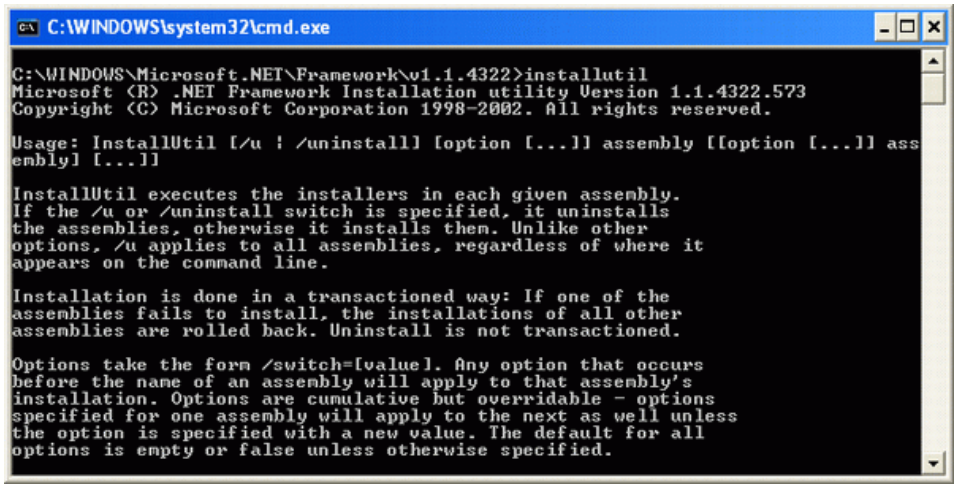
Install a Service using a **Script**

gtamir, 12 Nov 2004

How to install a service using a **script** instead of a Windows Installer MSI package.

Download source files - 4.51 Kb

Download demo project - 7.1 Kb



Introduction

Many believe that an MSI install file is the only way to install a Windows service. In truth, using an MSI is only one of several ways to install a Windows service. This article will explain how to perform a **script**ed install of a Windows service. In addition to explaining how to install a service using a **script**, this article will also explain how to install multiple instances of the same assembly as multiple Windows services.

Background

I maintain a distributed software application. The application back-end requires multiple services running on multiple servers. Each server hosts 15-25 services.

Creating an MSI file for each service and then manually running Windows Installer for 25 services on each machine is a terrible burden on my IT OPS team.

The solution I use is 'XCOPY deployment'. To install, just unzip (or copy from CD) the entire directory tree (containing all services), and then run an install **script** to register all services. To uninstall – run the uninstall **script** and delete the directory tree.

In my experience, using the XCOPY deployment is much quicker and requires far less work than using individual Windows Installer MSI files. Yes, there is a way to bundle multiple MSI files into one 'master' package, but keep reading and you will understand why **script**ed install is so much better. For additional information about XCOPY deployment, see Determining When to Use Windows Installer Versus XCOPY.

The problem with MSI

Installing a service using an MSI file can be very convenient if you install a very limited set of services (one to four). When using an MSI file, you may encounter the following problems:

- Your service executable and DLLs are stored in an arbitrary location (usually under *C:\Program Files*). The installation location is user-controlled so you cannot make too many assumptions about assembly and DLL locations.
- By default, Windows Installer puts your DLLs in the GAC. Your DLLs become machine-global. You have to correctly manage version numbers (or make sure DLLs are backward compatible) to avoid version conflict problems.
- Every so often, Windows Installer corrupts the GAC reference counts. Once corruption occurs, you can't uninstall your service. When trying to uninstall your service, you get an error like:

Hide Copy Code

```
Unable to uninstall: assembly is required by one
                        or more applications Pending references:
SCHEME: <WINDOWS_INSTALLER>ID: <MSI> DESCRIPTION:<Windows Installer>
```

If you have already encountered this problem, you can find a solution here.

Using XCOPY deployment, you avoid the Windows Installer GAC corruption problem, **and** get to keep private versions of your DLLs in each service executable directory.

Scripted install basics

Scripted installs use the .NET SDK *InstallUtil.EXE* utility. InstallUtil invokes the `ProjectInstaller` module in your assembly. For installation, InstallUtil monitors all installation steps and rolls back the installation if an error occurs. For uninstalling, InstallUtil runs the Uninstall code in your `ProjectInstaller` module. For more information about the install process, see MSDN documentation for the ServiceInstaller class.

To use **scripted** install, you need three pieces of code:

- A project installer module added to your assembly
- A short installation **script**
- A short un-installation **script**

The Project Installer Module

This is *ProjectInstaller.cs*. A *ProjectInstaller.vb* is available in the Source files package thanks to **David Wimbush**.

Hide Shrink Copy Code

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Configuration.Install;
using System.Collections.Specialized;
using System.ServiceProcess;
using Microsoft.Win32;

namespace <Your namespace here>
{
    /// <summary>
    /// This is a custom project installer.
    /// Applies a unique name to the service using the /name switch
    /// Sets user name and password using the /user and /password switches
    /// Allows the use of a local account using the /account switch
    /// </summary>
    [RunInstaller(true)]
    public class ScriptedInstaller : Installer
    {
        private ServiceInstaller serviceInstaller;
        private ServiceProcessInstaller processInstaller;

        public ScriptedInstaller()
        {
            processInstaller             = new ServiceProcessInstaller();
            serviceInstaller             = new ServiceInstaller();

            // Set some defaults
            processInstaller.Account     =
                                System.ServiceProcess.ServiceAccount.User;
            serviceInstaller.StartType   = ServiceStartMode.Automatic;
            serviceInstaller.ServiceName = "MonitoredServiceExample";

            Installers.Add(serviceInstaller);
            Installers.Add(processInstaller);
        }

        #region Access parameters

        /// <summary>
        /// Return the value of the parameter in dicated by key
        /// </summary>
        /// <PARAM name="key">Context parameter key</PARAM>
        /// <returns>Context parameter specified by key</returns>
        public string GetContextParameter(string key)
        {
            string sValue = "";
            try
            {
                sValue = this.Context.Parameters[key].ToString();
            }
            catch
            {
```

```
            sValue = "";
        }

        return sValue;
    }

    #endregion

    /// <summary>
    /// This method is run before the install process.
    /// This method is overriden to set the following parameters:
    /// service name (/name switch)
    /// account type (/account switch)
    /// for a user account user name (/user switch)
    /// for a user account password (/password switch)
    /// Note that when using a user account,
    /// if the user name or password is not set,
    /// the installing user is prompted for the credentials to use.
    /// </summary>
    /// <PARAM name="savedState"></PARAM>
    protected override void OnBeforeInstall(IDictionary savedState)
    {
        base.OnBeforeInstall(savedState);

        bool isUserAccount = false;

        // Decode the command line switches
        string name        = GetContextParameter("name");
        serviceInstaller.ServiceName = name;

        // What type of credentials to use to run the service
        // The default is User
        string acct        = GetContextParameter("account");

        if (0 == acct.Length) acct = "user";

        // Decode the type of account to use
        switch (acct)
        {
            case "user":
                processInstaller.Account =
                   System.ServiceProcess.ServiceAccount.User;
                isUserAccount = true;
                break;
            case "localservice":
                processInstaller.Account =
                  System.ServiceProcess.ServiceAccount.LocalService;
                break;
            case "localsystem":
                processInstaller.Account =
                  System.ServiceProcess.ServiceAccount.LocalSystem;
                break;
            case "networkservice":
                processInstaller.Account =
                  System.ServiceProcess.ServiceAccount.NetworkService;
                break;
            default:
                processInstaller.Account =
                  System.ServiceProcess.ServiceAccount.User;
                isUserAccount = true;
                break;
        }

        // User name and password
        string username = GetContextParameter("user");
        string password = GetContextParameter("password");

        // Should I use a user account?
        if (isUserAccount)
        {
            // If we need to use a user account,
            // set the user name and password
            processInstaller.Username = username;
            processInstaller.Password = password;
        }
    }

    /// <summary>
    /// Modify the registry to install the new service
    /// </summary>
    /// <PARAM name="stateServer">></PARAM>
    public override void Install(IDictionary stateServer)
    {
        RegistryKey system,
            //HKEY_LOCAL_MACHINE\Services\CurrentControlSet
            currentControlSet,
            //...\Services
            services,
            //...\<Service Name>
            service,
            //...\Parameters - this is where you can
            //put service-specific configuration
            config;
```

```
            base.Install(stateServer);

            // Define the registry keys
            // Navigate to services
            system              = Registry.LocalMachine.OpenSubKey("System");
            currentControlSet   = system.OpenSubKey("CurrentControlSet");
            services            = currentControlSet.OpenSubKey("Services");
            // Add the service
            service             =
              services.OpenSubKey(this.serviceInstaller.ServiceName, true);
            // Default service description
            service.SetValue("Description",
                                "Example ScriptedService implementation");

            // Display the assembly image path
            // and modify to add the service name
            // The executable then strips the name out of the image
            Console.WriteLine("ImagePath: " + service.GetValue("ImagePath"));
            string imagePath = (string)service.GetValue("ImagePath");
            imagePath        += " -s" + this.serviceInstaller.ServiceName;
            service.SetValue("ImagePath", imagePath);
            // Create a parameters subkey
            config            = service.CreateSubKey("Parameters");

            // Close keys
            config.Close();
            service.Close();
            services.Close();
            currentControlSet.Close();
            system.Close();
        }

        /// <summary>
        /// Uninstall based on the service name
        /// </summary>
        /// <PARAM name="savedState"></PARAM>
        protected override void OnBeforeUninstall(IDictionary savedState)
        {
            base.OnBeforeUninstall(savedState);

            // Set the service name based on the command line
            serviceInstaller.ServiceName = GetContextParameter("name");
        }

        /// <summary>
        /// Modify the registry to remove the service
        /// </summary>
        /// <PARAM name="stateServer">/PARAM>
        public override void Uninstall(IDictionary stateServer)
        {
            RegistryKey system,
                //HKEY_LOCAL_MACHINE\Services\CurrentControlSet
                currentControlSet,
                //...\Services
                services,
                //...\<Service Name>
                service;
                //...\Parameters - this is where you can
                //put service-specific configuration

            base.Uninstall(stateServer);

            // Navigate down the registry path
            system            = Registry.LocalMachine.OpenSubKey("System");
            currentControlSet = system.OpenSubKey("CurrentControlSet");
            services          = currentControlSet.OpenSubKey("Services");
            service           =
               services.OpenSubKey(this.serviceInstaller.ServiceName,true);
            // Remove the parameters key
            service.DeleteSubKeyTree("Parameters");

            // Close keys
            service.Close();
            services.Close();
            currentControlSet.Close();
            system.Close();
        }
    }//end class
}//end namespace declaration
```

Notice the `[RunInstaller(true)]` line – this line marks the Project Installer as an installer and makes sure the code is accessible to InstallUtil (and to the MSI installer for that matter).

The constructor

The installer starts by instantiating a process and service installer and setting some defaults.

Runtime parameter access

The `GetContextParameter` property allows access to the runtime parameters supplied by the InstallUtil utility.

Pre-installation

The `OnBeforeInstall()` method is executed prior to the actual installation. Here, the runtime parameters passed by the InstallUtil utility are parsed and the installation attributes are set.

The service name is set using the `/name` switch. Why not use the assembly name? Because, I may need to install multiple instances of the same assembly as different services. The concept of installing multiple instances of the same assembly probably deserves a separate article. For now, I refer you to the Unix *inetd* daemon.

After the service name is set, the service account type is determined. If the account type is a *user account*, the user and password information is obtained. The service is ready to be installed.

Installation

The `Install()` method creates the service registry keys and computes the `"imagepath"` which is the actual command line executed when the service is run. The service name is appended to the `imagepath` so the service can determine what name it is running under.

Pre-Uninstallation

The `OnBeforeUninstall()` method looks for the only relevant parameter - the service name, specified with the `/name` switch.

Uninstallation

The `Uninstall()` method removes the registry keys for the service from the registry.

The Install **Script**

This is *installService.bat*:

Hide Copy Code

```
@echo off
set SERVICE_HOME=<service executable directory>
set SERVICE_EXE=<service executable name>
REM the following directory is for .NET 1.1, your mileage may vary
set INSTALL_UTIL_HOME=C:\WINNT\Microsoft.NET\Framework\v1.1.4322
REM Account credentials if the service uses a user account
set USER_NAME=<user account>
set PASSWORD=<user password>

set PATH=%PATH%;%INSTALL_UTIL_HOME%

cd %SERVICE_HOME%

echo Installing Service...
installutil /name=<service name>
  /account=<account type> /user=%USER_NAME% /password=%

PASSWORD% %SERVICE_EXE%

echo Done.
```

The variables set at the top are for convenience only, you can certainly hardcode all information directly on the `installutil` line.

The Uninstall **Script**

This is *uninstallService.bat*:

Hide Copy Code

```
@echo off
set SERVICE_HOME=<service executable directory>
set SERVICE_EXE=<service executable name>
REM the following directory is for .NET 1.1, your mileage may vary
set INSTALL_UTIL_HOME=C:\WINNT\Microsoft.NET\Framework\v1.1.4322

set PATH=%PATH%;%INSTALL_UTIL_HOME%

cd %SERVICE_HOME%

echo Uninstalling Service...
installutil /u /name=<service name> %SERVICE_EXE%

echo Done.
```

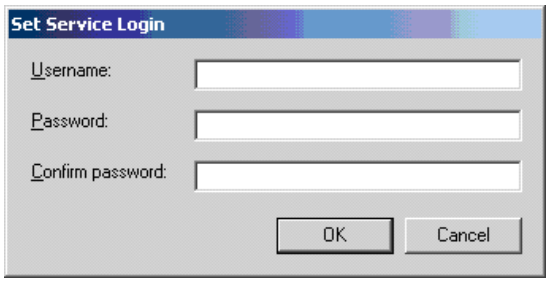
The Parameters

- `Name` – sets the service name to the given string. `/Name=MonitoredServiceExample`.
- `Account` – What type of credentials should the service use. Options are:

  - `/account=user` – use a user account (the default), an account defined by a specific user on the network.
  - `/account=localservice` – use a LocalService account, an account that acts as a non-privileged user on the local computer, and presents anonymous credentials to any remote server.
  - `/account=localsystem` – use a LocalSystem account, an account that has a high privileged level.
  - `/account=networkservice` – use a NetworkService account, an account that provides extensive local privileges, and presents the computer's credentials to any remote server.

- `User` – if the service is to run using a user account, specifies the name of the user. `/user=MyDomain\Me`.
- `Password` – if the service is to run using a user account, specifies the password associated with the user account. `/password=secret`.

Note that if the service is to run with *user account* credentials and either the user name or the password is not specified (or is incorrect), the installing user will be prompted for the user name and password to use during the installation.



The Example

The included example contains the source, project installer, and **script**s for a simple service that does nothing. Compile in Debug mode, and use the install and uninstall **script**s to install and remove the service. Amuse yourself by installing the service with different credentials: `user`, `network service`, `local service`, etc...

Installing Multiple Instances of the Same Assembly

Sometimes, you need to install multiple instances of the same assembly as different services. I can think of several reasons:

- Have multiple instances of the service, each dedicated to a different client/purpose.
- Create a 'master' service that behaves differently based on the name it is given.

Yes, you could multi-thread a service instead of installing multiple instances, but there are advantages to having a separate memory space and being able to start and stop individual instances.

The modification to the install **script** is as follows: Instead of:

Hide Copy Code

```
installutil /name=<service name> /account=<account
         type> /user=%USER_NAME% /password=%PASSWORD% %SERVICE_EXE%
```

use

Hide Copy Code

```
installutil /name=<first service name> /account=<account
         type> /user=%USER_NAME% /password=%PASSWORD% %SERVICE_EXE%
installutil /name=<second service name> /account=<account
          type> /user=%USER_NAME% password=%PASSWORD% %SERVICE_EXE%
```

and for the uninstall **script**, replace

Hide Copy Code

```
installutil /u /name=<service name> %SERVICE_EXE%
```

with

Hide Copy Code

```
installutil /u /name=<first service name> %SERVICE_EXE%
installutil /u /name=<second service name> %SERVICE_EXE%
```

Mind boggling, isn't it?

Allow Service to Interact with Desktop

The following uses undocumented Windows features to set the 'Interact with Desktop' bit. Use at your own risk. You can get some additional information here.

To get the 'interact with desktop' feature requires two steps:

- Add a new switch
- Add an `OnAfterInstall` method to the installer

After adding a new `bool` flag to the class, say `bool m_interactWithDesktop` add the following to the `OnBeforeInstall() method`

Hide Copy Code

```
protected override void OnBeforeInstall(IDictionary savedState)
{
    base.OnBeforeInstall(savedState);
    ...
    string interact = GetContextParameter("InteractWithDesktop");
    if (interact.ToLower() == "true")
    {
        m_interactWithDesktop = true;
    }
    ...
}
```

Add the new `OnAfterInstall()` method

Hide Copy Code

```
protected override void OnAfterInstall(IDicionary savedState)
{
    base.OnAfterInstall(savedState);
    if (m_interactWithDesktop)
    {
        string keyPath = @"SYSTEM\CurrentControlSet\Services\" +
          this.serviceInstaller.ServiceName;
        RegistryKey ckey = Registry.LocalMachine.OpenSubKey(keyPath, true);
        if ((null != ckey) && (null != ckey.GetValue("Type")))
        {
            ckey.SetValue("Type", (int)ckey.GetValue("Type") | 0x100);
        }
    }
}
```

The above cannot be guaranteed to work as the bit values for the 'Type' key are not published.

Possible improvements

Currently, the service name defaults to a hard-coded value if not specified. The name of the executing assembly may be a better default.

The service description is currently hard-coded. Service description could be passed in as a parameter, or an assembly property can be used.

SC.EXE

`SC.EXE` is part of the Windows 2000 resource kit. The `SC.EXE` utility allows you to add, delete and modify services even if you do not have access to the service assemblies or MSI files.

`SC.EXE` is a great administration tool and very handy when you have to deal with emergencies. If you have lost the executable and have a bogus service definition in your registry - `SC.EXE` is the tool for the job. Every system admin should have a collection of tools to help with maintenance, and SC.EXE is a good tool to have in your system admin toolbox.

You can find documentation about `SC.EXE` on Microsoft support and more information on Vasudevan Deepak Kumars blog.

Note however that using a `ProjectInstaller` and `InstallUtil` have the following advantages over `SC.EXE`:

- You can create/modify/delete customized registry entries in the `Install()` method of the `ProjectInstaller`. `SC.EXE` just creates/deletes the standard registry service keys.
- Using `ProjectInstaller` you can define your own command line switches for the service installation, passing in information that makes sense to you. There is no way to pass in optional information using `SC.EXE`.
- While you can instantiate multiple instances of the same assembly using `SC.EXE` (by creating multiple services) there is no way to inform the running service under what name it is running.
- Having a `ProjectInstaller` class makes your service compatible with Windows Installer. If you want to install the same service using an MSI package, no changes are required.
- Removing a service using `SC.EXE` marks the service for deletion on reboot, removing with `InstallUtil` is immediate as the registery changes are done as part of the uninstall.

Additional information about InstallUtil

You can find additional information about the InstallUtil utility on MSDN.

Version history

- V1.0 was used in my article Monitoring Distributed Service Performance in .NET.
- V1.1 with better documentation is included here.
- V1.2 Added Visual Basic.Net version from **David Wimbush**

License

A list of licenses authors might use can be found here

Share

TWITTER FACEBOOK

About the Author

gtamir
Web Developer
United States

Giora Tamir has been Architecting, Designing and Developing software and hardware solutions for over 15 years. As an IEEE Senior member and a talented developer, Giora blends software development, knowledge of protocols, extensive understanding of hardware and profound knowledge of both Unix and Windows based systems to provide a complete solution for both defense and commercial applications. Giora, also known as G.T., now holds the position of Principal Engineer for ProfitLine, Inc. architecting the next generation of .NET applications based on a Service-Oriented-Architecture.

Gioras areas of interest include distributed applications, networking and cryptography in addition to Unix internals and embedded programming.

Founded in 1992, ProfitLine manages hundreds of millions of dollars in annual telecom spend for its prestigious Fortune 1000 client base, such as Merrill Lynch, Charming Shoppes, Macromedia, CNA Financial Corporation, and Constellation Energy Group. ProfitLine's outsourced solution streamlines telecom administrative functions by combining a best practices approach with intelligent technology. For more information about ProfitLine, call 858.452.6800 or e-mail <a href=mailto:sales@profitline.com>sales@profitline.com.

You may also be interested in...

Comments and Discussions

Add a Comment or Question

Search Comments

First Prev Next

What is the license?
oyvind5151 10-Nov-15 6:40

What is the license of the example code for this Code Project article ?
We need to clarify what is the license, before we can use this example code in a commercial software product.

Reply · Email · View Thread

License of this article

tomailvenky 4-Oct-13 2:50

Hi gtamir,
I have gone through the article and saves the tons of my work,
Please let me know under which license category this article falls,
Can I use code in commercial product without modifying your source code(By mentioning your name & url in top of code in comment as reference).
Waiting for valuable reply,

Regards,
Venkat.

Reply · Email · View Thread

My vote of 5
williechen 5-Feb-13 8:27

It's works on windows7!

Reply · Email · View Thread

Why 25 MSI files?
PhilDWilson 28-Feb-11 12:33

Quote "Creating an MSI file for each service and then manually running Windows Installer for 25 services on each machine is a terrible burden on my IT OPS team."

Then why do it?

1) There no rule that says one MSI file per service.

2) Visual Studio is the only tool that insists on ServiceInstaller classes. Thw Wimdows Installer has built-in support for installing and configuring services that most install tools use.

Thousands of setups install multiple services with one install, one MSI file, and have done so long before Visual Studio .NET came along. MSI setups pre-date Visual Studio setup projects.

Reply · Email · View Thread

Alternative - online services
Kikoz68 26-Jan-10 12:29

You can use online services like Installer.CodeEffects.com to create installers for Windows services in several minutes

Reply · View Thread

Sample using MSI
alhambra-eidos 16-Sep-08 6:19

Please, any sample code about install Windows Services using MSI.

Or sample for create Setup Project, for XCOPY and install windows services.

Thanks in advance. Great article.

AE

Reply · Email · View Thread

anybody else getting this error?
thompsonson2 16-Oct-07 11:35

An exception occurred during the uninstallation of the NexusPortal.ScriptedInstaller installer.
System.NullReferenceException: Object reference not set to an instance of an object.
An exception occurred while uninstalling. This exception will be ignored and the uninstall will continue. However, the a
pplication might not be fully uninstalled after the uninstall is complete.

any info would be much appreciated!
cheers,
Matt

Reply · Email · View Thread

Calling from VS Deployment Project
thompsonson2 26-Sep-07 6:47

With a Visual Studio Deployment Project you can set up a Custom Action to call the installer class on Install, Commit, Rollback and Uninstall. I'm looking to get the Install and Uninstall working with the changes in this very useful article.

I've got it to work with the install, the parameters are passed and the service is installed correctly. I'm getting an error when the uninstall is being run. (The ever useful Object is not set to an Object error...). I've added a couple of lines to the OnBeforeUninstall and Uninstall methods to write to a text file. What i'm finding is that OnBeforeUninstall appears to be called and finishing then the error happens (as the Uninstall text file isn't written to).

Anybody know what happens after the OnBeforeUninstall and before the Uninstall is called?

Thanks,
Matt

Reply · Email · View Thread

Configuring Installed Windows Service to a User/Network account Through Code
naguo 11-Sep-07 3:31

Hi,
I already have a windows service installed using installutil.
I cant change the C3 code for that windows service.

But, what I need now is to write a custom c# code that enables me to modify the logon credentials to a User/Network account for any windows service already installed through installutil.
I have been trying to do this, but I think it cannot be done without using class ServiceProcessInstaller, which means I need to modify the c# code of my windows service, which I dont want to.

Any idea, how to do it?

Thanks

naguo

Reply · Email · View Thread

How to run the installed Service
chinkou2006 6-Apr-07 4:50

After I installed the Service in my C# file , but how to run it in the C# file

Reply · Email · View Thread

Re: How to run the installed Service
gtamir 6-Apr-07 10:36

After you install the service, it is available in the 'services' section of the computer management console. Select your new service and click on the 'start' button in the upper toolbar or choose 'start' from the properties list.

Reply · Email · View Thread

Re: How to run the installed Service
chinkou2006 6-Apr-07 22:53

thanks, but I want to start in the Program, we have add key to the rigistry , like this HKEY_CURRENT_USER \ Software \ Microsoft \ Windows \ CurrentVersion \ RunServices，和HKEY_LOCAL_MACHINE \ SOFTWARE \ Microsoft \ Windows \ CurrentVersion \ RunServices。

Reply · Email · View Thread 5.00/5 (1 vote)

Re: How to run the installed Service
thompsonson2 6-Sep-07 5:50

you could always try

NET START <SERVICENAME>

from the command line.

or get your hands on PSService from www.sysinternals.com (now owned by MS)

not sure about doing it programatically. sure there's a COM interface though as i think you can do it from VBScript.

Edit: encoded the < and > as it was treating SERVICENAME as HTML!

Reply · Email · View Thread

Many Thanks!
andrewpw02 29-Mar-07 5:03

This is very useful, I've looked just about everywhere for a solution but couldn't find one. It saved me creating around 20 installsets! I'm sure I will use this again in the future, keep up the good work!

Big Andy

Reply · Email · View Thread

DisplayName missing
surfzoid 21-Mar-07 12:21

I sugest to yu, to add the folowing line (i'm in VB), after the description part of install :

'Default service description
rkService.SetValue("Description", mServiceDescription)
'Default display name
rkService.SetValue("DisplayName", mServiceInstaller.ServiceName)

because at this time, i have some trouble to have interact with desktop, after install of the service i have the checkbox checked in the service propertie tab, but my service doesn't display msgbox, if i uncheck this intarct desktop option, press apply button, check again and press apply again then restart the service, my service display msgbox fine, so after i had a look in the registry, i see displayname added by Microsoft .....
But it wasn't solve my probleme to add it in the install code of my service .....

Reply · Email · View Thread

Problem on using this cs script
jfcarolea 7-Mar-07 8:35

Hello,

I'm not a developper and I'd like to use this script for installing a service in windows 2003 Server with DotNet SDK 2.0. Can you explain to me how the .cs is called by the .bat because when I use it to install a local system service I have always the "Set Service Login" msgbox.

ex: the lines in my .bat
REM Service directory
SET SERVICE_HOME=c:\install\ServiceWindowsPPA

REM EXE ServiceName
SET SERVICE_EXE=Service.Parametrage.exe

REM .NET Directory
SET INSTALL_UTIL_HOME=C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727

set PATH=%PATH%;%INSTALL_UTIL_HOME%

ECHO %PATH%

CD %SERVICE_HOME%

echo Installing Service...
installutil %SERVICE_HOME%\%SERVICE_EXE% /account=localsystem

echo Done.

Reply · Email · View Thread

Great help! 2.0 version?
hahpah 9-Feb-07 9:52

😎 I found your article and code a great help. I needed to alter the c# code for 2.0. The bat files are the same, but the c# is much shorter. Would you like the

code? Please advise as to where/how to submit it. Thanks again,

Jim Harper

Reply · Email · View Thread

Re: Great help! 2.0 version?
fpdeguzma78 23-Jun-08 2:37

hello Jim,
do you still have a copy of the c# code for 2.0?
can i have a copy of it?
thank you very much!
have a nice day.

Reply · Email · View Thread

Good Article, But....
Techno_Dex 15-Dec-06 10:34

The article is very good. I have a questions and comments. I don't follow the change you made to the InfoPath value in the registry. You mention that this is the line used to run the Service and other have suggested the Service commandline switches are stored here also. Can you point to any documentation about the commandline switch (particularly the -s) used in your article? Now for the comments.
As the article was written in 2004 I'm assuming the use of .NET 1.1, so some items in the ProjectInstaller may have been fixed since then. In .NET 2.0 the ServiceInstaller Instance has a Description property which can be set in the constructor without forcing the use of the Registry change you have shown. The ServiceInstaller also has a DisplayName property which you don't show in code. From testing, I have found that if the Display Name is not set, then the ServiceName is used (and displayed in the MMC snap-in). If the DisplayName is set to a default value in the constructor, then it too must also be changed via a parameter in the InstallUtil.exe commandline, otherwise an Exception is thrown that the Service Name is already in use. This may also effect the parameter you use in your -s of the "InfoPath" value but I can't confirm at this point without documentation.

Reply · Email · View Thread

How to use different configuration settings
arkey75444 26-Sep-06 11:42

This is an excellent article. I need to install multiple instances of the same assembly.I have App.config file for database connection and other parameters. Now how do i use this config file to connect to diffent database for each instance of the service.
If i can make it happen, it will be really great.

Reply · Email · View Thread

What's my name?
swoopy_g 25-Oct-05 15:47

As in the article, I have a situation where several services are installed from one exe. When a ServiceBase object is constructed, it is supposed to set the ServiceName property. Is there any way to determine what the service name (the one the SCM has) is within each instance of the exe?

Thanks!

Reply · Email · View Thread

Re: What's my name?
sierra498 20-Apr-06 9:51

I spent ages on this question for my service. Using Regedit, go to the entry for your service and look at the image path. You can add arguments to the end of this path to pass startup info to your service. In your service constructor you can use System.Environment.GetCommandLineArgs() to get the arguments.

Reply · Email · View Thread

Re: What's my name?
Rukaiya_m 24-Nov-06 3:29

Can you pls elaborate on your solution. I'm undergoing the same scenario.. of having multiple instance of services from the same exe. And have a need to know the Service Name dymnaically. CAn you pls provide me the details of your solution. Thanks!!!!

Rukaiya

Reply · Email · View Thread

Re: What's my name?
Rukaiya_m 24-Nov-06 5:53

Ok, got it. I did as per your suggestions to change the imagePath in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<your service name> and it did work!!! Thanks.

Reply · Email · View Thread

Installer Class & Service Exe in different Assembly.
Anonymous 19-Oct-05 8:06

Hi,
I want to install the service using the installer class. Is it possible to have this installer class in different assembly other than the service itself?

Thanks
Santaji GArwe

Reply · View Thread | Edit · Delete

Refresh

1 2 3 Next »

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Permalink | Advertise | Privacy | Terms of Use | Mobile
Web04 | 2.8.171205.1 | Last Updated 12 Nov 2004

Sélectionner une langue

Layout: fixed | fluid