AD04

Creating a Windows Service using SAS® 9 and VB.NET

David Bosak, COMSYS, Kalamazoo, MI

ABSTRACT

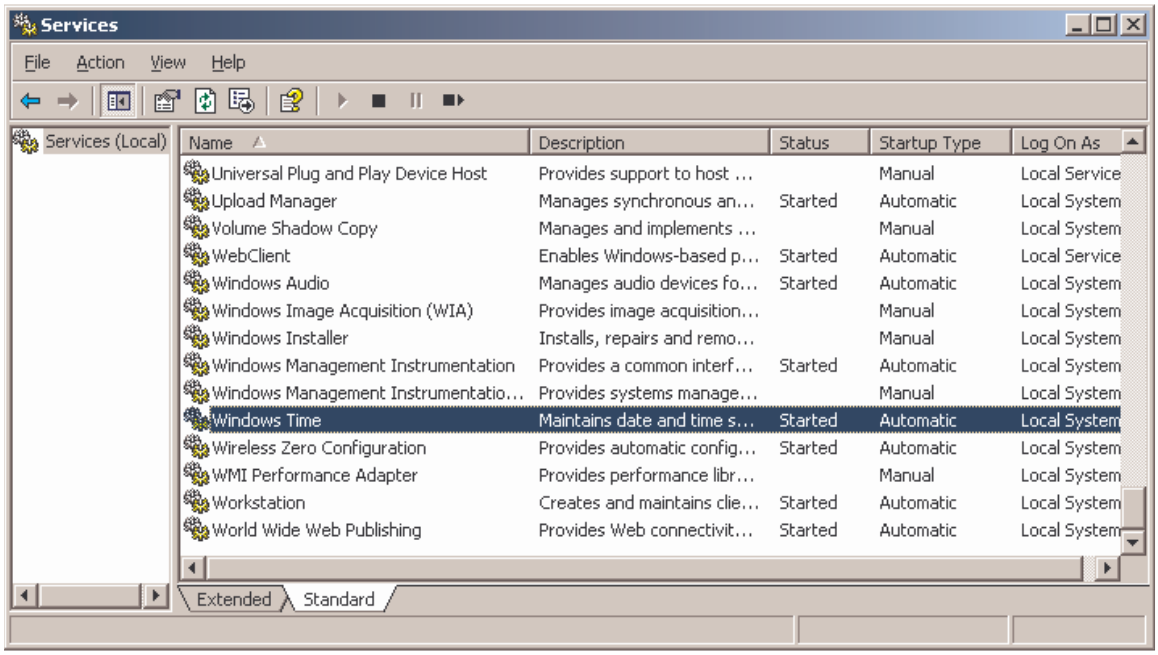
This paper describes how to create a Windows service using SAS 9 and VB.NET. VB.NET is used as a wrapper to SAS. The VB.NET executable is registered with the Windows Services Manager. You are then able to call SAS programs from the service.

INTRODUCTION

The latest version of Visual Basic allows you to easily create a Windows Service. The latest version of SAS allows you to easily integrate with Visual Basic. Therefore for the first time it is now possible to create a Windows Service using SAS. The purpose of this paper is to show you how to do it.

What is a Windows Service?

A Windows Service is an application that runs in memory all the time. A Windows Service has no user interface. Windows Services are used for applications that run continuously in the background. Sometimes the service performs an activity on a scheduled basis; like every day, or every hour. Sometimes the service performs an activity in response to an event; like a user interaction, or an action by another application.



The Windows services on your computer are located in Administrative Tools -> Services. If you open the Services manager, you will see a list of the services available on your computer. The Windows Time Service is an example of a service created by Microsoft. This service keeps the clock on your computer synchronized with the clock on the domain controller. At a periodic interval, this service will request the current time from the domain controller and update the local clock. In this manner, all the clocks on all the computers in the domain will stay synchronized.

Windows services are useful for applications that you want to run on demand, invisibly, and automatically. A service can be set to start automatically when the machine boots or reboots. If a service is set to start automatically, it will start no matter which user is logged in, or if no one is logged in.

A service runs under a designated user account (normally a system account). It can access any files and perform any operations according to the permissions granted to that account. Frequently a service has administrative level permissions. Therefore the service is able to perform operations that a normal user cannot perform.

A Windows service also runs independent of a particular user session, but has access to all open sessions. That is, when a service starts, it runs in the system session. When a user starts his or her own session, the service can interact with that user session, and respond to events initiated by the user.

A Windows service can also respond to other applications that run independent of a user session. Examples of such applications are web servers, email servers, and message queues. For instance, a service could be written to monitor a message queue for a particular message. When that message is received, the service could perform an action in response. The service might also send a return message saying what action was performed.

As you can see, a Windows Service is a very useful type of application. Before SAS 9, it was not possible to create a sophisticated Windows Service with SAS. SAS 9, in conjunction with VB.NET, make this type of application not only possible, but relatively easy to implement.

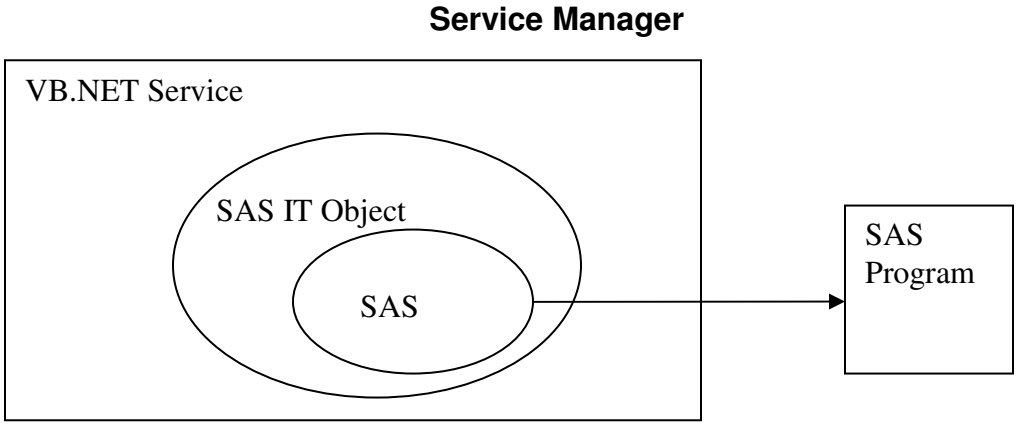
How do you create a Windows Service with SAS?

Creating and testing Windows Services require that you are an Administrator on your machine. So the first step in creating a Windows Service is to get local admin rights.

Once you have local admin rights, the basic steps for creating a Windows Service using SAS are as follows:

1) Write and test the SAS program you wish to make a service
2) Write a VB.NET Windows Service to manage the service
3) From the VB.NET service, create a SAS object
4) From the SAS object, submit your SAS program

The following diagram describes how the primary components of the system are related:



As shown above, the VB.NET service is a wrapper for the SAS object, which is a wrapper for SAS, which executes the SAS program. The entire system is loaded into the Windows Services Manager, which is part of Windows.

The VB.NET service handles the basic service events:

- **OnStart:** The OnStart event fires when the service is started.
- **OnStop:** The OnStop event fires when the service is stopped.
- **OnPause:** The OnPause event fires when the service is paused.
- **OnContinue:** The OnContinue event fires when then the service is continued, following a pause.
- **OnShutdown:** The OnShutdown event fires when the service is running, and then stopped.
- **OnPowerEvent:** The OnPowerEvent fires when the system is powered down while the service is running; for instance, if the machine is rebooted.

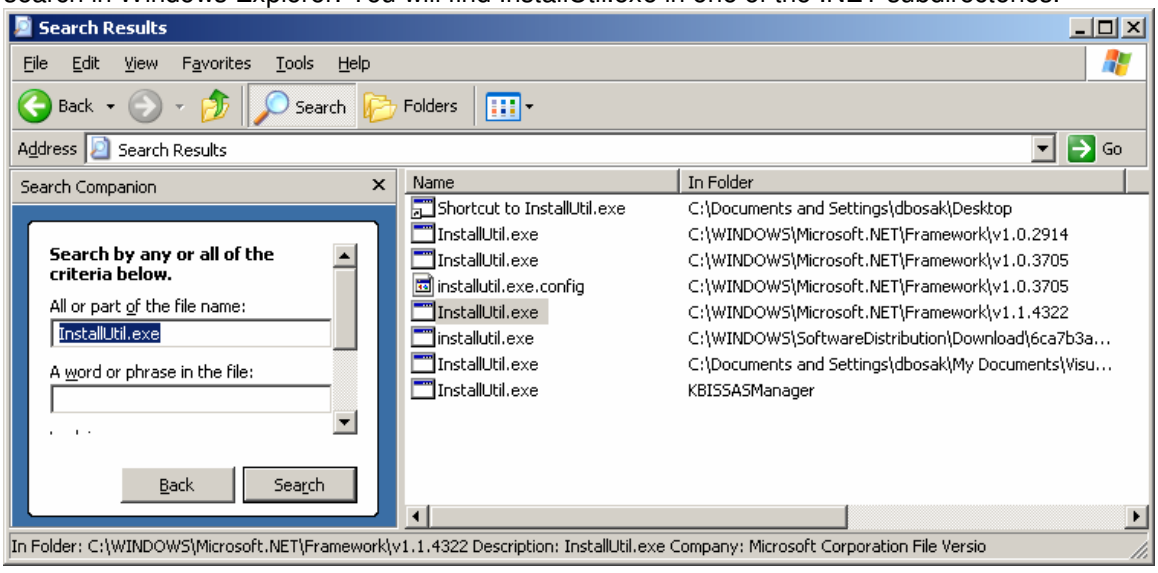
The VB.NET service creates a SAS Object using SAS Integration Technologies. The Windows subset of Integration Technologies is included as part of the SAS 9 foundation. The reason Integration Technologies is included as part of the SAS 9 foundation, is because SAS Enterprise Guide is written in VB. Enterprise Guide uses Integration Technologies to communicate with SAS. Thus, the only SAS license you need to create a Windows Service is SAS/Base.

The inclusion of the Windows subset of Integration Technologies into the SAS/Base is a huge benefit to SAS application developers. It allows us to create VB front ends for SAS by exposing an Application Programming Interface (API) for SAS. The SAS API allows you to call methods, set properties, and generally manipulate SAS programs. The SAS API also allows us to create SAS Services.

Once the service is created, you will register the service with the Services Manager, and start the service. The service is registered with the Services Manager using a utility called InstallUtil.exe.

InstallUtil.exe

InstallUtil.exe is a program from Microsoft that allows you to register and un-register services. The program is part of the Visual Studio.NET development environment, and is also provided with the .NET framework. If you have the .NET framework installed on your machine, perform a search in Windows Explorer. You will find InstallUtil.exe in one of the .NET subdirectories.



InstallUtil.exe has a simple command line syntax:

InstallUtil [/u] *ProgramName*

For instance, if you wrote a service named *MySASService.exe*, and the service was located in a directory called SASProject on the root C drive, you could register the service as follows:

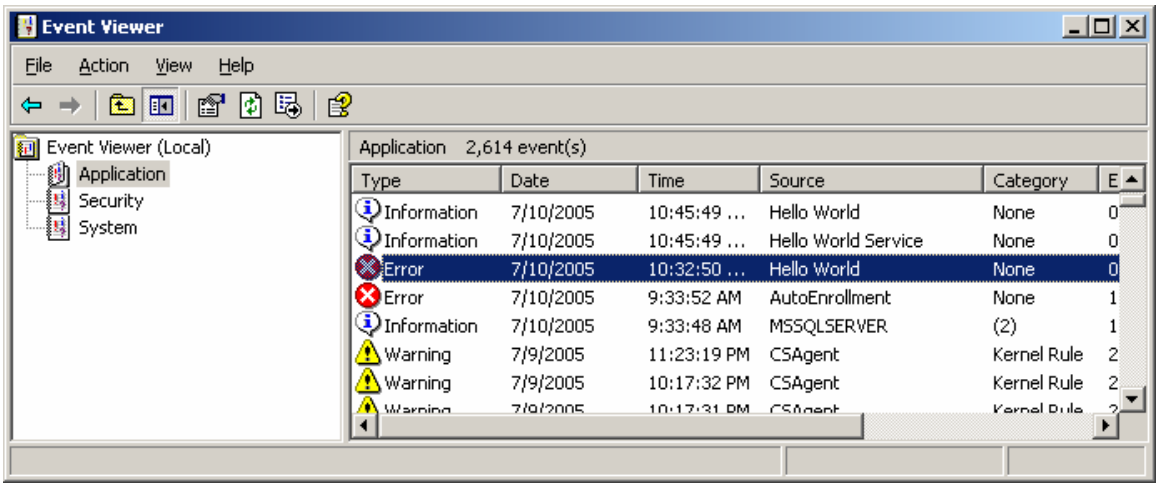
InstallUtil c:\SASProject\MySASService.exe

This operation will load the service into the Windows Services list, and allow you to start and stop the service. The optional /u switch is for un-registering services.

After you attempt to register any service, open Windows Services and double check that the application appears in the services list. Also test that the service starts without errors.

Checking for Service Errors

Since a service has no user interface, errors are typically sent to the Windows Event Log. The Event Log is a common repository of messages created by applications installed on the machine. You can view the event log by going to Administrative Tools -> Event Viewer.



The Event Viewer displays three different logs: Application, Security, and System. The messages your service generates should all go to the Application log.

There are three main types of Event Log message: Information, Warning, and Error. If your service starts without error, it will send an Information message to the Event Log. If your service has an error on startup, it will send an Error message to the Event Log. When you write your custom service with VB.NET, you will be able to send your own custom error messages to the Event Log. This will allow you and your system administrators to monitor the status of your service, and troubleshoot problems.

Example: Hello World Service

To understand how to create a service, let's create a simple "Hello World" service. The Hello World service will create a SAS object that will submit a simple SAS program that writes "Hello World" to the SAS log.

Here is the SAS program:

```
proc printto log='c:\hello.log';
run;

data _null_;
%put hello world;
run;
```

Now let's turn this program into a service. To turn it into a service, take the following steps:

1) Create a VB.NET Service Project
2) Add an Installer Class (see Microsoft Services Documentation)
3) Add a reference to System.ServiceProcess
4) Add a reference to the SAS Workspace manager
5) Create a SAS Workspace object
6) Create a SAS Language Service object
7) Submit the SAS program from the Submit method of the Language Service Object
8) Compile the executable and register it with the Service Manager using InstallUtil.exe

For this example, we will create a service called "Hello World Service" using a VB.NET project called "Hello World Project". The important source code statements follow. The complete source code can be found in Appendix A:

```
    Dim SASWorkspace As SAS.Workspace
    Dim SASLanguageService As SAS.LanguageService
    Dim ProgramString As String

    On Error GoTo ErrorHandler

    'First create a SAS Workspace
    SASWorkspace = New SAS.Workspace

    'Then create a SAS Language Service from the workspace
    SASLanguageService = SASWorkspace.LanguageService

    'Concatenate the program string
    ProgramString = "proc printto log='c:\hello.log';" & vbCrLf & _
                    "run;" & vbCrLf & _
                    "data _null_;" & vbCrLf & _
                    "%put hello world;" & vbCrLf & _
                    "run;"

    'Submit the program from the Language Service
    SASLanguageService.Submit(ProgramString)

    'Send a success message to the Event Log.
    appLog.WriteEntry("Hello World Service", "Hello World Service
completed.", EventLogEntryType.Information)
```

This code is an example of using the Integrated Object Model (IOM) to run SAS. IOM gives you considerable control over SAS from a language like VB.NET. See the Integration Technologies documentation for more information on the methods and properties exposed by IOM.

Now compile the VB.NET code and register the exe with the Services Manager using InstallUtil.exe. Once the service is loaded into the Services Manager, you can start it. If

successful, you will find a SAS log named “Hello.log” on the root C drive, and a message in the Event Log, indicating that the program completed successfully. If something goes wrong, an error message will be sent to the Event Log.

You can hard code the SAS program into the VB.NET code, as in the above example. However, this makes it difficult to update the code. A more effective technique is to write a small program that includes a larger, more complicated program. This technique gives you the ability to modify and expand the larger SAS program without recompiling the VB.

Possible Use of SAS Services

By combining VB.NET and SAS, there are number of very interesting and useful types of services you could create. Here are some examples:

- **SAS/Share Service:** Create a service to start SAS/Share automatically when the system boots.

- **SAS Web Statistics:** Create a service to monitor a web server, and provides statistical usage reports on a daily basis.

- **SAS Email Scanning Service:** Create a service to monitor an incoming email folder. You could use SAS text mining to calculate frequencies on certain text strings. Or you could use SAS to build an intelligent email reader to read the email, and send an appropriate email response.

- **SAS Data Processing Service:** Create a service to monitor a directory for an input file, and process that file as soon as it is placed in the directory.

- **Remote SAS Server:** Create a service to monitor a message queue for an instruction from another machine. The message body could contain the location of an input dataset and a program to run on it.

Additional Considerations

The services mentioned above are much more sophisticated than our Hello World project. Here are some additional considerations needed to create a sophisticated service:

- **Using an Internal Timer:** Services are frequently written to respond to an event like a user action, or the creation of a file. To respond to an event, the service usually contains a timer. The timer allows the service to check at intervals whether or not the event has taken place. If the event has taken place, the service can respond in some way. Many services will require a timer as part of the VB.NET module.
- **SAS Object Pooling Service:** Some services may create multiple SAS objects simultaneously. When this occurs, there is a danger that too many SAS sessions will be created at the same time. Too many simultaneous SAS sessions can overload the server. To avoid overloading the server, SAS provides an Object Pooling Service. The Object Pooling Service will only allow a defined number of SAS objects to be created at one time. Once all of the objects in the pool are used, a request must wait in line until a SAS object is free.
- **Multi-threading and Asynchronous Calls:** When you submit a SAS program from your VB.NET service, the submit call is synchronous. A synchronous call means that the VB.NET code stops executing until the SAS program completes. In some cases, you may want the VB.NET code to continue executing. This type of execution is known as an asynchronous call. To call SAS asynchronously requires that you create the SAS object

in its own thread. Fortunately, VB.NET has some very nice ways of doing this. See your VB.NET documentation to learn how to create a multi-threaded service.

- **Error Handling and Notifications:** Any service must have a very robust error handling and notification system. You must remember that the service will be running all the time, there is no user interface, and there are a host of things that could go wrong to break the service. If the service breaks and there is no robust error handling or notifications, no one will know that the service is broken for several hours or perhaps days. Therefore it is advisable to have a carefully planned error handling and notification strategy. The strategy could include detailed messages to the Event Log, email notifications, and a special service log with regular entries to indicate that the service is still running.

Conclusion

In conclusion, it is now possible to build a Windows Service using SAS 9 and VB.NET. This capability opens up the door to many new kinds of SAS applications. A service is created using VB.NET as a wrapper to a SAS Integration Technologies object, which then calls SAS. The only license requirements needed to write a service are VB.NET and SAS/Base.

For any questions regarding these techniques, please contact the author using the information provided below.

ACKNOWLEDGMENTS

Thanks to Kevin Kramer from COMSYS for help with many of the technical aspects of this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

David Bosak
Principal Consultant
COMSYS Center of Excellence
5278 Lovers Lane
Kalamazoo, MI 49002

Email: dbosak@comsys.com
Phone: 269-344-4100 ext 536

Appendix A

```
Imports System.ServiceProcess

Public Class Main

  Inherits System.ServiceProcess.ServiceBase

  Private appLog As New System.Diagnostics.EventLog


  Protected Overrides Sub OnStart(ByVal args() As String)
    Dim SASWorkspace As SAS.Workspace
    Dim SASLanguageService As SAS.LanguageService
    Dim ProgramString As String

    On Error GoTo ErrorHandler

    'First create a SAS Workspace
    SASWorkspace = New SAS.Workspace

    'Then create a SAS Language Service from the workspace
    SASLanguageService = SASWorkspace.LanguageService

    'Concatenate the program string
    ProgramString = "proc printto log='c:\hello.log';" & vbCrLf & _
                    "run;" & vbCrLf & _
                    "data _null_;" & vbCrLf & _
                    "%put hello world;" & vbCrLf & _
                    "run;"

    'Submit the program from the Language Service
    SASLanguageService.Submit(ProgramString)

    'Send a success message to the Event Log.
    appLog.WriteEntry("Hello World Service", "Hello World Service
completed.", EventLogEntryType.Information)
    Exit Sub
ErrorHandler:
    'Send a failure message to the Event Log.
    appLog.WriteEntry("Hello World Service", "Hello World Service
failed.", EventLogEntryType.Error)
  End Sub

  Protected Overrides Sub OnStop()
    ' Send a stop message to the Event Log
    appLog.WriteEntry("Hello World Service", "Hello World Service
Stopped.", EventLogEntryType.Information)

  End Sub

End Class
```