Remoting in C# using .net

The following tutorial note will give the user a basic framework for developing Remoting applications. The goal is educational usage in “Introduction to Distributed Systems” and therefore issues on e.g. security or exception handling are left untouched in the application.

About Remoting

Remoting is part of the .net framework intended for developing distributed applications. The framework allows for several remote object semantics where MarchallByReference is being exposed in this tutorial. Remoting also allows for several object creation methods e.g. Sever Activated Objects along with a several modes e.g. Singleton which dictates various attributes of the remote objects. For the complete MS sales pitch on Remoting see: http://msdn2.microsoft.com/en-us/library/kwdt6w2k.aspx and http://msdn2.microsoft.com/en-us/library/2e7z38xb.aspx

The Software in the tutorial

The tutorial consists of a server and a client. The Server is defined as the component that offers connections, as in found in the folder called RemotingServer. The client is aptly named RemotingClient is found in the folder by the same name. Both of these are developed in MS visual studio 2005 and is tested in this environment; for all other environments your mileage may vary.

Both the server and the configures the Remoting framework programmatically, where as it may be of use to understand and use

The rest of this tutorial note gives the overall steps in setting up the applications, detailed comments are given in the source code.

The server

The server consists of:

- An interface of the offered object.
- An implementation of the offered object.
- An implementation of the server offering an object.

Interface of the offered object

The interface serves the purpose of forming the basis for a DLL that the client can uses as a type reference. This is required as a security measure such that unknown types not are used. The interface defines two methods:

- void setString(string str)
- string getString()

All other purposes and uses of interfaces are not considered here.

Implementation of the offered object

The server also needs an implementation of the object that it intends to server. Is straight C# code, the only things to remember is that is must inherit from *MarshalByRefObject* and implement the interface.

Implementation of the server offering an object

The server is a command line program, that doesn't require any user interaction. It acquires a tcp channel on a predefined port and registers this with *the ChannelServices* interface as required by the Remoting framework. Finally it registers the offered object as a *WellKnownServiceType* and then goes in to service mode until terminated.

Building the server

Build the RemotingServer solution should result in the following files:

- RemotingServer.exe
- RemotingServerObjectInterface.dll

Besides this there may some debug files if Visual studio is in debug configuration.

The Client

The client consists of a single file:

- An implementation of RemotingClient

Implementation of RemotingClient

The implementation of the client is straight forward C# code, the only caveat is that the client class must have a reference to the RemotingServerObjectInterface.dll otherwise it is not possible to type cast the remote object correctly in the client.

The client acquires its needed resources, i.e. a tcp channel and along with the registration of the channel. Here after it is a simple matter of activating the remote object using the correct host, port and service name.

Exercise 1:

Build the server and client and run to ensure that it operates correctly on your computer.

Exercise 2:

Build a distributed CAT server. The Distributed CAT server offers the following methods:

- `public boolean openFile(String filename)`

Opens a file, returns true if file is successfully opened, false otherwise.

- `public String nextLine()`

Reads and returns the next line from the file. Returns `null` if end-of-file is reached or if no file has been opened.

- `public boolean closeFile()`

Close the file which is currently open. Returns true if the file is successfully closed, false otherwise.

Credits:

Thomas H. Clausens orginal tutorial on JAVA RMI:
http://www.cs.auc.dk/~bnielsen/DS-E06/Java-RMI-Tutorial/