

## **DCOM, 3-tier C/S and the Internet**

Using COM Internet Services

by Randy Charles Morin B.C.S. B.Comm.

I guess I threw enough key words into the title to get your attention. You might have asked yourself while reading the title, why would anybody use DCOM over the Internet? Well over the last few years, people have been talking about distributed objects and many of you have implemented this ideology in very controlled environments using CORBA and even DCOM. But I've heard nothing about distributing objects over the Internet. About the closest I've seen is those Java objects that I download to my web-browser. But remember, those objects are not well distributed. The code still runs on my machine, not the server.

This is where Microsoft has positioned COM Internet Services. COM Internet Services allows client programs to call upon COM servers through port 80, IIS and the RPC proxy. Normal DCOM activity occurs on port 135. It is highly likely that some proxy server or firewall will get in the way. So Microsoft decided that it should allow DCOM to be tunneled through normal http and port 80. Unfortunately, this was not easily accomplished as it is likely that between your COM server and it's Internet client, stood a web server listening on that very port. Arrgh! IIS and COM Internet Services to the rescue and now we have DCOM over the Internet.

So how you do it? It's not easy. I know. I tried. Hopefully, if I tell you of my experiences you'll have some better luck. That's my goal, please read on.

### **DCOM**

DCOM or Distribute COM or Distribute Component Object Model, is Microsoft's answer to almost everything these days. DCOM is a protocol for allowing objects created on various platforms and languages to communicate effectively across computer boundaries. A major hurdle in Microsoft's path was porting DCOM to more than Intel/Win32 computer systems. Although DCOM is not proven beyond its Intel roots, it is now available on alternate platforms.

### ***3-Tier C/S***

The 3-tier client-server architecture has become all the rage with IT managers. Everything must be 3-tier. Although most IT managers would hesitate at answering the question "What is the 3-tier client-server model?" most understand that it is the breaking of components into three layers, that is, the user interface, the business logic and the data. The advantage of this approach is a thinner client and increased scalability. It's not my intent to explain the 3-tier client-server model, so I'll leave you with this very brief explanation.

### ***Internet***

So how does DCOM fit into the 3-tier client-server model and the Internet? Until recently, that is Windows NT4-SP4, it didn't. With the release of SP4 came COM Internet Services. Previous implementations of DCOM 3-tier Internet applications required that

all firewalls between the client and server open up a range of ports for DCOM to perform its magic. Unfortunately, most firewall administrators are very touchy about opening up ports, especially if they hear the word DCOM.

COM Internet Services allows DCOM to tunnel through port 80, through IIS and to your backend server. Hurray. If you are not already aware almost all HTML traffic travels over port 80 and because of this, the port is rarely ever blocked by firewalls.

## Requirements

In order to show you how to development applications that uses COM Internet Services, I'll begin by presenting an HTML-based application. We will then extend this HTML-based application to use DCOM and XML with a 2-computer configuration. Then I'll port the application to 3-tier, 3-computer. And lastly, I'll port the application to use port 80 and IIS. Oh, I forgot to tell you that I'm developing an application to maintain a list of favorite web sites.

## Object Model and The Data Layer

Let's start by defining our object model. Well, we have one object in our model, that we'll call "Favorites". The object has two properties, that is "Name" and "URL". I usually move immediately from defining my object model to defining my data layer. I find the data layer is easily derived from the object model. Since we have one object we'll also have one table and listing 1 presents the DDL for that table.

### Listing 1: DDL for Favorites

```
CREATE TABLE favorites
(
    name VARCHAR(255),
    url VARCHAR(255),
    PRIMARY KEY (name)
)
```

Creating data access logic for this small database is also quite easy. Listing 2 has all the data access logic necessary to code our data layer. As you can see, I chose "Ms-SQL Server" as my DBMS and ADO to access the database. My goal is not to introduce ADO so I leave it to you to understand the code.

### Listing 2: The Data Layer code

```
// modify these for your own database
static std::wstring s_driver = L"{SQL Server}";
static std::wstring s_server = L"DEV_NEWS";
static std::wstring s_uid = L"sa";
static std::wstring s_pwd = L"";
static std::wstring s_database = L"Alerts";

void FavoritesImpl::InsertFavorite(std::wstring name, std::wstring url)
{
    std::wstringstream ss;
    ss << L"INSERT INTO favorites (name, url) VALUES ('"
        << name << L"', '" << url << L"') ";
    Automation object;
    object.CreateObject(L"ADODB.Connection");
    {
        VariantList parameters;
        parameters << L"driver=" << s_driver
            << L";server=" << s_server
            << L";uid=" << s_uid
    }
}
```

```

        << L";pwd=" << s_pwd
        << L";database=" << s_database;
    object.Invoke(L"Open", parameters);
}
{
    VariantList parameters;
    parameters << ss.str();
    object.Invoke(L"Execute", parameters);
}
}

void FavoritesImpl::GetFavorites(std::map<std::wstring,
    std::wstring> &favorites)
{
    Automation recordset;
    Automation connection;
    connection.CreateObject(L"ADODB.Connection");
    {
        VariantList parameters;
        parameters << L"driver=" << s_driver
            << L";server=" << s_server
            << L";uid=" << s_uid
            << L";pwd=" << s_pwd
            << L";database=" << s_database;
        connection.Invoke(L"Open", parameters);
    }
    {
        VariantList parameters;
        parameters << L"SELECT * FROM favorites";
        recordset.pdisp = connection.Invoke(L"Execute",
            parameters).pdispVal;
    }
    recordset.Invoke(L"MoveFirst");
    while (!recordset.Get(L"EOF").boolVal)
    {
        std::wstring strName;
        std::wstring strUrl;
        {
            VariantList parameters;
            parameters << L"name";
            Automation field;
            field.pdisp = recordset.InvokeGet(L"Fields",
                parameters).pdispVal;
            VARIANT var = field.Get(L"Value");
            strName = var.bstrVal;
        }
        {
            VariantList parameters;
            parameters << L"url";
            Automation field;
            field.pdisp = recordset.InvokeGet(L"Fields",
                parameters).pdispVal;
            VARIANT var = field.Get(L"Value");
            strUrl = var.bstrVal;
        }
        favorites[strName] = strUrl;
        recordset.Invoke(L"MoveNext");
    }
}
}

```

## Interfaces

Another important part of defining your application is to define the interfaces between your different components. Because I intend to port this code to DCOM later in this article, I wrote the interface (see listing 3) in IDL.

### Listing 3: IDL

[

```

    object,
    uuid(0C7B7B82-D8BA-11D2-BF02-00C04F8B72E7),
    dual,
    helpstring("IFavorites Interface"),
    pointer_default(unique)
]
interface IFavorites : IDispatch
{
    [id(1), helpstring("method InvokeFavorites")]
    HRESULT InvokeFavorites([in] BSTR request, [out] BSTR * response);
};

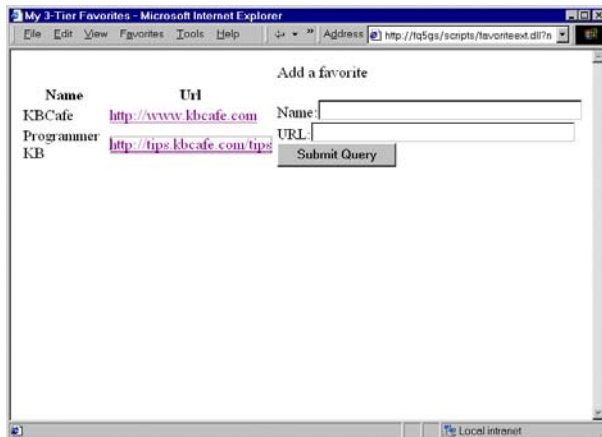
```

As you can see, there is nothing special about the interface. It takes a request string, that is the CGI-GET parameter list received from the HTML-client and returns a response string, that is returned HTML. The parameter list may contain only two parameters, the name of a favorite and the URL of a favorite.

Next I wrote the application logic to implement the above interface for my HTML client and an ISAPI extension that calls the application logic. In the application logic, I simple parse the parameter list and call the appropriate data layer logic to interact with the database. You can see these routines on the source code provided separately. Later in the article, I'm going to rewrite the routines for COM Internet Service use, so I've omitted them here for brevity.

The last thing we have to do is find an NT Server with IIS and SQL Server to install all this code. Don't forget to copy both the ISAPI extension (favoriteext.dll) and the application logic (favorite3tier.dll) to the IIS server. And of course, register the application logic (regsvr32 favorite3tier.dll). Once that's done, presto you have an HTML-based client (see figure 2).

**Figure 1: HTML client**



## ***DCOM client***

That was the easy part. Now we have to convert the HTML client to a DCOM client. I wouldn't suggest using MFC as an Internet client as the install is simply too cumbersome, but for demonstration purpose, I will use MFC.

Once you've put together the UI in MFC you'll have to write a routine to call our application logic COM server. Here 'tis (listing 4).

**Listing 4: Remote Call**

```

void CClientDlg::SendQuery(const std::string &str)
{
    VARIANT response;
    response.vt = VT_BSTR+VT_BYREF;
    BSTR responsebstr = ::SysAllocString(L"");
    response.pbstrVal = &responsebstr;

    VariantList parameters;
    parameters << response << str;
    object.Invoke(L"InvokeFavorites", parameters);
    std::string retval = WideToAnsi(*response.pbstrVal);
    try
    {
        XmlParser parser;
        parser.ParseCanonicalXml(retval);
        if (parser.m_node.m_type == "FAVORITES")
        {
            std::vector<XmlNode>::iterator i =
                parser.m_node.m_children.begin();
            for (int k=0;i!=parser.m_node.m_children.end();i++,k++)
            {
                if (i->m_type != "FAVORITE")
                {
                    continue;
                }
                std::string strName;
                std::string strUrl;
                std::vector<XmlNode>::iterator j = i->m_children.begin();
                for (;j!=i->m_children.end();j++)
                {
                    if (j->m_type == "URL")
                    {
                        strUrl = j->m_value;
                    }
                    if (j->m_type == "NAME")
                    {
                        strName = j->m_value;
                    }
                }

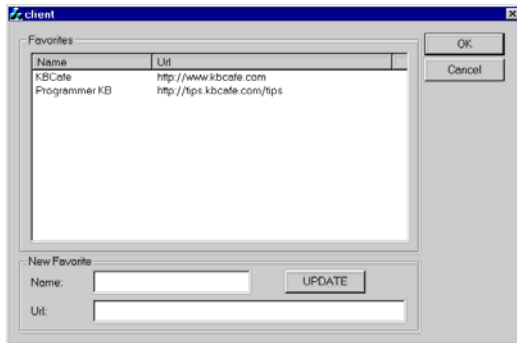
                if (strName.empty() || strUrl.empty())
                {
                    continue;
                }
                m_listcontrol.InsertItem(k, strName.c_str());
                m_listcontrol.SetItemText(k, 1, strUrl.c_str());
            }
        }
    }
    catch(XmlParserException &)
    {
    }
}

```

I've built the XML request before this code is called, so I just send it to our COM server using an IDispatch Invoke helper function.

```
object.Invoke(L"InvokeFavorites", parameters);
```

I then parse the returned XML response and populate our list control. Simple. Figure 2 shows our MFC client in action.

**Figure 2: MFC client**

But wait. Something's wrong here. The application logic (favorite3tier.dll) was running in the same process space as the user interface. This isn't 3-tier is it? Yes it is. Because using a surrogate process, I can then move the application logic to a separate process. So you ask "What is a surrogate process?"

## Surrogate Process

Early after the release of DCOM, which allowed you to remotely access local servers (known also as executables), Microsoft introduced a mechanism where-as a DLL can be loaded as a local server. This would in turn allow DLLs to be accessed remotely.

In order to use a DLL as a local server, you have to do two things. When you create the server objects with `CoCreateInstance()` or `CoGetClassObject()`, you have to specify the `CLSCTX_LOCAL_SERVER` class context. If you specified `CLSCTX_INPROC_SERVER`, `CLSCTX_SERVER` or `CLSCTX_ALL`, then COM will create the object in-process.

If you look at the `CClientDlg::OnInitDialog()` method in our MFC client, you'll note that I have commented-out a line of code that creates the application logic DLL as a local server (`CreateLocalObject()`). You can uncomment this line and comment-out the `CreateObject()` line to begin using your application logic in a surrogate process.

### Listing 5: OnInitDialog

```

BOOL CClientDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE); // Set big icon
    SetIcon(m_hIcon, FALSE); // Set small icon

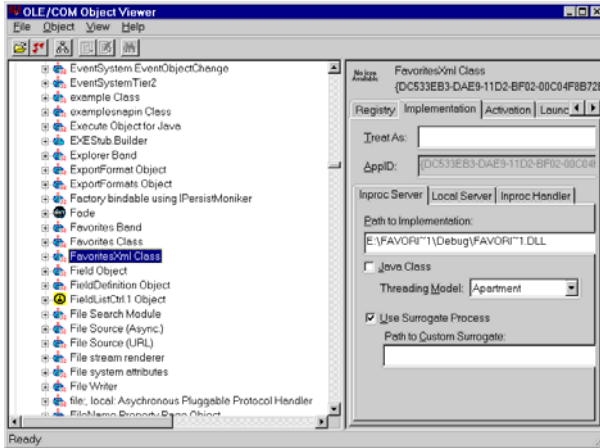
    // TODO: Add extra initialization here
    m_listcontrol.InsertColumn(0, "Name", LVCFMT_LEFT, 150);
    m_listcontrol.InsertColumn(1, "Url", LVCFMT_LEFT, 300);
    // object.CreateLocalObject(L"Favorite3Tier.FavoritesXml");
    object.CreateObject(L"Favorite3Tier.FavoritesXml");
    SendQuery();

    return TRUE; // return TRUE unless you set the focus to a control
}

```

You also have to mark the COM server to use a surrogate process. You can do this in OLEVIEW. Find the COM object in OLEVIEW and check the Use Surrogate Process checkbox in the Implementation tab (see figure 3).

Figure 3: OLEView Use Surrogate Process



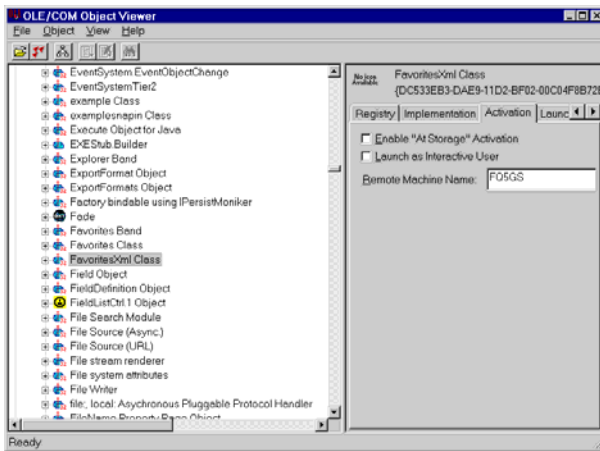
Once you've changed the one line of code and checked the Use Surrogate Process checkbox, your application logic will begin running in a separate process.

### Remoting the Application Logic

The next step is to move the application logic to separate machine, not only a separate process. This is quite easy. If you haven't already registered the application logic DLL on the server machine, then do it now (regsvr32 favorite3tier.dll). We have to mark the object to run in a surrogate on the server, so load up OLEVIEW on the server and navigate to the FavoritesXml class and check the Use Surrogate Process checkbox (see figure 3).

Now we configure the client-side by running OLEVIEW on the client and unchecking the Use Surrogate Process checkbox and specifying the Remote Machine Name in the Activation tab (see figure 4).

Figure 4: OLEView Remote Machine Name



Run the client again and wowie, it's running in a surrogate process on the remote machine. Or is it? Make sure that the dllhost.exe process is running on the server machine and not the client.

Another problem you are very likely to run into is configuring the DCOM security properly to get the example running across computer boundaries. You'll know if you have run into this problem by stepping through the code in Automation.cpp. If you are getting E\_ACCESSDENIED 80070005 errors from CoGetObject() then your security is not configured properly. This article is not about security, so I won't try to explain how DCOM uses NT security. But if you are struggling with the aspect, then I suggest you simply turn off DCOM security. To disable DCOM security, simply go to the Default Properties tab in DCOMCNFG and set the Default Authentication Level to None (see figure 5).

**Figure 5: DCOMCNFG Default Authentication Level**



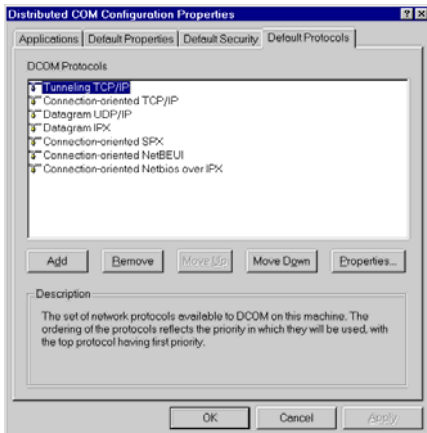
## COM Internet Services

A second problem you might encounter at this stage is the presence of a firewall between your client and server. If this was true, then I suggest you go back and repeat the last steps with a configuration that doesn't have a firewall between the client and server. I'd do this to make certain that your configuration is correct without the presence of the firewall. But don't worry in this section I will puncture a hole right through those firewalls.

In the next few paragraphs we will attempt to tunnel DCOM over TCP/IP port 80, using COM Internet Services. The steps I will enumerate here can also be found in the readme.txt file for both SP4 and SP5. I suggest you print section 3.12 of these files and verify that both you and I are correctly implementing these procedures.

Setting up the client side is quite easy. Well that is, if you have Windows NT 4 service pack 4. If you don't already have SP4 or later installed, then I suggest you take the time to do it now, as the next steps require SP4 on both the client and server machines. Once you have SP4 installed, simply start DCOMCNFG, go to the Default Protocols tab and add Tunneling TCP/IP as a DCOM Protocol. Now move Tunneling TCP/IP up to the top of the protocol list (see figure 6). You'll also have to reboot for these protocols to take affect.

**Figure 6: DCOMCNFG Tunneling TCP/IP**



Setting up the server side is a little more complicated. You have to add the Tunneling TCP/IP protocol as we did in the previous paragraph, but don't reboot right-away. We'll have plenty of occasions to reboot the server while we complete this multiple step program.

Also in DCOMCFNG, under the Default Properties tab, you have to Enable COM Internet Services on this computer by checking the checkbox.

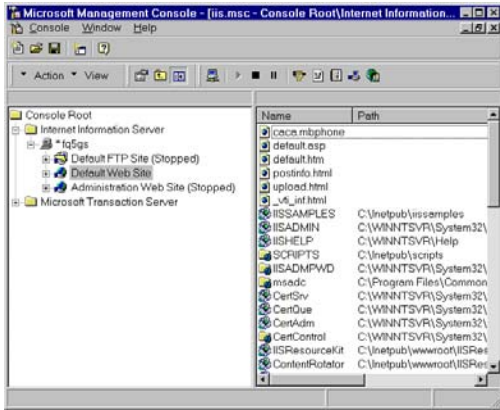
**Figure 7: DCOMCNFG - Enable COM Internet Services**



You should also have an Inetpub directory on this server. Under this directory, add a subdirectory called rpc. Copy Rpcproxy.dll from you windows system directory to this newly created directory.

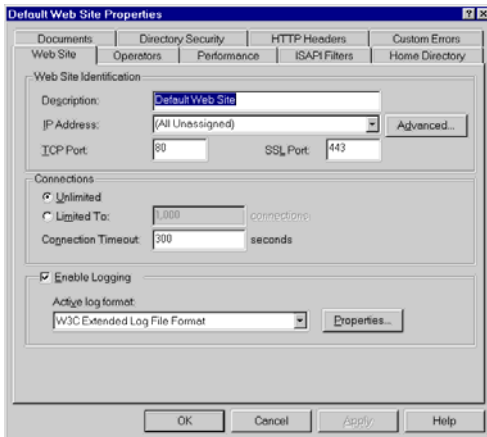
You also want to create a virtual root for the newly created directory. You can do this with the Internet Server Manager. Internet Server Manager is Microsoft's new Microsoft Management Console snap-in for IIS. From the Internet Server Manager left-pane, select Console Root | IIS | your server name | Default Web Site (see figure 8). Right-click on Default Web Site and select New | Virtual Directory in the popup-menu. The alias of your virtual directory will be rpc, the physical path will be that of your newly created directory and the permissions should include Execute Access.

Figure 8: MMC IIS



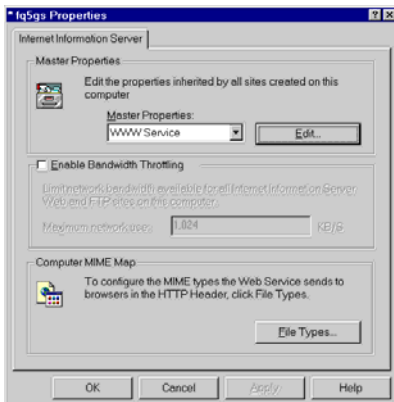
The second last step is to change the connection timeout to five minutes. I don't know why, but that's for the readme.txt file says. Again in the left-pane of the Internet Server Manager, right-click on Default Web Site then select Properties from the popup-menu. From the Web Site tab change the Connection Timeout to 300 seconds (see figure 9).

Figure 9: MMC Web Site Connection Timeout



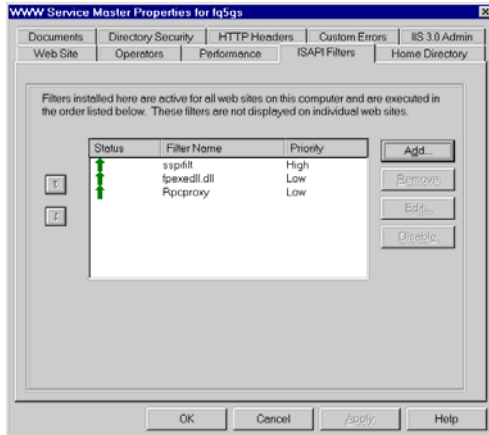
Now we're down to one more step. In the left-pane of the Internet Server Manager, right-click on the machine name then select Properties from the popup-menu (see figure 10). Select WWW Service in the Master Properties combo box and click on the Edit button.

Figure 10: MMC Master Properties



From the ISAPI filter tab, click Add to add a filter with the filter name Rpcproxy and executable rpcproxy.dll (see figure 11). Specify the full path to the rpcproxy.dll module in your newly created directory. You have to reboot the server for some of these changes to take affect.

Figure 11: MMC ISAPI Filters



Now if you run the client again you should notice that the dllhost.exe surrogate process starts on the server machine. You should also verify that we are using the correct port. You can do this by typing NETSTAT /A at the Command Prompt. You scan this list of Active Connections to determine which are being used by DCOM and what ports are being used by those connections. Now I dare you to throw that firewall between this DCOM connection.

Notes: You may be wondering why I created the application logic as a DLL and not an NT Service. You would think that it would scale better as an NT Service. This is true, but I've also anticipated the next step, which is to load our DLL into MTS.

## About the Author

Randy Charles Morin is co-author of [COM/DCOM Primer Plus](#) and [COM/DCOM Unleashed](#).