MANAGING MICROSOFT INTERNET INFORMATION SERVER DURING ISAPI DEVELOPMENT PROJECTS (TAMING THE BEAST FROM REDMOND)

By Dave Ball MCSE/MCP+I/MCT - delphi@eurocomp.co.uk

This article is produced from a Windows NT4 & Windows 2000 (W2K) Microsoft Certified System Engineer (MCSE) perspective and is designed to complement other published articles on Delphi ISAPI Web development. The bulk of the technical detail for this article has been extracted from Microsoft TechNet white papers and resource kits, combined with a fair amount of in-house testing! Due to the potentially confusing way in which Microsoft name their technologies, the Microsoft terms within this document which are not consistent with standard Delphi definitions will be shown in italics.

In the Beginning

So why is a MCSE developer using Delphi? Well, my company often has to integrate data held on diverse legacy systems into MS Back Office solutions. Usually the most effective way to do this is by means of custom ISAPI dll, feeding the data straight into an extended Intranet Knowledge Management (KM) Solution. In my view Visual Basic is not an ideal contender to build such ISAPI dlls, due to the memory footprint of the executables produced by the compiler and poor non-visual exception handling capabilities. When you then compare the ease of development with the Delphi VCL over MS C++ & MFC, Delphi becomes the only logical choice for this type of rapid, custom application development. However, many Delphi developers may be reluctant to exploit these types of solutions because the ISAPI environment of Internet Information Server (IIS) is often seen as unstable, frequently needing system re-boots to load and unload the ISAPI dlls. This is certainly true with an 'out of the box' installation; but, after you have finished reading this article I hope that your fears over IIS instability will have diminished to acceptable levels and that you will be more confident when developing to IIS.

Internet Information Server - a short history

IIS came about as the Microsoft opposition to the Netscape http server products, in very much the same way that Internet Explorer was developed to compete with the Netscape browser products. As is often the case with Microsoft products they are not as '*feature rich*' as would be optimal on initial launch; indeed IIS at version 3 was still a poor competitor to other Windows based http server products. Once web site developers started to need faster and more dynamic sites the IIS3 non-standard Win-CGI scripting gateway in IIS3 held them back. Microsoft responded with the ISAPI concept - a type of dll that would remain in memory and not have the reload overheads of the competing scripting environments. Realistically this was not implemented successfully until the arrival of IIS4 in the NT4 Option Pack. However, it is important to note that Microsoft see IIS as a core component of the Windows Servers and therefore IIS patches are contained within the normal Service Pack (SP) release cycle. As a result you should always have the latest SP installed unless there is a major clash with another mission critical application or service. Furthermore, IIS security holes are well documented on the web

and not having the latest SP (and hot fixes) installed is a sure way to be hacked. The majority of IIS security breaches are caused by lack of SP updates and/or poor configuration of supplementary firewall services, i.e. poor administrative practices by the system operators.

The ISAPI Development Platform

In order to prepare a Windows Server for ISAPI development specific software loads are required, especially for NT4 installations.

So what has to be installed on NT4 Server to support the ISAPI developer? Firstly install NT4 Server, the baseline CDROMs are normally at SP1 or SP3 standard depending on the purchase date (if you do not have NT4 Server on site then the easiest way for a developer to get a copy is to purchase a MSDN subscription, this provides development versions of all MS products and the MSDN Technical Library). Next load the NT 4 Option Pack prior to applying any SP; you will be prompted to load the pre-requisite components. If you do not have the NT4 Option Pack or latest SP you can download them from www.microsoft.com. When loading the Option Pack select both IIS 4 and the *Microsoft Transaction Services* (MTS) options. Once the Option Pack set-up is complete you must then install the latest NT4 SP to update all the OS, IIS and MTS components.

For IIS5 on W2K select the IIS components during installation or by adding Windows Components through Control Panel (add/remove software). The W2K equivalents of MTS, the COM+ services, are automatically installed by default.

Finally, if you want to dual-boot your NT4/W2K development server you **must** install NT4 and apply at least SP4 **before** you install W2K. W2K updates the NTFS file system to NTFS version 5, which cannot be accessed by NT4 prior to SP4, causing non-SP4+ builds to blue screen hang on boot!

MTS/COM+ - the Developers Secret Assistant

For this section I will concentrate on NT4 issues, as for W2K the concepts are very similar and fully integrated into the core OS as the COM+ environment. So why would a *transaction service* assist an ISAPI developer? Well if that were all that MTS did it would be of little interest to this discussion. However, MTS can really be thought of as a complete automatic control environment for COM applications on NT4.

Since the IIS4 service, the ASP.dll (active server pages engine) and all ISAPI dlls are also COM objects they can all be controlled by the MTS environment. The three key MTS functions of interest to the ISAPI developer are process isolation, automatic thread pooling and automatic object instance management.

Process Isolation:
MTS can be configured to run objects in their own process to ensure that failures do not spread to other components of the OS.

Automatic Thread Pooling:
As requests come from clients, MTS automatically assigns threads to components from a pre-alloated pool. When a component finishes executing, MTS reclaims the thread. This reduces the overhead of thread creation/deletion for better performance.

Automatic Object Instance Management:
MTS only creates instances of components on the server when requested and reclaims memory resources when the component finishes executing. This form of just-in-time (JIT) activation and as-soon-as-possible deactivation minimizes memory requirements on server machines.

So if the MTS environment can control IIS4 COM objects how can we use that during ISAPI dll development? To answer this question a short explanation of IIS *directory* and *web application* concepts is required.

Microsoft Web Applications

Surely Microsoft hasn't redefined *web application* - well actually yes. Within the context of IIS a *web application* is a collection of html & asp pages, together with any additional ISAPI dlls, located within the same IIS directory. Additionally, there are 3 different types of IIS *directories*: the *home directory* *virtual directories* and *sub-directories*.

The *home directory* is the default file system location to which a url pointing to http://www.devserver.com is mapped. Let's assume that the development IIS4 server http document root is at D:\inetpub\wwwroot- the files contained in that physical file system directory on the server are thus the contents of the *home directory* and are also the default *web application*. The default *web application* uses the same process space as the IIS server and therefore any crash caused by an ISAPI dll stored within this folder will bring down the core IIS service. So the first rule for a stable IIS system is that no ISAPI dlls should be executed from within the *home directory*.

Designed to ease security configuration a *sub-directory* is a simply a physical file system directory below the document root. For example http://www.devserver.com/shop would map to D:\inetpub\wwwroot\shop ; this url sub-directory structure allows access controls to be applied more efficiently but has no additional MTS specific functionality.

The IIS structure that is of the greatest use to the ISAPI developer is the *virtual directory*. This allows an IIS url to be mapped to any file system folder. For example the url http://www.devserver.com/ukbug/iserver.dll could map to the local file at F:\isapi test\iserver.dll. Additionally, and critically, *virtual directories* can also be configured to run in separate memory spaces from the host IIS instance. So our second rule is always put your ISAPI dlls into *virtual directories*, creating a new *virtual directory* for each ISAPI dll installed in the web server.

The MTS/COM+ Web Application Manager

The final key element in the MTS/COM+ control of IIS is the Web Application Manager

(WAM). This is the custom object wrapper that encapsulates any process separated IIS *component*. It is important to note that using the WAM MTS control object in NT4 does not mean that your ISAPI dll needs to be written as a MTS Object; it inherits full MTS instance management support from the *web application* object that calls it. IIS WAMs are created and destroyed automatically to manage requests from browser clients.

The WAM MTS/COM+ Process Protection System

Now that we know that an ISAPI dll can be considered a *web application* within a *virtual directory*, it is a logical step that the WAM can treat that*web application* as a separate process when the core IIS server instance calls it. The WAM will simply instantiate the required COM(+) objects outside the core IIS server process space. Additionally, if the *web application* crashes the WAM can shut down the separate process space and when the *web application* is called again a new instance is automatically created. Finally, because WAM owns the *web application* object instance an administrator can use the Microsoft Management Console (MMC) to shut down that instance on demand; without having to close down the core IIS service. For the ISAPI developer this means that you do not have to repeatedly register the ISAPI dll on the IIS Server system. Simply compile it on your Delphi development workstation, shutdown the controlling WAM *virtual directory* object on the IIS server, transfer the new dll into the mapped *virtual directory* file system location and let the WAM do all the instancing wok when the dll is next called.

Outline Development Environment

Now we have seen the theory let's walk through the configuration of a process protected NT4 & IIS4 environment. To begin with we need to define the server name and how we want to call our ISAPI dll from a client browser. In this example the server has the following configuration:

An additional IP address, which has been configured on the servers' network card TCP/IP settings and will be used solely for this site. The adding of a TCP/IP address will require a re-boot of the NT4 server.

A local DNS record or LMHOSTS entry on the workstation has been created for www.devserver.com, which points to the new IP address on the NT4 Server hosting the site.

A new *web site* on the IIS server to host the files for www.devserver.com .

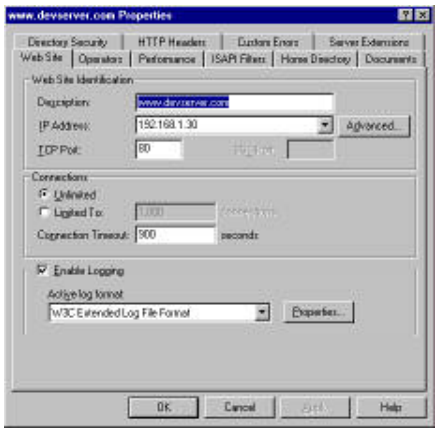
An IIS Virtual Directory called UKBUG, which is mapped to the file system directory f:\isapi test and contains a copy of the iserver.dll file compiled from the ISAPI sample project in the Delphi 5 demos\webserv directory

Therefore, a browser request to http://www.devserver.com/ukbug/iserver.dll will call the custom page defined within the Delphi demo iserver.dll .

IIS Server Configuration Step by Step

To set up the server firstly create a new IIS4 Web server instance for the www.devserver.com host. The administration tool used to complete his task is the MMC version 1.1for NT4 (Note: MMC v1.2 for W2K cannot fully manage NT4 servers).

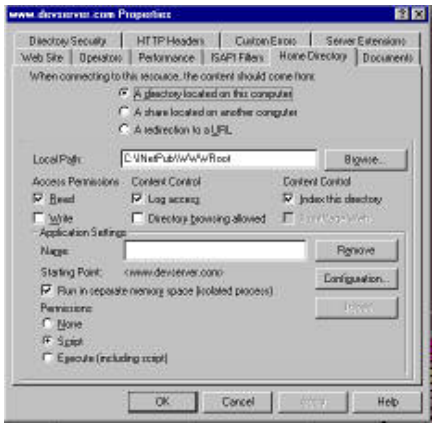
Select the server object in the MMC, right click and select New, Web Site, run the wizard using the following settings:



The IP Address is the new unique address (not the default all assigned).
The server port is 80 (default).
The Home Directory local path is where your files will be stored on the server.
Permissions are set to Read & Script (required ASP default).

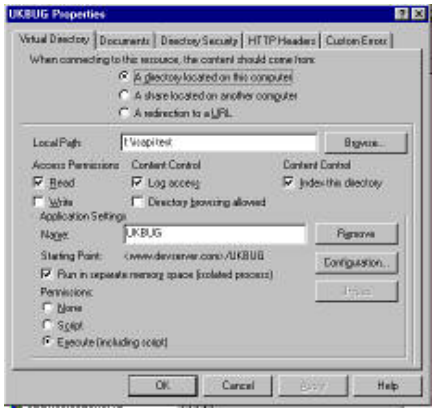
When the Wizard is finished select the new site, right click, select properties and check the settings are as shown opposite and as follows:

Logging is enabled, to use for testing purposes.
Run in Separate Memory Space (Isolated process) is set.



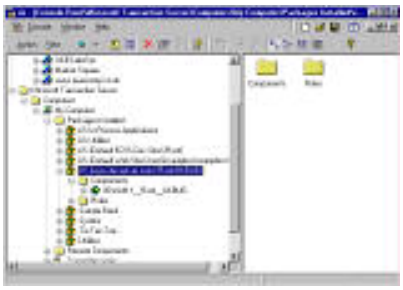
Using a unique IP address will stop interference from other IP stack processes, the script permission is adequate for this *home directory* location, as we now know that ISAPI dlls should not be executed within the *home directory*. For extra process protection the 'Run in Separate Memory Space' setting automatically creates a new MTS object for this website IIS4 process.

Now that a WAM object for the IIS4 server has been created within the MTS system we now need to configure a protected memory space for our ISAPI dll. The www.devserver.com IIS4 website is selected in the MMC window and a new virtual directory called UKBUG is created. For good security practice the local path is set to a directory outside the main IIS4 file system hierarchy. The compiled iserver.dll can be copied to that mapped directory when a new version is ready for testing. Once the New Virtual Directory Wizard has completed check that the properties for the directory are as shown . The key settings are as follows:



A directory located on this computer is selected.
Logging is enabled.
The local path is set to a physical drive on the server.
Run in Separate Memory Space (Isolated process) is set.

Permissions are set to Execute (including script), which is required for ISAPI dlls.
Note: Although you can map virtual directories across network shares MTS has a few issues managing objects across machine boundaries and this is not advised.



Now that the virtual directory has been configured a new WAM instance definition has been created. When the ISAPI dll is called from a browser using the URL http://www.devserver.com/ukbug/iserver.dll a new WAM instance is generated by MTS, as shown, encapsulating all the objects required to run iserver.dll in a protected memory space.

Creating a memory protected Virtual Directory in W2K & IIS5 is even easier; use the wizards to create the site and virtual directory and then set the appropriate properties as shown alongside. Note the change of terminology within *Execute Permissions* and *Application Protection*.

Re-deployment of the ISAPI.dll file(s)

Now that iserver.dll is in memory you could not normally copy a recompiled version over the original because a file system access violation would occur. But, because iserver.dll is in a virtual directory under MTS control you can simply right click on the UKBUG WAM instance in the NT4 MMC window and select shtu down, as shown in figure 7.

This automatic instancing is managed in the same way for W2K and IIS5; but is even easier to administer. Once a WAM is running a dll in memory an unload function becomes available within Internet Services Manager as shown below. To replace or update an ISAPI dll in a web application simply:

1. Start the Internet Service Manager MMC.
2. Right-click the UKBUG Virtual Directory in the object tree, then click Properties.
3. Select the Virtual Directory tab; click Unload.

The WAM will gracefully close down the in memory iserver.dll, allowing the recompiled iserver.dll to be copied into the file system directory. When the browser screen is refreshed the WAM will automatically launch a fresh instance of the new iserver.dll without the core web server instance

being compromised. Note that IIS may time out waiting for WAM to re-launch iserver.dll and return 'The remote procedure call failed and did not execute' - simply refresh the browser once more and the dll will load.

Finally, should iserver.dll cause a fatal exception, only the UKBUG instance will crash and then be automatically closed down by the WAM. The IIS instance will continue to run and no other web sites on the server will be compromised. Once the error is resolved and the dll has been replaced, IIS will restart the *web application* when it receives the first request for that url from a browser.

Deploying Your ISAPI dlls

Given the relative fragility of poorly configured IIS servers, and there are an awful lot of them out there, an ISAPI developer has two real options: either pester your ISP to set up a process protected virtual directory for your dlls to stop the production server from failing on errors or develop your dlls in within a local protected environment prior to deployment until you are sure they are robust.

In my experience the hassle and stress incurred while your ISP tries to set up a WAM protected ISAPI environment and then assign you the required administrator permissions is very likely to far outweigh the small investment required to install an NT4 / W2K box to use as a development tool as described in this article.

Also, now that you know how to set up these memory protected IIS *web applications* wouldn't it be great if your corporate IIS System Operators could set them up for you. Good idea; but, as these WAM concepts are not well covered in the on-line IIS administration documentation, they may not be aware of the ramifications of these configuration settings. This may be your chance to gets some beers out of the operations folks at last!

Finally, developers should not overlook custom ISAPI solutions to bring legacy data into an Intranet environment. Microsoft is spending large amounts of marketing funds pushing IIS and Exchange as the KM platform of choice for corporate Intranets. Utilising the open data connectivity of the Delphi environment to drag information into IIS and thus the corporate KM infrastructure is an opportunity waiting to be exploited by the Delphi developer community.

Summary

This article has sought to explain how to deploy ISAPI dlls within a MTS or COM+ memory protected development environment by simply configuring IIS settings within the MMC. The IIS WAM control instances are automatically created and destroyed on demand by browser calls and custom NT4 ISAPI dlls do not have to be written as MTS objects to be used in this way. The WAM also provides crash protection for ISAPI dlls under development and should a non-trapped exception occur the dll instance will be safely shutdown. The WAM environment provides a robust solution to building, debugging and deploying ISAPI dlls, which otherwise could fatally crash the hosting IIS web server.

There are two rules to remember:

1. Never execute ISAPI dlls within the *home directory*.
2. Place every ISAPI dll within a separate *virtual directory*.

Dave Ball is a Microsoft Certified System Engineer and Certified Trainer on both Windows NT4 and Windows 2000 and is also a member of the UK Borland User Group. He has been using Delphi, since version 1.0, to design and build custom solutions for customers using MS Back Office 4.5 / 2000 products. He can be contacted by email at delphi@eurocomp.co.uk .