# Steps to Implementing RMI

1. There are essentially four software parts to implementing RMI. These are:
   - client program (does the request)
   - server program (implements the request)
   - stub interface (used by the client so that it knows what functions it can access on the server side)
   - skeleton interface (used by the server as the interface to the stub on the client side)

2. The information flows as follows: the client talks to the stub, the stub talks to the skeleton and the skeleton talks to the server.

3. **The Client.** The client requires the client program itself and an interface to the server object that we are going to connect to. An interface example is shown below:

```
Public interface WeatherIntf extends java.rmi.Remote {
   Public String getWeather() throws java.rmi.RemoteException;
}
```

Figure 1. The Client Interface.

The important thing to note about the interface it that it defines an interface to the function to be called on the server. In this case, the function to be called is getWeather(). The client program itself is shown in Figure 2. The main points of the client program are as follows. First, to implement RMI, we need to use the "java.rmi" package. This package contains all the guts to implement RMI. In the main function, we first instantiate a Remote object of the type that we want to use. In this case, we are going to remotely connect to the WeatherServer object through the WeatherIntf interface. Once we have the object from the server then we can call its functions like we would as if the object were located on our computer.

```
Import java.rmi.*;

public class RMIdemo {

   public static void main(String[] args) {
      try {
         Remote robj = Naming.lookup("//192.168.0.9/WeatherServer");
         WeatherIntf weatherserver = (WeatherIntf) robj;

         String forecast = weatherserver.getWeather();

         System.out.println("The weather will be " + forecast);
```

```
    } catch (Exception e) {System.out.println(e.getMessage()); }
  }

}
```
Figure 2. The Client Program.

Once the Client and Interface have been written, they have to be compiled into class files.

4. **The Server.** It is interesting to note that the Server program also requires the Interface class as shown in Figure 1. This makes sense since the Server will be implementing the Interface. To use the Server, we need to use the "java.rmi" package and the "java.rmi.server.UnicastRemoteObject" package. The Server IS a UnicastRemoteObject, and is declared as such in the class declaration shown below. This means that the system will be a remote object that can be called from a client. Note that the method that we are going to call from the client is the getWeather() function and it can throw a RemoteException. The interesting function to see here is the main function. To implement RMI, we need to set a security manager with permissions that will allow clients to access functions on this pc from another pc. This is easily done by setting the System Security Manager to a RMISecurityManager. This will allow other pcs to call functions on this pc. Then we create an instance of this class simply so that we can pass the object as an argument to the Naming.Rebind. Rebind is the way a server announces its service to the Registry. The Registry is a program that contains a table of services that can be used on a server by client programs.

```
Import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class WeatherServer extends UnicastRemoteObject implements WeatherIntf {

  public WeatherServer () throws java.rmi.RemoteException {
    super();
  }

  public String getWeather() throws RemoteException {
    return Math.random()>0.5? "sunny" : "rainy";
  }

  public static void main(String[] args) {
    System.setSecurityManager(new RMISecurityManager());
    try {
      WeatherServer myWeatherServer = new WeatherServer();
      Naming.rebind("/WeatherServer", myWeatherServer);
    } catch (Exception e) { System.out.println(e.getMessage()); }
  }
```

```
}
```
Figure 3. The Server Program.

5. **Compiling the code.** So far we have created three java programs: the client, the server and the interface. Each of these must be compiled into class files. This can be done in the IDE or on the command line by using "javac WeatherIntf.java", "javac WeatherServer.java", or "javac RMIdemo.java" for example.

   Once the three class files have been compiled, we now have to create the stub and the skeleton for the network communications between them. The is done by using the rmic compiler and the server program. The following can be entered at the command prompt: "rmic WeatherServer". This will create "WeatherServer_Stub.class" and "WeatherServer_Skel.class" for the client and server respectively.

6. **Running the Program.** The Server needs the skeleton, the interface and the server program class files. The Client needs the Stub, the interface and the client program class files. The best place to put these files is in the Java Runtime Environment directory. On my machine it is: C:\Program Files\JavaSoft\JRE\1.3\lib\ext.

   On the Server side we need to start the registry so that the server functions can be made public to other machines. This is done by starting the rmi registry via typing at the command prompt: "rmiregistry". The registry program will continue to run in this window until CTRL-C is pressed.

   To run the server with the RMISecurityManager, we have to define the permissions that we want to grant clients. We do this via a permit file. The permit file that I used is shown in Figure 4. Basically, I set connect and accept permissions on the socket connections. I also set read permissions on files in the tmp directory just to illustrate, although it is not required in this demo.

```
Grant {
     permission java.net.SocketPermission "*", "connect";
     permission java.net.SocketPermission "*', "accept";
     permission java.io.FilePermission "/tmp/*", "read";
};
```
Figure 4. The permit file.

Now we can start the server and make it use the permit file by typing the following at the command prompt: "java –Djava.security.policy=permit WeatherServer"

The Client can then be started on the client machine by typing the following at its command prompt: "java RMIdemo"

7. **Some things to note.** Note that all that is needed on the Client side is the main client program, and the stub (WeatherIntf_stub). The Server side requires that rmiregistry is running, the main server program is started with the permit file, and the skeleton (WeatherIntf_skeleton).