Chapter 4

1. Implementation

1.1 Service implementation

In order to capacitate the system described in the previous chapter, the author implemented a multi class model explained in this section. The system is defined under the namespace of “eFacebook.Mediator.Source”. The WCF service is declared under the class FocalService, which implements Mediator while the service hosting the same is named FocalWindowsService. The ProjectInstaller class helps in creating installation files for the Windows service. The class CommunicationManager handles the intra-service communication pathway.

* Mediator
  + The interface that outlines the method “receiver”. This method is designed to receive the AJAX calls. The method receiver takes in two string arguments named “stuff” and “action”. The former variable takes in the HTML data being transferred to the service while the later is involved in noting down the action that is supposed to be taken by the service. It is to be noted that the “action” converts into the filename, which is being filled with “stuff” under a particular folder.
  + The Operation Contract as mentioned in the section X is also utilized. Here the contract mentions the kind of method this function behaves as. In this case, the author has used “POST” as it is better than using “GET” when sending or receiving sensitive data. The method is logically an invoke operation and can be called by any WCF REST programming model. The response as well as received body is chosen to be “wrapped”. This means that when the service or client serializes parameters and returns values to a message, it writes them within infrastructure-provided XML elements and is wrapped. Furthermore, the contract also mentions the URI template of the method. This is the suffix added to the endpoint for ease of differentiation from other methods. In this case, the URI template is “/form”.
* ConnectionManager
  + This class acts as the messenger between the FocalService and the FocalWindowsService classes. The variable “content” is used for storing the value of “action” received by the method “receiver”. The string variable “sendback” is used to read in the data from the file created by the client-side standalone application and pass on to the WCF service. The Boolean variable, “status” is used for storing the readability of the variable, “sendback”. Only when set to true, can the data read be sent to the webpage wanting the same.
* FocalService
  + This class implements Mediator and is the pioneer WCF service class. The method “receiver” is the only function in the class. It sets the “path” variable (“…/efb/in”) for the service and concatenates the “action” variable received from the call as the filename. It also sets the value of the variable action to a “content” from ConnectionManager. It then utilizes the built in functionality of System.ServiceProcess.ServiceController’s property viz. ExecuteCommand to access FocalWindowsService’s OnCustomCommand method.
  + The method receiver also accomplishes writing variable “stuff’s” contents to the file specified by “path”. The method keeps polling for the data to be sent back i.e. “sendback” using the Boolean variable of “status”. When the later becomes true, it returns the value as string in XML format.
* FocalWindowsService
  + This class implements System.ServiceProcess.ServiceBase class in order to act as the pioneer Windows service implementation. It contains a public “serviceHost” variable of type System.ServiceModel.ServiceHost. This type provides the hosting capability to any service and is needed in this case to host the FocalService class.
  + Its protected method, “OnStart” is used for detailing out the tasks to be implemented when the service is started. First things first, this method initiates the FocalService by calling “ServiceHost.Open” method. It then creates a System.IO.FileSystemWatcher variable. This watcher listens to the folder “…/efb/out” for any “FileChange” events to happen. As mentioned in the section X, the author clearly chose event based monitoring technique over polling one. If the event does happen, it enlists the “FileWasChanged” method to act as the executer.
  + The “FileWasChanged” method takes up event and waits for 20 milliseconds before checking if the event was raised by the file whose name is same as the variable “content” from the class ConnectionManager. If so, it reads the contents of the file into ConnectionManager’s “sendback” and toggles the “status” variable to true. It also checks for errors in any file input/output process.
  + The “OnCustomCommand” menthod is supposed to take in requests from ServiceController’s ExecuteCommand. It relays the commands received to their proper handler functions defined in this class.
  + The “makeFiles” method is added in as a debugger tool in order to help the future developers. It adds the current file name being written to by the FocalService and the current time, as a file to another folder called “C:/test/watched”. Its main purpose is to catch the filename and the timestamp of the same.
  + The protected method, “OnStop” is used to stop the execution of the FocalService after the FocalWindowsService has been stopped.
  + The “main” method of the class runs the FocalWindowsService class by calling ServiceBase’s method “Run”.
  + The constructor of the said class is used to set the visible name of the Windows service, as displayed in the Windows Service Manager.
* ProjectInstaller
  + This class implements System.Configuration.Install.Intaller. Its constructor adds in the files required for the InstallUtil.exe to help the system install the Windows service. It also adds in an event handler for after install scenario.
  + The said event handler is implemented by the “ServiceInstaller\_AfterInstall” method. It starts the Windows service once it is installed onto the system.

1.2 Concept flow

The Windows service has been named eFacebookMediatorService. In order to install and uninstall the Windows service using .NET InstallUtil.exe, the author wrote scripts that can be run once the service is downloaded onto the machine in the future. Figure x shows the major flow of the way this operation works.

Once the service is installed using the script talked about above, the ProjectInstaller class’s method “ServiceInstaller\_AfterInstall” starts the same. After that the “main” method of the FocalWindowsService class is executed, running the service completely. The “OnStart” method then runs the FocalService which is now ready to be called through AJAX by any webpage. It also sets the file system watcher and file change event handlers up.

When an AJAX call to the endpoint with the correct URI template as mentioned in the previous sub-section, is made, the “receiver” function gets called. A file of the name “action” and the timestamp is created for debugging purposes. No content is copied there. The method then writes the content to the named file in “…/efb/in” and waits for the data to be sent back. This file is then taken up by the client-side standalone application which reads it and writes modified data into another file of the same name but in “…/efb/out” folder. This folder is being watched by the FocalWindowsService class. It triggers a file- changed event and the handler method (FileWasChanged) is called. This method checks if the said file is actually the one required by comparing its name to the action received previously. If so, the “sendback” variable is set and the “status” flag is set to true. This immediately triggers the “receiver” method’s polling operation and the contents of “sendback” variable are sent back to the calling webpage. All of this happens in a time range of 30 to 70 milliseconds.

1.3 Peripheral implementations

* Metadata for Windows service
  + As mentioned in the section X, for every Windows service, an endpoint is a complete necessity. The author created a file named App.config which describs the binding used, endpoint behaviors and exception handling triggers.
  + The endpoint chosen is <http://localhost:8000/ServiceModelSamples/service> with the URI template having “/form”. This means that all AJAX calls are being made to <http://localhost:8000/ServiceModelSamples/service/form>
  + The webHttpBinding is used to define the behavior of the system in terms of protocols, encoding and transfer channels. Refer to section X for more insights into this concept.
* AJAX implementation
  + An illustration of a common AJAX call is shown in figure X. First, the data to be sent is converted into a JSON object with stuff and action as shown in lines 4 and 5.
  + The JSON object is then sent as a string and the data expected back is marked as XML.
  + The URL mentioned is nothing but the endpoint of the WCF service/FocalService class.
  + If successful, the necessary data is extracted from the XML received and is utilized in the innerHTML of another content. Lines 16 to 21 illustrate this.
  + If unsuccessful, the error caught is shown. Lines 23 to 26 show this.