Laying the Foundation

Let’s start this chapter in a simple, straightforward way. A Windows Workflow Foundation (WF) or Windows Communication Foundation (WCF) service cannot be used until it is hosted. Every WF or WCF service must be hosted in a process known as a host process (roughly speaking, you can think of this as a Windows executable program). The important point is that WF and WCF services cannot run by themselves. A single host process can host multiple services, and the same service can reside in multiple hosts.

Reviewing the Methods for Hosting a Service

There are four methods you can use to host a WF or WCF service. You should be familiar with all of them, so that you can choose the method best-suited for your application. The four methods include the following:

*Self Hosting*: This method requires that the developer provide the host process. This host process can be a Windows Forms application, a console application, or even a WPF rich client application. This host process must be running before the service can receive messages. For the service to start receiving messages, you will need to place the code for the service inside the managed application code, define an endpoint for the service either in code or through a configuration file, and then create an instance of ServiceHost. At this point, you will need to call the open method on the ServiceHost object to create and open a listener for the service. After the service finishes receiving messages, you close the service session by calling `System.ServiceModel.Channels.CommunicationObject.Close` on the ServiceHost. In most cases, this will occur when the application closes.

Two kinds of self-hosting occur more frequently than others: hosting a service within a console application and within a rich client application. Hosting a service within a console application is useful when debugging, trying to get trace information, and stepping through a round-trip service call. This proves useful during the development phase, enabling you to make modifications easily, as well as insert additional error checking and handling. This scenario also allows you to move the application around quickly to test different deployment scenarios.

The second common type of self-hosting occurs in rich client applications. These applications can host services that allow the application to communicate with other services to exchange information. This kind of service also allows other applications to communicate with it by providing a means for other clients to connect to and share information with it.
**Managed Windows Service.** This method of hosting a service is similar to self hosting; however, in this case, you host the service in a managed Windows Service (previously known as a NT Service). Note that the developer must also write the host code when using this method. The advantage of a Windows Service over other self-hosting scenarios is that you can start a Windows Service and listen automatically. The Windows Service becomes both a Windows Service and a WCF service once the implementation code inherits from the `ServiceBase` class and a WCF service contract interface. As when using the self-hosting method, you create and open the ServiceHost; the difference here is that you do so inside the Windows Services `OnStart` overridden method, and you close the ServiceHost within the `OnStop` overridden method. As with all Windows Services, an installer class that inherits from the Installer class must also be implemented to allow the Windows Service to be installed by the `InstallUtil.exe` tool and controlled by the service control manager (SCM) for Windows Services.

This method of hosting, which is available in all versions of Windows, provides the option of hosting a long-running WCF service, outside of IIS, but without the automatic message activation. The operating system manages the lifetime of a Windows Service, so it isn’t tied to the lifetime of the applications that use the Windows Service. Also, these Windows Services don’t have the services that come with hosting in IIS, as you’ll see in the next section.

**IIS Hosting.** IIS hosting is integrated with ASP.NET and thus uses the features that ASP.NET provides. First, you don’t need to write any hosting code to take advantage of this approach. Second, you can find several services already provided. These services include process recycling, idle shutdown, process health monitoring, and message-based activation; however, you should keep in mind that the message-based activation included with IIS hosting works only with the HTTP protocol.

All of the services provided by IIS have made it the preferred host for web service applications that require high availability and scalability. IIS has been available for many years, has matured over the course of many versions, and is included in all Windows operating systems. Unfortunately, people often forget that different versions of IIS come installed on the different versions of the operating system. These differences can lead to different behavior and different functionality. One such example revolves around message-based activation. When working with IIS 5.1 on Windows XP, IIS blocks self-hosted WCF services on the same computer from using port 80 to communicate. When working with IIS 6, IIS provides the ability for hosted services to run in the same worker process (implemented as an Application Pool) as other applications. In this case, WCF and IIS 6 (and IIS 7) use the `HTTP.sys` component. This means they can now share port 80 with other self-hosted services running on the same machine. The `HTTP.sys` component was eventually separated from IIS, and it is now a standalone component that is used by IIS, but which can also be used by your application.

**Windows Process Activation Services (WAS):** This is a new process-activation method that Microsoft released with Windows 2008; it is also available with Windows Vista and Windows 7. WAS has a similar activation method to what is available through IIS hosting. Unlike IIS, WAS hosting has no dependency on the HTTP protocol. WAS provides message-based activation over other protocols supported by WCF, including TCP, MSMQ, and named pipes.

IIS 7 uses WAS to accomplish several things. First, it enables message-based activation over HTTP. Second, WCF components can plug into WAS to provide activation for the other protocols that WCF supports.