.NET console application as Windows service

Asked 10 years, 1 month ago Active 6 months ago Viewed 152k times

157

66

I have console application and would like to run it as Windows service. VS2010 has project template which allow to attach console project and build Windows service. I would like to not add separated service project and if possible integrate service code into console application to keep console application as one project which could run as console application or as windows service if run for example from command line using switches.

Maybe someone could suggest class library or code snippet which could quickly and easily transform c# console application to service?

c# .net-4.0 windows-services console-application

Share Edit Follow Flag

asked Oct 14 '11 at 6:58
Tomas
16k 39 138 236

Why don't you just create a temporary service project and copy over the bits that make it a service? – Gabe Oct 14 '11 at 7:04

4 You could try Topshelf topshelf-project.com – Artem Koshelev Oct 14 '11 at 7:12

You could try the technique described here: einaregilsson.com/2007/08/15/... – Joe Oct 14 '11 at 7:34

huh? I'm not sure. about this. – user824152 Feb 21 '13 at 5:39

2 A very simple top shelf alternative: runasservice.com – Luis Perez Sep 25 '13 at 2:53

10 Answers

Active Oldest Votes

207

✓

I usually use the following techinque to run the same app as a console application or as a service:

```
using System.ServiceProcess

public static class Program
{
    #region Nested classes to support running as service
    public const string ServiceName = "MyService";

    public class Service : ServiceBase
    {
        public Service()
        {
            ServiceName = Program.ServiceName;
        }
        protected override void OnStart(string[] args)
        {
            Program.Start(args);
        }

        protected override void OnStop()
        {
            Program.Stop();
        }
    }
    #endregion

    static void Main(string[] args)
    {
        if (!Environment.UserInteractive)
            // running as service
            using (var service = new Service())
                ServiceBase.Run(service);
        else
        {
            // running as console app
            Start(args);

            Console.WriteLine("Press any key to stop...");
            Console.ReadKey(true);

            Stop();
        }
    }

    private static void Start(string[] args)
    {
        // onstart code here
    }

    private static void Stop()
    {
        // onstop code here
    }
}
```

`Environment.UserInteractive` is normally true for console app and false for a service. Techically, it is possible to run a service in user-interactive mode, so you could check a command-line switch instead.

Share Edit Follow Flag

edited May 14 at 7:32
Joshua Swain
304 1 3 15

answered Oct 14 '11 at 7:33
VladV
9,583 2 30 46

3 You use ServiceInstaller class, see msdn.microsoft.com/en-us/library/.... – VladV Oct 25 '14 at 20:45

2 That's expected - your service would run as a separate process (so it would be shown in the task manager), but this process would be controlled by the system (e.g. started, stopped, restarted according to the service settings). – VladV May 22 '16 at 8:43

2 If you run it as a console app, you won't see a service. The whole purpose of this code is to enable you to run it either as a console app, or as a service. To run as a service you need to install it first (using ServiceInstaller class - see MSDN link above - or installuitil.exe), and the run the service from the control panel. – VladV May 24 '16 at 18:23

2 ServiceInstaller is just a utility class to deal with Windows services (a little bit like installutil.exe or sc.exe utilities). You could use it to install whatever you want as a service, the OS doesn't care about the project type you use. – VladV Jun 22 '16 at 14:13

7 Just add a reference in your project to **System.ServiceProcess** and you'll be able to use the code above – danimal

Aug 24 '18 at 4:32

62

I've had great success with TopShelf.

TopShelf is a Nuget package designed to make it easy to create .NET Windows apps that can run as console apps or as Windows Services. You can quickly hook up events such as your service Start and Stop events, configure using code e.g. to set the account it runs as, configure dependencies on other services, and configure how it recovers from errors.

From the Package Manager Console (Nuget):

```
Install-Package Topshelf
```

Refer to the code samples to get started.

Example:

```
HostFactory.Run(x =>
{
    x.Service<TownCrier>(s =>
    {
       s.ConstructUsing(name=> new TownCrier());
       s.WhenStarted(tc => tc.Start());
       s.WhenStopped(tc => tc.Stop());
    });
    x.RunAsLocalSystem();

    x.SetDescription("Sample Topshelf Host");
    x.SetDisplayName("Stuff");
    x.SetServiceName("stuff");
});
```

TopShelf also takes care of service installation, which can save a lot of time and removes boilerplate code from your solution. To install your .exe as a service you just execute the following from the command prompt:

```
myservice.exe install -servicename "MyService" -displayname "My Service" -description
"This is my service."
```

You don't need to hook up a ServiceInstaller and all that - TopShelf does it all for you.

Share Edit Follow Flag

edited Apr 27 '15 at 22:13

answered Aug 3 '14 at 21:11

saille
8,589 3 43 56

1 Hi, i am getting this :- "Could not install package 'Topshelf 4.0.1'. You are trying to install this package into a project that targets '.NETFramework,Version=v4.5', but the package does not contain any assembly references or content files that are compatible with that framework." what is wrong here? – user6102644 May 20 '16 at 7:23

3 Make sure you are targetting the full .NET 4.5.2 runtime, not the Client profile. – saille Jul 28 '16 at 1:33

please can you throw more light on myservice.exe and from which directory are you going to open the command prompt – Izuagbala Mar 5 '18 at 15:58

1 @Izuagbala myservice.exe is the console application that you have created, with TopShelf bootstrapped into it as shown in the code sample. – saille Mar 12 '18 at 21:36

Can myservice.exe be run as console after it is installed as a service?. Documentation is not clear: "Once the console application is created, the developer creates a single service class " docs.topshelf-project.com/en/latest/overview/... – Michael Freidgeim Mar 18 '19 at 3:46

32

So here's the complete walkthrough:

1. Create new Console Application project (e.g. MyService)
2. Add two library references: System.ServiceProcess and System.Configuration.Install
3. Add the three files printed below
4. Build the project and run "InstallUtil.exe c:\path\to\MyService.exe"
5. Now you should see MyService on the service list (run services.msc)

*InstallUtil.exe can be usually found here: C:\windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe

Program.cs

```
using System;
using System.IO;
using System.ServiceProcess;

namespace MyService
{
    class Program
    {
        public const string ServiceName = "MyService";

        static void Main(string[] args)
        {
            if (Environment.UserInteractive)
            {
                // running as console app
                Start(args);

                Console.WriteLine("Press any key to stop...");
                Console.ReadKey(true);

                Stop();
            }
            else
            {
                // running as service
                using (var service = new Service())
                {
                    ServiceBase.Run(service);
                }
            }
        }

        public static void Start(string[] args)
        {
            File.AppendAllText(@"c:\temp\MyService.txt", String.Format("{0}
started{1}", DateTime.Now, Environment.NewLine));
        }

        public static void Stop()
        {
            File.AppendAllText(@"c:\temp\MyService.txt", String.Format("{0}
stopped{1}", DateTime.Now, Environment.NewLine));
        }
    }
}
```

MyService.cs

```
using System.ServiceProcess;

namespace MyService
{
    class Service : ServiceBase
    {
        public Service()
        {
            ServiceName = Program.ServiceName;
        }

        protected override void OnStart(string[] args)
        {
            Program.Start(args);
        }

        protected override void OnStop()
        {
            Program.Stop();
        }
    }
}
```

MyServiceInstaller.cs

```
using System.ComponentModel;
using System.Configuration.Install;
using System.ServiceProcess;

namespace MyService
{
    [RunInstaller(true)]
    public class MyServiceInstaller : Installer
    {
        public MyServiceInstaller()
        {
            var spi = new ServiceProcessInstaller();
            var si = new ServiceInstaller();

            spi.Account = ServiceAccount.LocalSystem;
            spi.Username = null;
            spi.Password = null;

            si.DisplayName = Program.ServiceName;
            si.ServiceName = Program.ServiceName;
            si.StartType = ServiceStartMode.Automatic;

            Installers.Add(spi);
            Installers.Add(si);
        }
    }
}
```

Share Edit Follow Flag

edited Jan 21 '17 at 19:31

answered Jan 21 '17 at 19:13

Nikolai Koudelia
2,296 1 23 25

2 If you are compiling your project for 64 bit you have to use the InstallUtil.exe for 64 bit which can be found here: C:\windows\Microsoft.NET\Framework64\... The version for 32 bit (C:\windows\Microsoft.NET\Framework) will throw a BadImageFormatException at you... – snytek Oct 31 '17 at 16:33

This works very well, note that as @snytek says, if you are using base 64, make sure that you use the correct directory. Also, if you happen to do the same as me and forget to rename the service to something other than "MyService", make sure that you uninstall the service before making the changes to the code. – dmoore1181 Oct 29 '18 at 19:48

8

Here is a newer way of how to turn a Console Application to a Windows Service as a Worker Service based on the latest **.Net Core 3.1**.

If you create a Worker Service from Visual Studio 2019 it will give you almost everything you need for creating a Windows Service out of the box, which is also what you need to change to the console application in order to convert it to a Windows Service.

Here are the changes you need to do:

Install the following NuGet packages

```
Install-Package Microsoft.Extensions.Hosting.WindowsServices -Version 3.1.0
Install-Package Microsoft.Extensions.Configuration.Abstractions -Version 3.1.0
```

Change Program.cs to have an implementation like below:

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace ConsoleApp
{
    class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).UseWindowsService().Build().Run();
        }

        private static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureServices((hostContext, services) =>
                {
                    services.AddHostedService<Worker>();
                });
    }
}
```

and add Worker.cs where you will put the code which will be run by the service operations:

```
using Microsoft.Extensions.Hosting;
using System.Threading;
using System.Threading.Tasks;

namespace ConsoleApp
{
        public class Worker : BackgroundService
        {
            protected override async Task ExecuteAsync(CancellationToken stoppingToken)
            {
                //do some operation
            }

            public override Task StartAsync(CancellationToken cancellationToken)
            {
                return base.StartAsync(cancellationToken);
            }

            public override Task StopAsync(CancellationToken cancellationToken)
            {
                return base.StopAsync(cancellationToken);
            }
        }
    }
```

When everything is ready, and the application has built successfully, you can [use *sc.exe*] to install your console application exe as a Windows Service with the following command:

```
sc.exe create DemoService binpath= "path/to/your/file.exe"
```

Share Edit Follow Flag

answered Jan 12 '20 at 23:27
meJustAndrew
4,953 5 42 66

4

Firstly I embed the console application solution into the windows service solution and reference it.

Then I make the console application Program class public

```
/// <summary>
/// Hybrid service/console application
/// </summary>
public class Program
{
}
```

I then create two functions within the console application

```
    /// <summary>
    /// Used to start as a service
    /// </summary>
    public void Start()
    {
        Main();
    }

    /// <summary>
    /// Used to stop the service
    /// </summary>
    public void Stop()
    {
       if (Application.MessageLoop)
            Application.Exit();   //windows app
        else
            Environment.Exit(1);  //console app
    }
```

Then within the windows service itself I instantiate the Program and call the Start and Stop functions added within the OnStart and OnStop. See below

```
class WinService : ServiceBase
{
    readonly Program _application = new Program();

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    static void Main()
    {
        ServiceBase[] servicesToRun = { new WinService() };
        Run(servicesToRun);
    }

    /// <summary>
    /// Set things in motion so your service can do its work.
    /// </summary>
    protected override void OnStart(string[] args)
    {
        Thread thread = new Thread(() => _application.Start());
        thread.Start();
    }

    /// <summary>
    /// Stop this service.
    /// </summary>
    protected override void OnStop()
    {
        Thread thread = new Thread(() => _application.Stop());
        thread.Start();
    }
}
```

This approach can also be used for a windows application / windows service hybrid

Share Edit Follow Flag

answered Feb 13 '15 at 13:31
Patrick Reynolds
41 1

this is basically what JonAlb have said in the prev answer, but thanks for the code example – tatigo Jan 26 '16 at 16:35

3

I hear your point at wanting one assembly to stop repeated code but, It would be simplest and reduce code repetition and make it easier to reuse your code in other ways in future if...... you to break it into 3 assemblies.

1. One library assembly that does all the work. Then have two very very slim/simple projects:
2. one which is the commandline
3. one which is the windows service.

Share Edit Follow Flag

answered Oct 14 '11 at 7:22
JonAlb
1,702 10 13

1 This is how I've done it for years - the Service pretty much has `Start()` and `Stop()` methods and the console app has a loop. Short of using a framework like [TopShelf](), this is the best option – Basic Nov 30 '13 at 0:34

agree with that answer the most. using 3d party tools for simple solutions makes the future maintenances unnecessary complex – tatigo Jan 26 '16 at 16:33

2

You can use

```
reg add HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run /v ServiceName
/d "c:\path\to\service\file\exe"
```

And it will appear int the service list. I do not know, whether that works correctly though. A service usually has to listen to several events.

There are several service wrapper though, that can run any application as a real service. For Example Microsofts SrvAny from the Win2003 Resource Kit

Share Edit Follow Flag

answered Oct 14 '11 at 7:15
Darcara
1,568 1 11 32

As you say, the service exe will need to communicate with windows. +1 for link to SrvAny – Jodrell Oct 14 '11 at 7:34

5 I'd consider this approach unsafe. Windows has special libraries and utilities to manage services, and they are more likely to work consistently in different OS versions and environments. For .NET app it is quite easy to create an MSI installer in VS. It is also posisble to perform installation progrmmatically using ManagedInstallerClass.InstallHelper method. – VladV Oct 14 '11 at 7:42

1 No need for installers and stuff: just use this command line: sc create MyServiceName binPath= "c:\path\to\service\file\exe" – JDC May 19 '17 at 8:23

0

You need to seperate the functionality into a class or classes and launch that via one of two stubs. The console stub or service stub.

As its plain to see, when running windows, the myriad services that make up the infrastructure do not (and can't directly) present console windows to the user. The service needs to communicate with the user in a non graphical way: via the SCM; in the event log, to some log file etc. The service will also need to communicate with windows via the SCM, otherwise it will get shutdown.

It would obviously be acceptable to have some console app that can communicate with the service but the service needs to run independently without a requirement for GUI interaction.

The Console stub can very useful for debugging service behaviour but should not be used in a "productionized" environment which, after all, is the purpose of creating a service.

I haven't read it fully but this article seems to pint in the right direction.

Share Edit Follow Flag

edited Oct 14 '11 at 7:27

answered Oct 14 '11 at 7:22

Jodrell
32.3k 4 77 117

0

I use a service class that follows the standard pattern prescribed by `ServiceBase`, and tack on helpers to easy F5 debugging. This keeps service data defined within the service, making them easy to find and their lifetimes easy to manage.

I normally create a Windows application with the structure below. I don't create a console application; that way I don't get a big black box popping in my face every time I run the app. I stay in in the debugger where all the action is. I use `Debug.WriteLine` so that the messages go to the output window, which docks nicely and stays visible after the app terminates.

I usually don't bother add debug code for stopping; I just use the debugger instead. If I do need to debug stopping, I make the project a console app, add a `Stop` forwarder method, and call it after a call to `Console.ReadKey`.

```
public class Service : ServiceBase
{
    protected override void OnStart(string[] args)
    {
        // Start logic here.
    }

    protected override void OnStop()
    {
        // Stop logic here.
    }

    static void Main(string[] args)
    {
        using (var service = new Service()) {
            if (Environment.UserInteractive) {
                service.Start();
                Thread.Sleep(Timeout.Infinite);
            } else
                Run(service);
        }
    }
    public void Start() => OnStart(null);
}
```

Share Edit Follow Flag

answered Mar 12 '19 at 15:28
Edward Brey
37k 15 180 229

0

Maybe you should define what you need, as far as I know, you can't run your app as Console or Service with command line, at the same time. Remember that the service is installed and you have to start it in Services Manager, you can create a new application wich starts the service or starts a new process running your console app. But as you wrote

> "keep console application as one project"

Once, I was in your position, turning a console application into a service. First you need the template, in case you are working with VS Express Edition. Here is a link where you can have your first steps: C# Windows Service, this was very helpful for me. Then using that template, add your code to the desired events of the service.

To improve you service, there's another thing you can do, but this is not quick and/or easily, is using appdomains, and creating dlls to load/unload. In one you can start a new process with the console app, and in another dll you can just put the functionality the service has to do.

Good luck.

Share Edit Follow Flag

answered Oct 14 '11 at 7:21
BlackCath
804 11 23