**Chapter 2**

**Technologies Used**

**1 Microsoft Windows Services**

**1.1 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 while it is in operation. 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 without explicitly showing anything on the display. 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. A User level service has lesser clearances than a System level service.

**1.2 Basic Features**

* All services must be installed on the System or User level before being started. Unlike normal computer programs, Windows services cannot be compiled and run.
* 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 as their station of operations are different.
* 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.

**1.3 Types**

There are two broad categories of Windows Services. Services that are the only service in a process are assigned Win32OwnProcess category whereas the ones that share a process with one or more services are assigned Win32ShareProcess category. In terms of the kind of jobs done, Windows services can be of the following types,

* **Adapter service**
  + For a hardware device that needs its own driver.
* **File System Driver service**
  + File system driver or a kernel device driver.
* **Interactive Process service**
  + Rare breed that can only communicate with the desktop.
* **Kernel Driver service**
  + For low-level hardware devices such as a hard disk.
* **Recognizer Driver service**
  + Used during start up to determine if certain file systems exist on the system

**1.4 Architecture**

All Windows service applications inherit from System.ServiceProcess.ServiceBase class. They can utilize the following methods in order to fulfill 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 install the service and furthermore run it, the service class must also inherit System.ServiceProcess.ServiceProcessInstaller. By overriding its methods, this program adds 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.

**2 Windows Communication Foundation**

**2.1 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 either be hosted in a variety of processes or can sustain as self-hosted services. For most common uses, WCF services are hosted in a Windows service or under Internet Information Services (IIS). The behavioral aspects of the service are written in metadata, like XML.

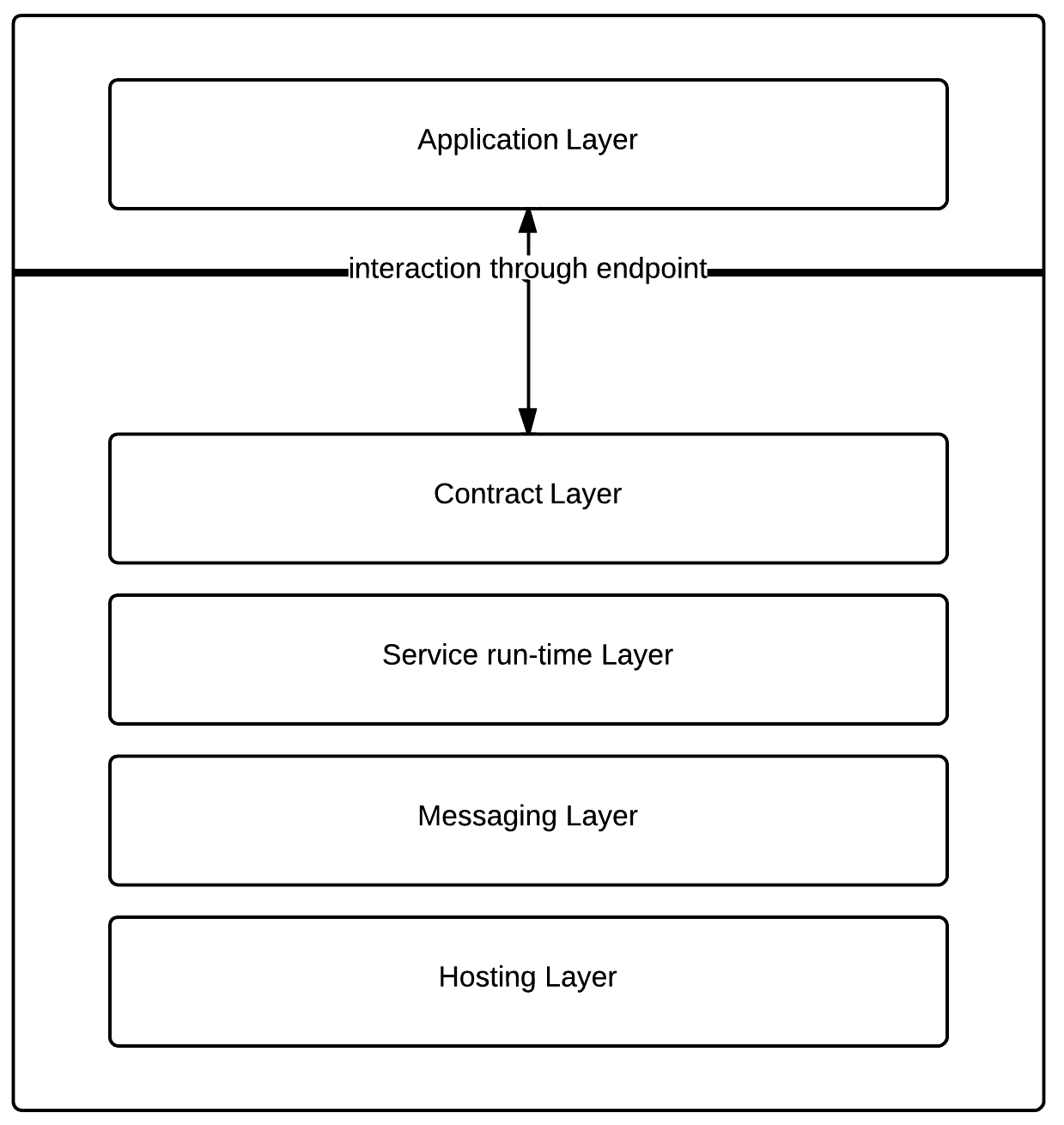
**2.2 Terminology**

* **message**
  + self contained unit of data with a header and a body
* **endpoint**
  + construct that assists in sending or receiving messages. 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 via these endpoints. Each endpoint can have a function associated on its end.
* **address**
  + the location where messages are received and is specified as Uniform Resource Identifier or URI. This can be integrated with HTTP or TCP/IP protocols. A WCF service must have a unique endpoint address in order for it to function calmly.
* **binding**
  + definition of how the endpoint communicates with the world. It is constructed by a set of components called binding elements that "stack" on each other in order to produce a communication infrastructure. A binding defines the transport protocol; security protocol and the encoding used being used in the data communication. There are four major System-Provided bindings viz.
    - *BasicHttpBinding*
      * HTTP protocol binding used majorly in letting ASMX-based web services communicate with the WCF service.
    - *WSHttpBinding:*
      * interoperable binding used in letting web services of all types communicate with the WCF service.
    - *NetNamedPipeBinding:*
      * connects with WCF endpoints on the same machine.
    - *NetMsmqBinding:*
      * connects with WCF endpoints using queued message connections.

* **binding element**
  + component of the communication stack in the binding. Describes the communication protocol, encoding and the security context of the address.
* **behaviors**
  + component that controls the various run-time aspects of the service. Behaviors are grouped according to the scope. Common behaviours affect all components whereas specific ones do not.
* **service operation**
  + procedure that defines functionality of the service. It can have multiple arguments and can return data of any type.
* **service contract**
  + ties together operations into one bed. It defines the namespace and other service level settings. It is a required component of the class.
* **operation contract**
  + defines the parameters and return type of each service operation in the service using System.ServiceModel.Web namespaces like WebInvoke. Return type, method to do so, body style of the returned value and the URI extension are mentioned.
* **hosting**
  + WCF service must be hosted in some process or can be self hosted. The host controls the lifetime of the service. Services can be self-hosted or managed by Windows Services, IIS etc.
* **client application**
  + program that exchanges messages with the WCF service. This can either be a web page or another service for that matter.
* **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 security specifications.
* **transport security mode**
  + confidentiality, integrity, and authentication are provided by the transport layer protocols. Despite being robust, “man-in-middle” attacks can be a worry.

**2.3 Architecture**

An application interacts with the service by sending or receiving data through endpoints provided by the WCF service. The WCF service then initiates its response by looking up at the Contract Layer. This layer has contracts for the Service, Operations and Messages as detailed in the previous sub-section. It describes the behaviors of the service in terms of protocols to be used and the encoding to be followed. The Service run-time Layer describes the actions that are to be taken when this service is operational. It acts as a watchdog and provides contingency measures in case of any failure. The Message Layer is composed of channel stacks. There exist two types of stacks, viz., transport stack and protocol stack. The former stack is involved in reading and writing messages received/sent and to encode them respectively. The protocol stack on the other hand acts as the implementer of the message processing protocols. It reads or writes additional headers to the messages. The Hosting Layer describes the way WCF service is hosted on the system. It has nothing to do with the communication and thus acts as a cup to hold the WCF service.



**2.4 Integration with web pages**

WCF services’ endpoints can be accessed via AJAX requests from any web application. Having initially only set to be compatible with SOAP (Simple Object Access Protocol) WCF is now in hooks with REST (Representational State Transfer) architectures.

For instance, as shown in the figure X, the WCF service exposes one of its operations via an endpoint. Now once the service is installed and is running, AJAX calls from a web page can send and receive data from the WCF service. The service takes in the data and sends appropriate response. The response is converted to JSON and used by the web page.

