**Transforming Java programs into Windows Services using Apache Commons Daemon**

***Creating windows service was possible only using MFC programs, since it involved manipulation of registry values and access to system resources. Since Java maintains its platform independence, it does not support platform specific features like running services. Apache Commons Daemon is a small utility to transform a Java program into a Windows service.***

**INTRODUCTION**   
Windows services are programs which start without any user intervention and run until the system shuts down or until explicitly closed by the administrator. These programs do not have any interaction with the desktop. Since these programs run behind the scene, they are called daemons.

All our computers run daemons, whether we notice their presence or not. The printer spooler, the anti-virus, the firewall etc. run as background daemons and interact with the user desktop only when necessity arises. Background programs usually wait for particular condition to occur and swing into action based on the input provided.

Windows service is like an unnamed servant waiting for the errands of his master. The servant waits until the master assigns some work. Otherwise, waiting is the only work for the servant to do. Similarly, Windows services wait for proper circumstances to act, otherwise, they simply wait in a typical while loop.

A service is invoked by the Service Control Manager (SCM), which looks for a start method and a stop method (nothing else). Apache Commons Daemon requires that the service program should have start and stop methods. Any other method can be used to start and stop the service, provided proper option is specified while installing the service. In typical cases, a Java service program has two methods, viz., i)public static void start(String []arg) and ii)public static void stop(String []arg) to start and stop the service.

A service is said to run as long as its start method runs. Hence, the start method of a service usually enters a while loop and begins executing endlessly (unless system is shut down or explicitly stopped). A service does not provide any user interface. It communicates through TCP or UDP sockets or through the Inter Process Communication mechanism.

**WAYS FOR CREATING JAVA SERVICES ON WINDOWS**   
There are many tools available for creating Java services on Windows. Java Service Launcher (JSL)[1], Java Service Wrapper[2],  Apache Commons Daemon[3] and Excelsior Jet for Windows[4] are some of the most common tools. The Apache Commons Daemon is a free software which can be used without any restriction.

The Apache Commons Daemon contains two executable files named prunsrv.exe and prunmgr.exe and is available for download from Apache project site[5]. Out of the two executable files, prunsrv is the program for installing or removing the service and prunmgr is the program for controlling the service and changing its configuration after installation.

**HOW THE SERVICE MECHANISM WORKS?**   
Windows offers a dedicated service manager for handling service processes. The Service Control Manager (SCM) of Windows handles all registered services. The SCM is started by Windows at boot time. SCM looks at all installed services and starts those which are marked for automatic start up. SCM is responsible for starting and stopping services as the need arises.

**CREATING WINDOWS SERVICE**  
Windows services require some entries to be made in the HKEY\_LOCAL\_MACHINE->SYSTEM->CurrentControlSet->services area of the system registry. Manually making these entries is tedious, difficult and meant only for low level programmers. Easy installation of a service requires a small program which can make required entries to start the Java program as a service.

Windows services could be traditionally generated by Visual Studio package. Java always maintained its platform independence and distanced itself from involving too deeply into the specifics of an operating system. Whereas, creation of general purpose server frameworks like Apache Tomcat required an easy way to handle Java services on Windows. Hence, the Apache Software Foundation (ASF) created two programs, one for installation/ deletion of Java service and the other for managing the service. These two programs are now available as a separate utility named Apache Commons Daemon.

**CREATING A JAVA SERVICE**  
To test the use of Apache Commons Daemon, a simple Java program for sending square of the input obtained from the client is written. To make it run as a service, the *start* method runs a while loop listening for clients. The service is said to run until the start method keeps running.

Following code shows the Java service for sending square of a given number.  
  
//SquareServer.java  
1. import java.net.\*;  
2. import java.io.\*;  
3.   
4. public class SquareServer  
5. {  
6. static ServerSocket serverSocket = null;  
7. static boolean running = true;  
8.  
9.   
10. public static void start(String arg[]) {     
11.    Socket client = null;  
12.    try {  
13.       System.out.println("Starting service on IP Address "+  
14.          InetAddress.getLocalHost().getHostAddress()+  
15.          " port No.12345");  
16.       serverSocket = new ServerSocket(12345);  
17.       }  
18.    catch(Exception ex) { ex.printStackTrace(); }  
19.    byte b[] = new byte[256];  
20.    String tmp = null;  
21.    while(running && !serverSocket.isClosed()) {  
22.       try {  
23.          System.out.println("Listening for clients on IP Address "+  
24.             InetAddress.getLocalHost().getHostAddress()+  
25.             " port No.12345");  
26.          client = serverSocket.accept();  
27.        System.out.println("Client from "+client.getInetAddress()+":"+  
28. client.getPort()+" connected at "+(new java.util.Date().toString()));  
29.          int n = client.getInputStream().read(b);  
30.          tmp = new String(b, 0, n).replaceAll(",","");  
31.          double x = Double.parseDouble(tmp);  
32.          double y = x\*x;  
33.          client.getOutputStream().write((""+y).getBytes());  
34.          client.getOutputStream().flush();  
35.          System.out.println("Received "+x+" and sent "+y);  
36.          client.close();  
37.          }  
38.       catch(Exception ex) { ex.printStackTrace(); }  
39.       }  
40.    }  
41.   
42. public static void stop(String arg[]) {  
43.    if(!running) return;  
44.    running = false;  
45. try { serverSocket.close(); } catch(Exception ex) { ex.printStackTrace(); }  
46.    System.out.println("Stopped service");  
47.    }  
48. }

The service code is contained in a file named SquareServer.java. Note that there is no main method in the above code. The start and stop methods are enough to manage the service. For a service, all messages written on standard output and standard error got to log files. Hence, the start method writes a log message stating that the service has started (lines 12, 13 & 14).

The server socket is opened (line 15) on port number 12345 and the start method enters a while loop. It goes on listening for client requests until the Boolean variable named *running*is true in value and the server socket remains open. On accepting a client (line 25), the server reads the input number from the client (lines 29 & 30), calculates its square and sends the result back to the client (line 33).

Messages printed on the standard output and standard error are written into appropriate log files. The service framework of the operating system takes care to capture the appropriate streams and write them into the preconfigured log files. We might specify the name of log file as an argument while installing the service.

The code for stopping the service is contained in the stop(String []arg) method (lines 42 to 47). The stop method checks whether the *running* variable holds *false* value and returns without further processing if it is false (line 43). Otherwise, the value of *running* is set to *false* (line 44) and the server socket is closed (line 45). A log message is written to indicate that the service has been stopped (line 46).

Compile SquareServer.java using the command *javac SquareServer.java*. The resulting class file (*SquareServer.class*) is sufficient for installing the service using the Apache Commons daemon.

**INSTALLING THE SERVICE USING APACHE COMMONS DAEMON**  
  
The tools needed for installing the service are available from the URL   
http://www.apache.org/dist/commons/daemon/binaries/1.0.5/windows/[5]. The binary executable and other files are contained in a zip file named *commons-daemon-1.0.5-bin.zip*. After extracting the contents of the zip file, copy the files named *prunsrv.exe* and *prunmgr.exe* into the folder containing the source and class files of SquareServer.

The binary file *prunsrv.exe* is renamed to *squareserver.exe* and *prunmgr.exe* is renamed to*squareserverw.exe*. This makes the installation of the service easier. It is useful to learn some of the most commons options (I am not listing all) that can be used to control the installation parameters for the service. Table 1 shows the most common parameters used to control the service installer.

**Table 1 Commons options for service installation using Apache Commons Daemon**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Option** | **Description** |
| 1. | //IS//<service name> | Install a service of given name |
| 2. | //DS/<service name> | Delete a service of given name |
| 3. | //ES//<service name> | Edit parameters of the installed service |
| 4. | --Description=<value> | Description about the service. This description appears in the Service Control Manager window. |
| 5. | --Install=<value> | The value is the name of executable file used as service starter. In our case, the service starter is the same as the installer, i.e., squareserver.exe. |
| 6. | --Classpath=<value> | The folder containing the class files needed for the service should be provided as the value for the parameter. |
| 7. | --StartClass=<Class name> | Name of class file used as service. |
| 8. | --StopClass=<Class name> | Name of class file used for stopping the service. In many cases, same class file is used for starting and stopping the service. Hence, the value provided for --tartClass and --StopClass may be the same. |
| 9. | --StartMethod=<name> | Name of the method used to start the service. The method should have (*String []arg*) as argument. In normal cases, the method is named *start.* |
| 10. | --StopMethod=<name> | Name of the method used to stop the service. The method should have (*String []arg*) as argument. In normal cases, the method is named *stop.* |
| 11. | --LogPath=<value> | Name of directory used for logging service messages. |
| 12. | --StartMode=<value> --StopMode=<value> | The argument for these two options can be any one of *jvm, java or exe*. If the option is *jvm*, the service is launched from the same process which is used for starting the service, by accessing*jvm.dll*. If the option is *java*, the service is launched in a separate process by invoking *java.exe* available in the JAVA\_HOME environment variable. If the argument is *exe*, service is run as a separate process using the specified binary executable file. |

For installing SquareServer, log into an administrative account, open command window, change to the directory containing the files *SquareServer.class, squareserver.exe*and*squareserverw.exe*and type the following command to install the service.

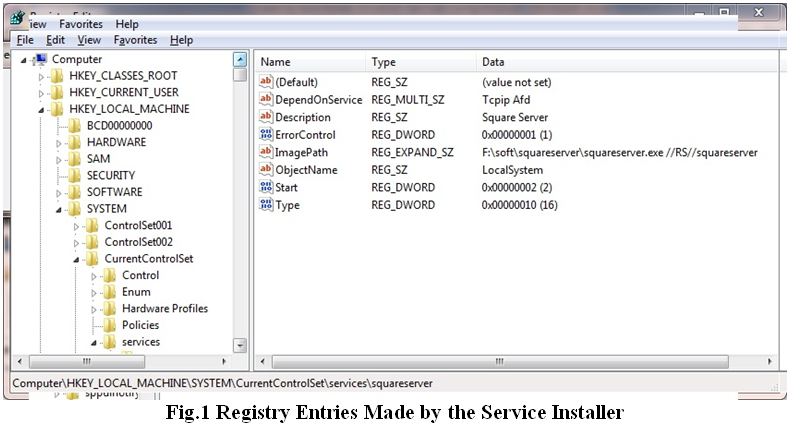
*squareserver.exe //IS//squareserver  --Description="Square Server" --Install=%CD%\squareserver.exe --Jvm=auto --Classpath=%CD% --StartMode=jvm --StartClass=SquareServer --StartMethod=start --StopMode=jvm --StopClass=SquareServer --StopMethod=stop --LogPath=%CD%\logs --StdOutput=auto --StdError=auto --Startup=auto*

If you are not logged into an administrative account, create a batch file named *install.bat* with content typically reflecting the following statements:  
  
*F:*  
*cd soft\squareserver*  
*squareserver.exe //IS//squareserver  --Description="Square Server" --Install=%CD%\squareserver.exe --Jvm=auto --Classpath=%CD% --StartMode=jvm --StartClass=SquareServer --StartMethod=start --StopMode=jvm --StopClass=SquareServer --StopMethod=stop --LogPath=%CD%\logs --StdOutput=auto --StdError=auto --Startup=auto*  
*pause*

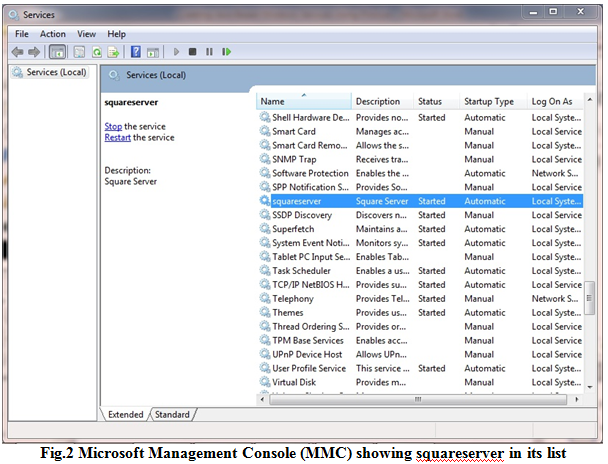
The first line changes to the drive containing the server files (which is F: in my case). The second line changes to the directory containing the service files (which is soft\squareserver) in my case. The third line is the service installation command. The pause at the end is provided to allow inspection of the result of installation and find out whether any error occurred during installation.

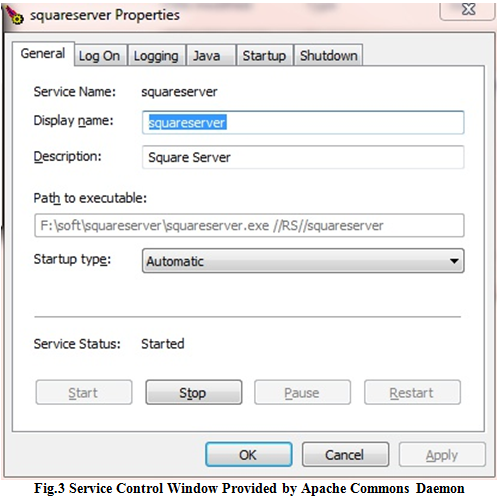
After creating the batch file, open the service folder using My Computer (or issue the command*explorer .* from the command window). Right click on the *install.bat* file and choose *Run as administrator* option. After reading the message displayed by the service installation program, press enter to close the new command window.

After running this installation command, necessary entries are made in the Windows registry as shown in Fig.1. Service installer creates an entry carrying the name of service in the key structure*HKEY\_LOCAL\_MACHINE->SYSTEM->CurrentControlSet->Services*. Compare the options provided during installation and the corresponding registry values.



**MANAGING THE SERVICE**  
  
After installing the service, it can be controlled from the service management console which can be opened using Control Panel->System and Security->Administrative Tools->Services or by issuing the command *mmc services.msc*in a command window or after pressing Windows+R button to run a command.

Typical service controls for square server are shown in Fig.2. Our program has now joined the elite club of background processes working in zero visibility mode. The service can be started stopped and managed like any other service provided by windows. Another alternative to control the service parameters is to run the command *squareserverw.exe* from the directory containing the binaries (Fig.3).   
  


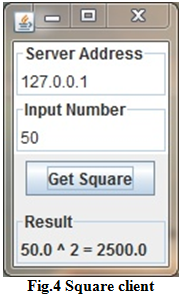
  
  
With this, squareserver is ready to serve the world on portion No.12345 of your Windows machine. We have to ensure that the service remains started during the testing process.

**TESTING THE SERVICE USING A JAVA CLIENT**  
To test the working of the service, a Java client has been written. It sends input number to the server and receives the computed results from the server. The following code shows SquareClient.java:

//SquareClient.java  
1. import java.io.\*;  
2. import java.net.\*;  
3. import java.awt.\*;  
4. import javax.swing.\*;  
5. import java.awt.event.\*;  
6.   
7. class SquareClient extends JFrame implements ActionListener  
8. {  
9. JTextField serverAddress = new JTextField("127.0.0.1");  
10. JTextField input = new JTextField();  
11. JButton b = new JButton("Get Square");  
12. JLabel result = new JLabel(" ");  
13.   
14. public SquareClient() {  
15.    super("Square Client");  
16.    JPanel p = new JPanel(new GridLayout(4,1)),  
17.       p1 = new JPanel(new FlowLayout());  
18.    p.add(serverAddress);  
19.    p.add(input);  
20.    p1.add(b);  
21.    p.add(p1);  
22.    p.add(result);  
23.    serverAddress.setBorder(BorderFactory.createTitledBorder("Server Address"));  
24.    input.setBorder(BorderFactory.createTitledBorder("Input Number"));  
25.    result.setBorder(BorderFactory.createTitledBorder("Result"));  
26.      
27.    input.addActionListener(this);  
28.    b.addActionListener(this);  
29.      
30.    this.getContentPane().add(p);  
31.    this.pack();  
32.    this.setDefaultCloseOperation(this.EXIT\_ON\_CLOSE);  
33.    this.setVisible(true);  
34.    }  
35.   
36. public void actionPerformed(ActionEvent ae) {  
37.    String txt = input.getText();  
38.    if(txt == null)  
39.       return;  
40.    byte b[] = new byte[256];  
41.    txt = txt.trim().replaceAll(",","");  
42.    try {  
43.       double in = Double.parseDouble(txt);  
44.       Socket s = new Socket(serverAddress.getText(), 12345);  
45.       s.getOutputStream().write(txt.getBytes());  
46.       s.getOutputStream().flush();  
47.       int n = s.getInputStream().read(b);  
48.       txt = new String(b, 0, n).replaceAll(",","");  
49.       double res = Double.parseDouble(txt);  
50.       result.setText(in+" ^ 2 = "+res);  
51.       } catch(Exception ex) { ex.printStackTrace(); }  
52.    }  
53.   
54. public static void main(String arg[]) {  
55.    new SquareClient();  
56.    }  
57. }

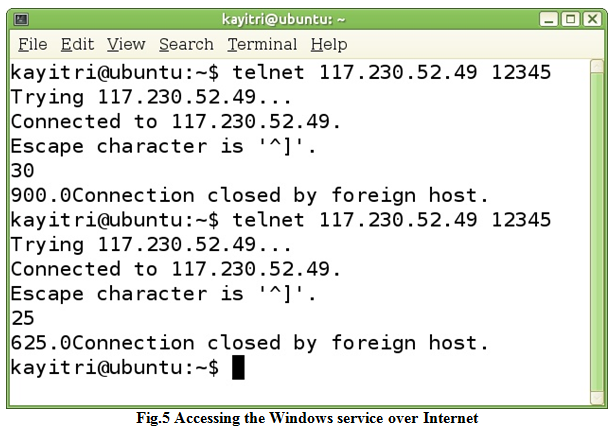
SquareClient displays two text fields (one for IP address and the other for input number), one button to initiate communication with server and a label to display the result obtained from the server. Lines 36 to 52 show the handling of communication between server and client.

Input provided by the user is written to the server (lines 45 and 46) and the result from the server is read (line 47). The result is displayed in the label at the bottom of the window (lines 48 to 50). The SquareClient is shown in Fig.4. In the present test, the squareserver running on local host (127.0.0.1) was accessed to get the results.

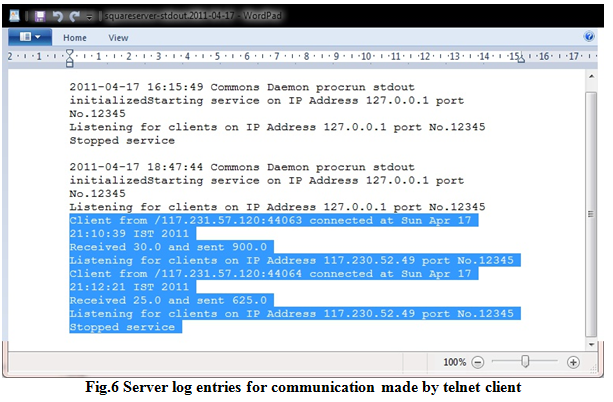


**TESTING THE SERVICE USING TELNET CLIENT**  
The service was accessed from over the Internet from an Ubuntu client using the telnet command. To access the service, the IP address of the server machine (which was 117.230.52.49, found using a small Java program which use  
  
s java.net.InetAddress.getLocalHost().getHostAddress() method to find the IP address.

The command telnet 117.230.52.49 1234 (IP address and port number) was issued from command terminal of the Ubuntu machine, which displayed a message confirming successful connection. The connection was established twice and numbers 30 and 25 were passed as input data. The results obtained from the server are shown in Fig.5. The connection termination message is displayed just following the result obtained from server, since the server did not append any new line character to the result (to keep it easy for parsing from programs).



Server log for session is shown in Fig.6. Highlighted portion of the log file indicates that the client had IP address 117.231.57.120. The log shows the input received and the result sent back to the client.



**CONCLUSION**  
Apache Commons Daemon provides an easy and cost effective mechanism for creating service programs on Windows platform. Although this article shows a simple illustration of the service handling architecture, Java offers limitless possibilities for creating services on Windows platform.