Exercise 11 Distributed COM

In this exercise you are going to learn how to deploy and run a COM object remotely on a different machine. You will use both configuration tools and (optionally) programmatic techniques to access the remote object.

Step 1 - Getting Started

Create a new ATL COM Wizard project in the directory:

```
C:\COM Programming\Exercises\Distributed COM Fundamentals
```

Give the project the name ***Remote*** and make the project an EXE.

Step 2 - Adding the COM Object

Since you are intending to test our server on someone else's machine you need to be careful about naming our server. We suggest that you choose a name that is unique, perhaps by including the last digit of your IP address in the server name. To create the server, make sure you are in class view and then add a simple object control with this unique short name. We will assume you have chosen the server name ***Server50*** in what follows. Select a ***Custom*** interface rather than a ***Dual*** interface because MIDL marshalling will be used in this example.

Add a method to the ***IServer50*** interface called ***Hello*** that takes no parameters and simply displays a message box.

Build the project.

Step 3 - Creating and Registering the Proxy/Stub Code

The MIDL compiler creates all the necessary code for your ***Proxy*** and ***Stub*** but you still have to build and register them. The easiest way to do this is to modify your project settings. So choose the ***Remote*** project and select the settings for ***Custom Build***.

You will find that the wizard has already added settings to register your EXE. Add the following lines:

```
nmake RemotePS.mk
regsvr32 RemotePS.dll
```

When you rebuild your project these lines will build the ***Proxy*** and ***Stub*** from the makefile ***Remoteps.mk*** and then perform the registration.

You should now perform a ***Rebuild, All*** now, and make sure that the proxy/stub DLL is built and registered correctly.

Step 4 - Testing the Server Locally

Add a console-based project to your workspace called ***TestLocal.*** Write a simple test harness in this project to prove your COM object works correctly on the local machine. You can copy the harness below, or write your own. Remember to replace the coclass name and interface name with your own.

```
#include <windows.h>
#include <assert.h>

#import "../Remote.tlb"
using namespace REMOTELib;

int main()
{
	CoInitialize(0);
	IServer50* spIServer50 = 0;

	HRESULT hr;
	hr = CoCreateInstance(__uuidof(Server50),
				0,
				CLSCTX_LOCAL_SERVER,
				__uuidof(IServer50),
				(void**)&spIServer50);
	assert(SUCCEEDED(hr));

	spIServer50->Hello();
	spIServer50 = 0;

	CoUninitialize();
	return 0;
}
```

If you use the above test harness make sure the type library location is correct.

Step 5 - Setting up Security, Access and Launch Permissions

You must set up security, access and launch permissions for our server before you can expect DCOM to work. You will use the ***DCOMCnfg*** utility to set these permissions.

Run ***DCOMCnfg*** and select the ***Remote*** class. If you choose the ***Properties*** option you will be able to set the required permissions. On the ***Security*** tab select custom access and launch permissions. Edit these permissions such that ***Everyone*** and ***System*** are allowed access and launch permission. On the ***Identity*** tab select make sure the ***interactive user*** is selected. Close down ***DCOMCnfg***.

Step 5 – Using DCOMCnfg to Perform Remote Access

In this step, we are going to use DCOM to access our object from a remote machine, but without changing our client.

Copy the server, test client and the proxy/stub DLL to another (the client) computer. The easiest way to do this is to copy the three files to a directory and then to share the directory so that everyone can gain access to it. You will then be able to retrieve the see these files from the client machine.

On the client computer, register the server (using ***Remote /RegServer***). When you have done this, delete the server from the client computer. The purpose of this step was to get the CLSID and APPID registered on the client computer. Having completed this, you will no longer need the server on the client computer.

On the client computer, register the proxy/stub DLL.

Run ***DCOMCnfg*** on the client computer. Select the appropriate server class, and edit its properties. On the location tab, deselect the "Run application on this computer" checkbox. Check the "Run application on the following computer" and enter the **server** machine name in the edit box.

Close down ***DCOMCnfg***. Run your test client on the client computer and a message box will appear on the server machine.

(Optional Extension)

Step 1 - Programmatic access with CoCreateInstanceEx()

All that remains is to write the code to test the COM object remotely. Add a new console based project to your workspace called ***TestRemote***.

Using the local test harness as a guide, write a harness for remoting the COM object. You will need to create ***COSERVERINFO*** and ***MULTI_QI*** structures and pass them to ***CoCreateInstanceEx***. You will also need to change ***CLSCTX_LOCAL_SERVER*** to ***CLSCTX_REMOTE_SERVER***.

The ***COSERVERINFO*** structure will contain the name of the server machine you intend to act as a server (your machine name) and a pointer to the security settings contained in a ***COAUTHINFO*** structure. Use a NULL pointer for the security settings - COM will then use the registry settings you've just set up with DCOMCNFG.

The ***MULTI_QI*** structure will contain your interface pointer. Use this pointer to invoke the ***Hello*** method. Don't forget to initialise the ***COSERVERINFO*** structure correctly! Consult the on-line help for details.

N.B. You must define the pre-processor symbol ***_WIN_32_DCOM*** before you can use ***CoCreateInstanceEx***. We suggest your test harness begins as follows:

```
#define _WIN32_DCOM

#include <windows.h>
#include <assert.h>

#import "../Remote.tlb"
using namespace REMOTELib;

int main()
{
        // ...
```

When you've built your new client, you can copy it to the client machine.

Step 2 – Removing the server information from the client machine's registry

If you run this new test client on the client machine, it will work just as before. So how can we prove that it is using the machine name supplied in ***CoCreateInstanceEx*** and is not simply using the registry.

The easiest way to prove this is to remove the CLSID from the client machine. Using either ***RegEdit*** or ***RegEdt32*** remove the CLSID for your server from the client machine. Now run the new test client again, and it still works!

It works because if the COM runtime sees the CLSCTX_REMOTE_SERVER flag it can pass the request directly to the SCM on the remote machine, and therefore doesn't check the client's registry.

(Optional Extension)

Step 1 - Experimenting with Permissions.

Once you have DCOM running you might like to play around with permissions on the remote machine. Use ***DCOMCnfg*** to change the permissions. If the client doesn't have the correct permissions then ***CoCreateInstanceEx*** will fail. Resetting the permissions should make things work.