Literature Review

Windows Services

Definition

Microsoft Windows services, also known as NT services, help user to create long-running executable applications running in the background as long as the Windows session is live. Windows services can be started, paused and stopped automatically whenever the PC boots up or otherwise. Since these services do not operate with a user interface, it makes them ideal to run on servers or mainframes. These services act as long running scripts executing commands. Windows services can also be run on a specific User account. They can be managed by employing Windows Service Manager. Since they run on the Operating System level, the services must have proper security clearances.

Basic Features

* + All services must be installed on the System or User level before being started.
  + Windows Services run in a different security context from any other program
  + .NET framework does not allow the services to interact with the interactive sessions of the User.
  + A service can exist in one of three basic states: running, paused or stopped. A service can exist in its running state indefinitely until it is either stopped or paused or until the computer shuts down.

Types

There are two broad categories of Windows Services. Services that are the only service in a process are assigned Win32OwnProcess whereas the ones that share a process with one or more services are assigned Win32ShareProcess.

Architecture

Any Windows service application inherits from System.ServiceProcess.ServiceBase class. The service can utilize the following methods in order to articulate the basic functionality of a background process:

* OnStart
  + Actions to be taken when the service starts. It can either be a call to another method or can contain the business logic of the application.
* OnPause
  + Actions to be taken when the service is paused either manually or programmatically.
* OnStop
  + Actions to be taken when the service is stopped either manually or programmatically.
* OnContinue
  + Actions to be taken when the service restarts from after being paused.
* OnShutdown
  + Actions to be taken when the PC is shutting down, if the service is running then.
* OnCustomCommand
  + Actions to be taken when the service returns a custom command.
* OnPowerEvent
  + Actions to be taken when a power management event is received, such as a low battery or suspended operation.

The ServiceBase class’s Run method must also be called in order to initiate the process of the service. In order to be able to install the service and furthermore run it, the service class must also inherit System.ServiceProcess.ServiceProcessInstaller. By overriding its methods the program can add executable classes that are used further. Services can be run directly from this class by calling appropriate post installation event handlers.

The service is installed onto the system by a .NET utility called InstallUtil.exe . This executable file uses the classes created above and loads the service onto the system. If not left to be manual, the service starts after the installation as coded in the post installation event handers.

Windows Communication Foundation

Definition

Windows Communication Foundation or WCF is a framework for designing services that can communicate with other processes or services by sending asynchronous data among themselves. WCF services can be hosted in a variety of processes. For most common uses, WCF services are hosted in a Windows service or under Internet Information Services (IIS).

WCF Terms

Other concepts and terms used in the WCF documentation include the following.

**message**

A self-contained unit of data that can consist of several parts, including a body and headers.

**service**

A construct that exposes one or more endpoints, with each endpoint exposing one or more service operations.

**endpoint**

A construct at which messages are sent or received (or both). It comprises a location (an address) that defines where messages can be sent, a specification of the communication mechanism (a binding) that described how messages should be sent, and a definition for a set of messages that can be sent or received (or both) at that location (a service contract) that describes what message can be sent.

A WCF service is exposed to the world as a collection of endpoints.

**application endpoint**

An endpoint exposed by the application and that corresponds to a service contract implemented by the application.

**infrastructure endpoint**

An endpoint that is exposed by the infrastructure to facilitate functionality that is needed or provided by the service that does not relate to a service contract. For example, a service might have an infrastructure endpoint that provides metadata information.

**address**

Specifies the location where messages are received. It is specified as a Uniform Resource Identifier (URI). The URI schema part names the transport mechanism to use to reach the address, such as HTTP and TCP. The hierarchical part of the URI contains a unique location whose format is dependent on the transport mechanism.

The endpoint address enables you to create unique endpoint addresses for each endpoint in a service or, under certain conditions, to share an address across endpoints. The following example shows an address using the HTTPS protocol with a non-default port:

HTTPS://cohowinery:8005/ServiceModelSamples/CalculatorService

**binding**

Defines how an endpoint communicates to the world. It is constructed of a set of components called binding elements that "stack" one on top of the other to create the communication infrastructure. At the very least, a binding defines the transport (such as HTTP or TCP) and the encoding being used (such as text or binary). A binding can contain binding elements that specify details like the security mechanisms used to secure messages, or the message pattern used by an endpoint. For more information, see [Configuring Services](http://msdn.microsoft.com/en-us/library/ms733830(v=vs.110).aspx).

**binding element**

Represents a particular piece of the binding, such as a transport, an encoding, an implementation of an infrastructure-level protocol (such as WS-ReliableMessaging), or any other component of the communication stack.

**behaviors**

A component that controls various run-time aspects of a service, an endpoint, a particular operation, or a client. Behaviors are grouped according to scope: common behaviors affect all endpoints globally, service behaviors affect only service-related aspects, endpoint behaviors affect only endpoint-related properties, and operation-level behaviors affect particular operations. For example, one service behavior is throttling, which specifies how a service reacts when an excess of messages threaten to overwhelm its handling capabilities. An endpoint behavior, on the other hand, controls only aspects that are relevant to endpoints, such as how and where to find a security credential.

**system-provided bindings**

WCF includes a number of system-provided bindings. These are collections of binding elements that are optimized for specific scenarios. For example, the [WSHttpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.wshttpbinding(v=vs.110).aspx) is designed for interoperability with services that implement various WS-\* specifications. These predefined bindings save time by presenting only those options that can be correctly applied to the specific scenario. If a predefined binding does not meet your requirements, you can create your own custom binding.

**configuration versus coding**

Control of an application can be done either through coding, through configuration, or through a combination of both. Configuration has the advantage of allowing someone other than the developer (for example, a network administrator) to set client and service parameters after the code is written and without having to recompile. Configuration not only enables you to set values like endpoint addresses, but also allows further control by enabling you to add endpoints, bindings, and behaviors. Coding allows the developer to retain strict control over all components of the service or client, and any settings done through the configuration can be inspected and if needed overridden by the code.

**service operation**

A procedure defined in a service's code that implements the functionality for an operation. This operation is exposed to clients as methods on a WCF client. The method can return a value, and can take an optional number of arguments, or take no arguments, and return no response. For example, an operation that functions as a simple "Hello" can be used as a notification of a client's presence and to begin a series of operations.

**service contract**

Ties together multiple related operations into a single functional unit. The contract can define service-level settings, such as the namespace of the service, a corresponding callback contract, and other such settings. In most cases, the contract is defined by creating an interface in the programming language of your choice and applying the [ServiceContractAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicecontractattribute(v=vs.110).aspx) attribute to the interface. The actual service code results by implementing the interface.

**operation contract**

An operation contract defines the parameters and return type of an operation. When creating an interface that defines the service contract, you signify an operation contract by applying the [OperationContractAttribute](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationcontractattribute(v=vs.110).aspx) attribute to each method definition that is part of the contract. The operations can be modeled as taking a single message and returning a single message, or as taking a set of types and returning a type. In the latter case, the system will determine the format for the messages that need to be exchanged for that operation.

**message contract**

Describes the format of a message. For example, it declares whether message elements should go in headers versus the body, what level of security should be applied to what elements of the message, and so on.

**fault contract**

Can be associated with a service operation to denote errors that can be returned to the caller. An operation can have zero or more faults associated with it. These errors are SOAP faults that are modeled as exceptions in the programming model.

**data contract**

The descriptions in metadata of the data types that a service uses. This enables others to interoperate with the service. The data types can be used in any part of a message, for example, as parameters or return types. If the service is using only simple types, there is no need to explicitly use data contracts.

**hosting**

A service must be hosted in some process. A *host* is an application that controls the lifetime of the service. Services can be self-hosted or managed by an existing hosting process.

**self-hosted service**

A service that runs within a process application that the developer created. The developer controls its lifetime, sets the properties of the service, opens the service (which sets it into a listening mode), and closes the service.

**hosting process**

An application that is designed to host services. These include Internet Information Services (IIS), Windows Activation Services (WAS), and Windows Services. In these hosted scenarios, the host controls the lifetime of the service. For example, using IIS you can set up a virtual directory that contains the service assembly and configuration file. When a message is received, IIS starts the service and controls its lifetime.

**instancing**

A service has an instancing model. There are three instancing models: "single," in which a single CLR object services all the clients; "per call," in which a new CLR object is created to handle each client call; and "per session," in which a set of CLR objects is created, one for each separate session. The choice of an instancing model depends on the application requirements and the expected usage pattern of the service.

**client application**

A program that exchanges messages with one or more endpoints. The client application begins by creating an instance of a WCF client and calling methods of the WCF client. It is important to note that a single application can be both a client and a service.

**channel**

A concrete implementation of a binding element. The binding represents the configuration, and the channel is the implementation associated with that configuration. Therefore, there is a channel associated with each binding element. Channels stack on top of each other to create the concrete implementation of the binding: the channel stack.

**WCF client**

A client-application construct that exposes the service operations as methods (in the .NET Framework programming language of your choice, such as Visual Basic or Visual C#). Any application can host a WCF client, including an application that hosts a service. Therefore, it is possible to create a service that includes WCF clients of other services.

A WCF client can be automatically generated by using the [ServiceModel Metadata Utility Tool (Svcutil.exe)](http://msdn.microsoft.com/en-us/library/aa347733(v=vs.110).aspx) and pointing it at a running service that publishes metadata.

**metadata**

In a service, describes the characteristics of the service that an external entity needs to understand to communicate with the service. Metadata can be consumed by the [ServiceModel Metadata Utility Tool (Svcutil.exe)](http://msdn.microsoft.com/en-us/library/aa347733(v=vs.110).aspx) to generate a WCF client and accompanying configuration that a client application can use to interact with the service.

The metadata exposed by the service includes XML schema documents, which define the data contract of the service, and WSDL documents, which describe the methods of the service.

When enabled, metadata for the service is automatically generated by WCF by inspecting the service and its endpoints. To publish metadata from a service, you must explicitly enable the metadata behavior.

**security**

In WCF, includes confidentiality (encryption of messages to prevent eavesdropping), integrity (the means for detection of tampering with the message), authentication (the means for validation of servers and clients), and authorization (the control of access to resources). These functions are provided by either leveraging existing security mechanisms, such as TLS over HTTP (also known as HTTPS), or by implementing one or more of the various WS-\* security specifications.

**transport security mode**

Specifies that confidentiality, integrity, and authentication are provided by the transport layer mechanisms (such as HTTPS). When using a transport like HTTPS, this mode has the advantage of being efficient in its performance, and well understood because of its prevalence on the Internet. The disadvantage is that this kind of security is applied separately on each hop in the communication path, making the communication susceptible to a "man in the middle" attack.

**message security mode**

Specifies that security is provided by implementing one or more of the security specifications, such as the specification named [Web Services Security: SOAP Message Security](http://go.microsoft.com/fwlink/?LinkId=94684). Each message contains the necessary mechanisms to provide security during its transit, and to enable the receivers to detect tampering and to decrypt the messages. In this sense, the security is encapsulated within every message, providing end-to-end security across multiple hops. Because security information becomes part of the message, it is also possible to include multiple kinds of credentials with the message (these are referred to as *claims*). This approach also has the advantage of enabling the message to travel securely over any transport, including multiple transports between its origin and destination. The disadvantage of this approach is the complexity of the cryptographic mechanisms employed, resulting in performance implications.