrmWtr = new StreamWriter("C:\\Temp\\MinuteStamp.txt");
        this.tmr = new System.Timers.Timer(60000);
        this.tmr.Elapsed += this.tmr_Elapsed;
        this.tmr.Start();
    }

    protected override void OnStop() {
        this.tmr.Stop();
        this.StrmWtr.Close();
    }

    protected void tmr_Elapsed(object sender, EventArgs e) {
        this.StrmWtr.WriteLine(DateTime.Now.ToString());
    }
}
```

That's it; we just created a Windows Service!

Create another service – to experiment with various properties

Let's create another service in the same project by taking the following steps:

1. Click on the Menu Item *Project → Add New Item… Ctrl+Shift+A*.
2. In the dialog box that opens, click on the item *Windows Service* in the right hand pane. On the bottom of the dialog box, give the name of the file as `Serv2.cs`. This will create a service called `Serv2`, just like `Service1`.
   You will see that the class that is created is also called `Serv2` and this class name is shown in the (Name) property in the Design View.
3. For its functionality, make it write to another file every 10 seconds.
4. Set its `ServiceName` property as `Serv2_OS`. This will not be seen anywhere but it will be used by the OS, for example, at the time of installation or uninstallation of the service or if we have to start the service programmatically or from the command prompt.
5. Make its `CanPauseAndContinue` property `true`. That way, we will be able to Pause and Resume the service in the Services snap-in.
6. Make its `CanHandleSessionChangeEvent` property `true`.
7. Besides changing the timer interval to 10000, here is the additional code that we should write to make these actions meaningful:

```
protected override void OnPause() {
    base.OnPause();
```

```
        this.tmr.Stop();
        this.StrmWtr.Close();
    }

    protected override void OnContinue() {
       base.OnContinue();
       this.StrmWtr = new StreamWriter(
            File.Open("C:\\Temp\\TenSecStamp.txt", FileMode.OpenOrCreate)
          );
       this.tmr.Start();
    }

    protected override void OnSessionChange(
        SessionChangeDescription changeDescription
    ) {
        base.OnSessionChange(changeDescription);
        int iSessID = changeDescription.SessionId;
        switch(changeDescription.Reason) {
           case SessionChangeReason.SessionLogoff:
             this.StrmWtr.WriteLine("Logging off from Session "+iSessID);
             break;
           case SessionChangeReason.SessionLogon:
             this.StrmWtr.WriteLine("Logging onto Session " + iSessID);
             break;
        }
    }
```

8. Open the `Program.cs` file.
9. Add `Serv2` to the `ServicesToRun` array in the `Main()` method. Here is how it will look like:

```
ServiceBase[] ServicesToRun =
    new ServiceBase[] { new Service1(), new Serv2() };
ServiceBase.Run(ServicesToRun);
```

Caution: If you forget the last step, you will get the error: *Could not start the service on local computer. Error 1083: the executable program that this service is configured to run in does not implement the service*, on trying to run the service.

Running the Service

A Windows Service Project cannot be debugged like we debug a Windows Forms Project or a Web Application Project. The only way to check our code is to build the solution and install the service to the OS and then run it from the Services snap-in. This involves 5 steps.

Step 1: Create installers

Open `Service1` in the Design View. Right click on the main pane and click *Add Installer*. That is all that is to it.

Note, that a file ProjectInstaller.cs will be created and it will open up in the Design View and show 2 components: serviceInstaller1 and serviceProcessInstaller1.

Add another installer by right clicking on the Design View of Serv2. Now, you will see only 1 additional component on the ProjectInstaller.cs Design View. This is serviceInstaller2.

That means the Visual Studio creates:

1. One serviceInstaller for each service in the project.
2. One serviceProcessInstaller for the entire project.

Step 2: Set installer properties

For both the serviceInstallers, set the DisplayName and Description properties. Set the StartType property if you want. These properties will appear for the services, in the Services snap-in, as we observed in the *Look and Feel* section.

Step 3: Build

Self-explanatory

Step 4: Install

1. Open the Visual Studio Command Prompt. You will find this under the Start → Programs → Microsoft Visual Studio 2008 → Visual Studio Tools.
2. In the Command Window, navigate to the bin → Debug folder of the Windows Services Project.
3. Enter the text `installutil WinServ.exe` and then press Enter.
4. In the Set Service Login window that opens up, enter the credentials of the user as whom *all* the services will run. Make sure that you also enter the domain in the username textbox. For example, `MyMachN\VineetS`.

This is how this process will appear in the command window (some portions are truncated).

```
C:\Documents and Settings\VineetS\My Documents\Visual Studio
2008\Projects\WinServ\WinServ\bin\Debug>installutil WinServ.exe

Microsoft (R) .NET Framework Installation utility Version 2.0.50727.3053
Copyright (c) Microsoft Corporation.  All rights reserved.
```

```
Running a transacted installation.

Beginning the Install phase of the installation.
See the contents of the log file for the
C:\...\WinServ\WinServ\bin\Debug\WinServ.exe assembly's progress.
The file is located at
C:\...\WinServ\WinServ\bin\Debug\WinServ.InstallLog.
Installing assembly 'C:\...\WinServ\WinServ\bin\Debug\WinServ.exe'.
Affected parameters are:
   logtoconsole =
   assemblypath = C:\...\WinServ\WinServ\bin\Debug\WinServ.exe
   logfile = C:\...\WinServ\WinServ\bin\Debug\WinServ.InstallLog

Installing service Service1...
Service Service1 has been successfully installed.
Creating EventLog source Service1 in log Application...
Installing service Serv2_OS...
Service Serv2_OS has been successfully installed.
Creating EventLog source Serv2_OS in log Application...

The Install phase completed successfully, and the Commit phase is
beginning.
See the contents of the log file for the
...
...
...
The Commit phase completed successfully.

The transacted install has completed.
```

Note: The OS used the ServiceName `Serv2_OS` in the installation log.

Step 5: Start the services

Start the services as we discussed in the *Look and Feel* section. Verify if the values that you set for the `DisplayName` and `Description` properties appear in the Services snap-in. Also, verify if you can Pause and then Resume the Serv2 service (the name in the snap-in will be the one that you gave in the `DisplayName` property of the serviceInstaller2).

Advanced Analyses

Startup Type

The Startup Type discussed in the Look and Feel section can be set up for *each* service by clicking on its serviceInstaller in the Design View of the ProjectInstaller.cs and setting the property `StartType`.

Log On As

The *Log On As* attribute of all the services is set up at the time of installation, as we did earlier. If you want to enter it at the time of writing the code itself, you can do so by opening up the `ProjectInstaller.cs` in the Code View and updating the constructor to:

```
InitializeComponent();
this.serviceProcessInstaller1.Account =
    System.ServiceProcess.ServiceAccount.User;
this.serviceProcessInstaller1.Username = "MyMachN\\VineetS";
this.serviceProcessInstaller1.Password = "MyPwd";
```

Tip: Usually, an enterprise policy is that the password of a user should be changed regularly. If you are supposed to run a service as a particular user, make sure that your account manager sets this account with the *Password never expires* option. Otherwise, you will have to update the *Log On As* attribute of all your services, every time the password changes.

If you want to set the *Log On As* attribute to other options that were tabulated earlier, set the `this.serviceProcessInstaller1.Account` property to the `LocalService`, `LocalSystem` or `NetworkService` value of the `System.ServiceProcess.ServiceAccount` enum. In these cases, setting the `Username` and the `Password` properties will have no effect on the service.

Note: We cannot set the `Account`, `Username` or `Password` properties differently for services in the same project.

AutoLog

Open the Event Viewer under the Administrative Tools or under the Administrative Tools → Computer Management. Click on the System node. You should see the log entries every time some operation is done on a service. If you don't want to see these entries, you can set the `AutoLog` property of the service to `false`.

SessionChangeDescription

We can know the details of change of a session while the service is running since we have set the `CanHandleSessionChangeEvent` property true and implemented the `OnSessionChange` method in the service `Serv2`. This is done using the `System.ServiceProcess.SessionChangeDescription` parameter. It has 2 important properties: `SessionId` (`int`: self-explanatory) and `Reason`. `Reason` can be any one of the `System.ServiceProcess.`**`SessionChangeReason`** enums: `ConsoleConnect`, `ConsoleDisconnect`, `RemoteConnect`, `RemoteDisconnect`, `SessionLogon`, `SessionLogoff`, `SessionLock`, `SessionUnlock`, `SessionRemoteControl`, all of which are self-explanatory.

OnShutDown

If you have the `CanShutDown` property of the service set to true, then you can override the method void `OnShutdown()` which will run when the machine shuts down. You can do the cleaning up work in this method.

OnPowerEvent

If you have the `CanHandlePowerEvent` property of the service set to true, then you can override the method `bool OnPowerEvent(PowerBroadcastStatus powerStatus)` to handle the change in power status. You can do this by using the `PowerBroadcastStatus` parameter. Some of its values are intuitive like `BatteryLow`, `PowerStatusChange` (from battery to AC or vice-versa), `Suspend`. Some other are: `OemEvent`, `QuerySuspend` (system has requested all applications to assent to suspend), `QuerySuspendFailed`, `ResumeAutomatic` (system has woken up to do something, for example, a scheduled task), `ResumeCritical` (system has woken up after battery went to a critical state), `ResumeSuspend` (system has woken up).

Controlling a Service Programmatically

You can review and control a Windows Service programmatically using an instance of the `ServiceController` class, say `ServCtrl`. All that you have to do is to specify the service. You can then access all its properties that we have seen earlier, using the `ServCtrl`.

Sample code

```
ServiceController ServCtrl = new ServiceController("Serv2_OS");
if (ServCtrl.CanPauseAndContinue) {
    ServCtrl.Pause();
}
```

Note: Like all Windows Service related classes, the `ServiceController` is in the `System.ServiceProcess` namespace and resides in the DLL of the same name. Remember to add a reference to this DLL in your project.

Analyses

ServiceName

Note that we have used the calue of the `ServiceName` property of the `Serv2` service that we created earlier, in order to get a handle to it.

MachineName

By setting the `MachineName` property of the `ServiceController` instance, we can control a service which resides on a remote machine.

Basic Properties and Methods

In the sample code, we used the `CanPauseAndContinue` property and `Pause` method to accomplish our goal. Similarly, we can use the `Start` and `Stop` methods and also get the values of all the properties that we discussed earlier, for the service which the `ServiceController` controls.

ServicesDependedOn

With the `ServicesDependedOn` property of the `ServCtrl`, we get an array of service controllers to all its dependency services.

Caution: The `ServiceController` class inherits the `Component` class and thus its `Site` property. This property has nothing to do with the machine on which the underlying service resides.

Controlling a Service from the Command Prompt

You can run the `Net Start Serv2`, `Net Stop Serv2`, `Net Pause Serv2` or `Net Continue Serv2` commands from the command prompt to start, stop, pause or continue the services, respectively. For example, the `Serv2_OS` which was paused by the last sample code could be continued. This will be the entire output:

```
C:\>net continue serv2_os
The Serv2_Display service continue is pending.
The Serv2_Display service was continued successfully.
```

Uninstalling a Service

The steps are the same as described in the *Step 4: Install* subsection of the section *Running the Service* section. This time, just enter the text `installutil /u WinServ.exe` after browsing to the appropriate folder in the command prompt.

References

The Services and Service Accounts Security Planning Guide, Microsoft Corporation.

http://msdn.microsoft.com/en-us/library/system.serviceprocess.sessionchangereason.aspx

http://msdn.microsoft.com/en-us/library/system.serviceprocess.powerbroadcaststatus.aspx