

Dimeric Server Pages Deployment Guide

Content

1 Dimeric Server Pages Deployment Guide	1
1.1 Introduction	1
1.2 Configuring Your DSP Application(s)	1
1.3 Configuring the DSPProxy	2
1.4 Configuring IIS	5
1.5 Configuring Houston	6
1.6 Configuring Your DSP Application to Run as a Web Server Plug-in	7
2 Index	9

1 Dimeric Server Pages Deployment Guide

You can contact us and find out more about DSP and our other products at <http://www.Dimeric.com/>

Copyright (c) 2002 by Dimeric, L.L.C. All rights reserved.

1.1 Introduction

Welcome to the Dimeric Server Pages Deployment Guide. In this guide, we will describe how to deploy a DSP application into a production environment. The configuration will provide scalability, robustness, high up-time and ease of management, including simplified upgrading of your applications with no service interruption.

One of the features that we at Dimeric most like about DSP applications is their ease of development. A DSP application is a normal Delphi application. To create a DSP application, use the DSP Application Wizard: File | New | Other | DSP | DSP Application. Hit F9, and you are running the application! The application is a web server, so you just need to point your browser to <http://localhost:8080/> (or whichever port you choose), and you can exercise your application. One of the best parts of this scheme is that **you are running in the debugger**. That means that you can set breakpoints, evaluate expressions, set watches, etc. Debugging your DSP applications is simple, unlike other web application development tools, which can make you jump through hoops to debug your applications.

When it comes time to deploy your application into production, we have provided a full-featured solution that is scalable, robust, efficient, and easy to manage. It does require some configuration, however. This guide will describe how to configure a new production environment to run DSP applications, and will describe how to upgrade your DSP applications without service interruption (preserving the previous configuration for easy rollback).

The following are the components that make up a typical production installation. This guide describes in detail the purpose of these components and how to configure them.

DSP Applications:

- The same stand-alone EXE you build and debug in development, or a web plug-in, such as an ISAPI DLL.
- Can shutdown after a period of inactivity, if a stand-alone EXE.
- Can store session state in memory or in a database for load balancing in a web farm.

DSPProxy:

- A web server plug-in that forwards requests to your DSP applications. It can also run as a stand-alone application or Windows service.
- Can launch your DSP applications when they aren't running, or it can ask Houston to launch them.
- Can shutdown your DSP applications in order to upgrade them, or can ask Houston to shut them down.
- Maintains a pool of connections to your DSP applications.
- Not necessary if you build your application into a web server plug-in.

Commercial Web Server

- Hosts either the DSPProxy or your web server plug-in DSP application.
- Provides SSL support.
- Manages a thread pool and efficiently maintains many socket connections.
- Currently IIS is supported, with Apache support planned.

Houston

- Launches your DSP applications.
- Can shutdown your DSP applications.
- Upgrades your application(s) on launch, preserving the previous configuration for easy rollback.
- Typically runs as a Windows service, but can run as a stand-alone application.
- Can run as any Windows user.
- Can be installed on a different machine than the DSPProxy, making it possible for the proxy to launch applications that reside on a different machine.
- Not necessary if you build your application into a web server plug-in or if you have the DSPProxy launch your stand-alone DSP application.

1.2 Configuring Your DSP Application(s)

When you are ready to deploy your application into production, you should set aside a folder for your application (on the server where you want your application to run). This folder and its subfolders should contain only your application and related

files. Put your application's EXE or DLL and INI files into that folder. You should also put that application's *wwwroot* folder in that folder. The *wwwroot* folder contains files that are specific to your application, and that your application serves up (for example, images, static HTML files, style sheets, PDF documents, etc.).

If your DSP application is a stand-alone EXE, then you should also create a folder named *Versions* as a sub-folder of the folder that contains your application. Initially, this folder will be empty, but Houston will use it to find a new version of your application (and its supporting files), and to store previous versions of the same. Houston uses this folder when it upgrades your application from one version to another. See the Houston documentation below for details. Note that if you are deploying your application as a web server plug-in, then you won't have the automatic upgrade feature that Houston or the DSPProxy provide because they aren't launching your application; your web server will be loading your application.

The application's INI file contains several configuration options that you may adjust to suit your needs.

- **Port.** This option specifies the TCP port that the application should bind to (*i.e.*, listen on). You should set the application's port so that it doesn't conflict with any other services on the machine. Also, the port shouldn't be in the server's "ephemeral port range" (see http://www.ncftpd.com/ncftpd/doc/misc/ephemeral_ports.html for details). You will need to configure the DSPProxy, in a later step, so that it knows what port to use when forwarding requests to your application. A single web server and DSPProxy can forward requests to multiple DSP applications. This setting only applies to stand-alone DSP applications.
- **AutoShutdownTimeoutMinutes** This is the application's idle timeout. If the application has served no requests for this number of minutes, it will shut itself down. When the next request comes in, the DSPProxy will notify Houston to launch your application. This setting only applies to stand-alone DSP applications.
- **RootFolder** This is the folder the application will use when serving up files (*i.e.*, not DSP pages, but images, static HTML, etc.). Depending on how you configure the web server, this setting may not matter. You can configure your web server to serve up all static files, and only forward .dsp requests to the proxy (and, hence, your application). On the other hand, you might want to configure the web server to forward all requests to your application. For example, you might want to do this if you are running your application on a separate server. In that case, you must set the RootFolder property so the application knows where to find the files you want it to serve up. If you configure your application as suggested above, then this will be the "wwwroot" subfolder of the folder that contains your application. This setting only applies to stand-alone DSP applications because you should configure the web server to serve up the files.
- **LogFileName** This is the location of the application's log file. By default, it is the same base name as the application with the .log extension (in the same folder as the application).
- **MaxLogFileSize** This is the maximum size of the application's log file, in bytes. The default value for this option is 1 megabyte. You can use a suffix of 'm' or 'k', meaning megabytes or kilobytes. For example, "5m" means 5 megabytes.

The DSP Application INI file supports more configuration options, as documented in the INI file itself. The options listed above are those that are most likely to be modified when the application is deployed to production.

1.3 Configuring the DSPProxy

The DSPProxy is a plug-in to your web server that can forward requests from the web server to one or more DSP applications. Currently IIS is supported (DSPProxyIIS.dll), with Apache support planned. There are also versions of DSPProxy that can be run as a stand-alone application (DSPProxy.exe) or Windows service (DSPProxySvc.exe). This section documents the configuration of the DSPProxy itself. The documentation on how to plug the DSPProxy into the web server is in the section [Configuring IIS](#) (see page 5) below.

Note that while the stand-alone and Windows service versions of the DSPProxy are functionally equivalent to the web server plug-in version of DSPProxy, their use in a website with heavy traffic is discouraged. IIS, Apache and other dedicated web servers are more efficient and provide services that the stand-alone and Windows service version of the DSPProxy do not. Web servers offer features like thread pooling, SSL / HTTPS support, efficient management of many socket connections, security services, etc.

The full source to the DSPProxy (including the IIS plug-in, the stand-alone DSPProxy application, and the stand-alone DSPProxy service) is included with the full version of DSP.

You can create a new folder on the web server machine where you put the DSPProxyIIS.dll and its INI file, or you can put these files in the *InetPub* folder. Either choice will work.

The DSPProxy's INI file is named *DSPProxy.ini* and is in the same folder as the DSPProxy executable (EXE or DLL). The following are the configuration options in the *DSPProxy.ini* file.

In the **[General]** section:

- **AdminFolder** This is the folder that can be used to access the proxy's administration website. For example, if the host is `www.SomeHost.com`, with the `AdminFolder` set to `Proxy`, then the administrative site can be accessed with `http://www.SomeHost.com/Proxy/Admin.dsp`. If the proxy is hosted by a web server, for example the `DSProxyIIS.dll` plug-in in IIS, then the web server (IIS in this example) would need to be have a virtual folder configured which matches the `AdminFolder` name. See the DSP Deployment Guide for more information. The default for this option is the empty-string (*i.e.*, no administration folder).
- **AdminPassword** *This feature is currently not supported.* By default, the `AdminPassword` is not set. If it is set, then the `AdminFolder` is only accessible if the user supplies the password. If the proxy is being hosted by a commercial web server (such as IIS) the you should use the built-in security of the web server, if possible.
- **VerboseLogging** Acceptable values are Y and N, defaulting to N if not specified. If `VerboseLogging` is enabled (Y), then the proxy will log every request and response that passes through it. This is a lot of information, and is normally used only in development (or temporarily in a production setting). `VerboseLogging` can be enabled and disabled without shutting down the proxy through the proxy's administration website (see `AdminFolder` above).
- **LogFile** This is the name of the log file the proxy will log to. If `VerboseLogging` is enabled, then this is the log file that will contain the verbose log output. The default for this option is the empty string, which disables all logging, regardless of other settings.
- **MaxLogFileSize** The maximum size of the proxy's log file, in bytes. If the log file grows to this size, then it will be renamed to `<log file name>.old.<ext>`. For example, if the log file name were `DSProxy.log`, then the log file would be renamed `DSProxy.old.log`. If `DSProxy.old.log` already exists, it will be overwritten. The default value for this option is one megabyte. The value specified must be a positive integer. You may append the letter "m" or "k" to indicate that value is in megabytes or kilobytes. For example "100k" means 100 kilobytes and "1m" means one megabyte.
- **PurgeIntervalSeconds** The connection pools maintained between the proxy and each application are periodically checked to find stale connections (connections that have not been used in the timeout specified for the application using `KeepAliveSeconds` -- see below). This option specifies how often the connection pools are checked for stale connections. By default, this value is 15 seconds.
- **Port** In the stand-alone `DSProxy.exe` and the `DSProxy` service (`DSProxySvc.exe`), but not the web server plug-in, you may set the port that the proxy listens on. By default the proxy listens on port 80.

Sections for each proxied DSP application:

- **[App:name]** Each application section is named `[App:name]`. These sections define how the proxy forwards requests for these virtual folders. For example, if the section `[App:Marketing]` is defined, and the hostname is `www.SomeHost.com`, then the proxy would forward requests that start with `"http://www.SomeHost.com/Marketing/"` to the application specified in the `[App:Marketing]` section. If the proxy is a web server plug-in to a commercial web server, then you would configure your web server to have a virtual folder with this name so that requests are routed to the `DSProxy` when they are for this folder name.
- **Connections** This option must be specified. It is of the form "host:port", for example, "localhost:5000". This indicates the host and port that the proxy should forward requests to.
- **SSL** Determines whether the application requires an SSL connection. It does NOT determine whether incoming requests can use SSL. The stand-alone proxy does not support clients connecting with SSL. If you want clients to use SSL, then use the `DSProxy` web server plug-in (such as `DSProxyIIS.dll`) and use the SSL support of a commercial web server. In order for the proxy to act as an SSL client to the application, you must supply the OpenSSL DLLs, as you would for the Indy internet components. See the Indy documentation that comes with Delphi or the documentation on the Indy website (<http://www.nevrona.com/indy/>). Acceptable values are "N" and "Y", with a default of "N".
- **KeepAliveSeconds** The time that a connection between the proxy and the application will remain unused in a pool of connections before the proxy will drop the connection. The default value for this option is 60 seconds. This is not mainly a performance enhancement (even though it should enhance performance). It is mainly to avoid running out of "ephemeral ports". For more information on ephemeral ports, see: http://www.ncftpd.com/ncftpd/doc/misc/ephemeral_ports.html
- **PingURLPath** The path portion of a URL that the proxy can use to ping the application to determine if it is still running. Normally you don't have to set this option. This URL will be called frequently, so it should be efficiently processed by the application. By default it is set to `"/ping"` which is handled specially (and efficiently) by DSP applications.
- **Launched** Indicates whether this application needs to be launched, either by the proxy itself or by the Houston launcher. Valid values are Y and N, with a default value of Y. If the application is configured with an `ExePath`, and `Launched` is Y, then the proxy will launch the application when it is unable to connect to it. If `Launched` is Y, and an `ExePath` is not specified, then the proxy will contact the Houston launcher on the same host as specified in the `Connections` option, and then ask it to launch the application. In this way, your proxy and application can run on different machines. See the section below on the `DSProxy` launching applications for more details.

- **HoustonName** If the proxy will contact a Houston launcher to start the application then, by default, it will ask Houston to launch the application with the same name as the application (*i.e.*, the name in the [App:<name>] section heading. If Houston is configured with a different name for this application, then you can use the HoustonName option to inform the DSPProxy of this name. When the proxy contacts Houston, it will ask Houston to launch this application. Note that if the proxy is launching the application, then it will ignore this setting.
- **HoustonPort** By default, the Houston launcher will bind to port 5200. If you have configured the Houston launcher that will launch this application to bind to another port, then you can use this option to inform the DSPProxy of this port. When the proxy contacts Houston to launch this application, it will connect to this port. Note that if the proxy is launching the application, it will ignore this setting.
- **ExePath** The full path to the application executable. When this option is specified, the DSPProxy will launch the application if it cannot connect to the application. See the section below on the DSPProxy launching applications for more details.
- **StartupDelaySeconds** The number of seconds that the proxy will wait for the application to start, before giving up. The default is 30 seconds. The proxy attempts to send a ping request to the application to determine if the application has launched successfully. This setting is only used if ExePath is also specified. See the section below on the DSPProxy launching applications for more details.
- **ShutdownDelaySeconds** The number of seconds that the proxy will wait for the application to shutdown, before giving up. The default is 45 seconds. See the Houston documentation for details on this mechanism. This setting is only used if the ExePath is also specified. See the section below on the DSPProxy launching applications for more details.

Here is a sample DSPProxy.ini file:

```
[General]
AdminFolder=Proxy
VerboseLogging=N
LogFileName=DSPProxy.log
MaxLogFileSize=1m
PurgeIntervalSeconds=15

[App:Marketing]
Connections=localhost:5201
ExePath=C:\Production\Marketing\Marketing.exe

[App:Sales]
Connections=SalesServer:5202

[App:Operations]
Connections=OperationsServer:5203
Launched=N

[Alias:Main]
App=Marketing
```

Given the above INI file, suppose that the web server is configured accordingly and that the hostname is www.SampleHost.com. Then the following URLs are routed as described:

http://www.SampleHost.com/Marketing/Default.dsp	Forwarded to the application listening on LocalHost:5201
http://www.SampleHost.com/Main/Default.dsp	Same as above (Main is an alias to Marketing)
http://www.SampleHost.com/Sales/Default.dsp	Forwarded to the application listening on SalesServer:5202
http://www.SampleHost.com/Proxy/Admin.dsp	Processed by the DSPProxy for administrative access

While the proxy is running, you can access the proxy's status and perform administrative functions using the proxy's *Admin.dsp* page. From this page, you can enable and disable logging (for debugging purposes). Additionally, you can shutdown applications (*i.e.*, you can have to proxy request Houston to shutdown an application). You can also see what applications are configured, whether the proxy is keeping a connection pool to that application, and how many connections are in the pool.

The DSPProxy Launching Applications

When the proxy launches an application for you, it works just like Houston. For example, it will upgrade your application to a new version if there is a new version to deploy, saving a copy of the existing version and all of its supporting files.

Note that if the proxy is a web server plug-in, and it launches the application, then the application will run with the same credentials (username) as the web server. IIS will only run as the SYSTEM user, which has limited access to resources, such as shared folders. If you want your application to run with different credentials, then you can use Houston to launch the application (even when running on the same machine as the web server). You can install the Houston service (HoustonSvc.exe), and then set its credentials through the Services control panel. In the Services control panel, select the Houston Launcher service, right-click and select Properties. In the Log On tab of the properties dialog, select the radio button for "This Account", and fill in the username and password of the desired user account.

1.4 Configuring IIS

You will want to use a commercial web server, such as IIS or Apache, to host your application, because of the services that it provides. The advantages of using a commercial web server include features such as SSL / HTTPS support, thread pooling, efficient connection management, security services, etc. This section will describe how to configure IIS to install the DSPProxy ISAPI plug-in or your DSP Application, which you have built into an ISAPI plug-in, and setup virtual folders that map to your DSP applications. These instructions apply to IIS version 5.0.

1. Open the Internet Service Manager.
2. Right-click on the website you are configuring (e.g., Default Website), and select New | Virtual Directory.
3. Click Next.
4. Type in the Alias name that matches either the administrative folder for the proxy (e.g., "proxy"), or that matches an application name that you configured in the DSPProxy.ini file (e.g., "Marketing" or "Sales"), or pick the folder name that you want to use to refer to your DSP Application ISAPI plug-in.
5. Click Next.
6. Click Browse and select the application's wwwroot folder that you configured when you created a folder for your application. See Configuring Your DSP Application(s) (see page 1) for details. If you are configuring a virtual directory for proxy administration or for an application that resides on another machine, then you should select an empty folder. You may want to create an empty folder on the web server machine just for this purpose.
7. Click Next.
8. In the Access Permissions page of the wizard, the Run and Run Scripts boxes should be checked. You should also check the Execute box.
9. Click Next.
10. Click Finish.
11. Notice that the virtual directory has been added to the tree-view under the website. Right-click this folder in the tree-view and select Properties.
12. Change Application Protection from "Medium (Pooled)" to "Low (IIS Process)". We have noticed that the Medium and High settings cause problems when restarting IIS. Because the proxy does so little (it just forwards requests to your DSP application(s) and can report its status, etc.), it is not a problem to run it in the IIS process.
13. Click on the Configuration button.
14. In the App Mappings tab, click the Add button.
15. Click the Browse button next to the Executable edit box. Change the File type to *.dll and find the DSPProxyIIS.dll or your application's DLL.
16. In the Extension edit type either ".dsp" or "*" and click Ok. If you type "*", then all requests for this virtual directory will be routed to the DSPProxyIIS.dll (and then forwarded to your application). If you type ".dsp", then only .dsp requests will be routed to the DSPProxy or your application's DLL. All other requests will be handled by IIS directly. In this case, you should have mapped the virtual directory folder to the folder containing the files that go with your application (e.g., images, static HTML, style sheets, etc.). If you are running your application on the same server as IIS then you should choose ".dsp" instead of "*", because IIS is more efficient at serving files directly when they are on the same machine.
17. Click Ok on the Application Configuration dialog.
18. While not part of the DSPProxy configuration, at this point you may want to set the Directory Security options to disallow anonymous access if you are configuring the proxy admin virtual folder. This way, external users will not be able to access the proxy admin pages. To do this, click on the Directory Security tab, and then click the Edit button in the box "Anonymous access and authentication control." Uncheck the "Anonymous access" checkbox and click Ok. If you only have "Integrated Windows authentication" checkbox checked, then a user will have to supply a valid windows login in order to access this virtual folder. In addition, the password is not sent across the internet in plain text (nor is an MD5 hash of the password sent across the internet). See MSDN for details.
19. Click Ok on the virtual directory properties dialog.
20. Repeat these steps for each DSP application and for the proxy's AdminFolder, if you are using the DSPProxy.

1.5 Configuring Houston

The Houston application is responsible for launching and terminating stand-alone DSP applications. It can also upgrade your stand-alone DSP application and its associated files to a new version at launch time, preserving the previous version for rollback, if necessary. Houston is typically installed as a Windows service in a production environment (HoustonSvc.exe), although it can run as a normal application as well (Houston.exe).

If your DSP applications are built as web server plug-ins, then you don't need either Houston or the DSPProxy. If you are running your DSP applications on the same server as the proxy, then you don't need Houston; you can simply have the proxy launch your DSP applications instead of separately installing Houston. The proxy will do exactly what Houston does to launch and upgrade an application. You may still want to use Houston, even in this case, because the Houston launcher service can run with any credentials (NT account). IIS, for example, can only run as the SYSTEM username, which has limited access to some resources, such as shared folders on other machines. See the section on Configuring the DSPProxy (see page 2) for details.

The version of Houston that is a Windows service is called HoustonSvc.exe. To install the Houston service, run the HoustonSvc.exe with the command-line: "HoustonSvc /install". To uninstall the HoustonSvc.exe, run the command-line: "HoustonSvc /uninstall". After installation, you can go to the Services control panel to set the username and password that the service should use to log in. This is the user that the applications launched by Houston will run as. Right-click on the Houston Launcher and select Properties. From the Log On tab of the properties dialog, you can select the radio button for "This Account", and supply a username and password the Houston service should use. From the services control panel, you can also, of course, start, stop and set the startup policy of the Houston launcher.

Houston has a configuration file named Houston.ini, which has the configuration information for every application that Houston can launch. Here are the configuration options you can specify.

In the **[General]** section:

- **Port** The port that Houston will bind to and listen on. By default, this is 5200. You may set this option to change the port. In that case, Houston clients (such as the DSPProxy) must also be configured with this Houston port.

In the **[App:<name>]** section:

The Houston.ini file can contain any number of sections that start with "App:", such as [App:Marketing] and [App:Sales]. These sections define how Houston launches the associated application.

- **ExePath** The full path to the application executable. This is the application that Houston will launch on behalf of a Houston client.
- **PingURL** This option must be specified for DSP applications. It is of the form "http://host:port/path". A typical example would be "PingURL=http://localhost:5201/ping" for a DSP application listening on port 5201. Because both the DSPProxy and Houston must be able to ping DSP applications, every DSP application understands the "/ping" request and will automatically (and efficiently) respond to it.
- **StartupDelaySeconds** The number of seconds that Houston will wait for the application to start before giving up. The default is 30 seconds. Houston attempts to send a ping request to the application to determine if the application has launched successfully.
- **ShutdownDelaySeconds** The number of seconds that Houston will wait for the application to shutdown, before giving up. The default is 45 seconds. See below for details.

For example, if you had applications named Sales.exe and Marketing.exe, you might have the following in your Houston.ini file:

```
[App:Sales]
ExePath=C:ProductionSalesSales.exe
PingURL=http://localhost:5201/ping
```

```
[App:Marketing]
ExePath=C:ProductionMarketingMarketing.exe
PingURL=http://localhost:5202/ping
```

Startup

When Houston launches your application, it first runs the executable. Then, it repeatedly tries to ping the application using the PingURL provided. Once it is able to ping the application, it responds to the Houston client (*i.e.*, the DSPProxy) with a success status. If it encounters any errors while launching the application, or if it is unable to ping the application for the time specified by StartupDelaySeconds, then it will respond to the Houston client with a failure status.

Shutdown

When a DSP application launches, it creates a file with the same name as the application, but with the extension .kill (e.g., Sales.kill), which it keeps open. It periodically checks the file to see if the word "kill" has been written to it. When it sees this command, the application shuts itself down. All of this functionality is in the *DSHoustonKillTask* utility unit.

When Houston is asked to shutdown an application, it writes the word "kill" in the application's .kill file. This triggers the application to shutdown. Houston then repeatedly attempts to delete the .kill file, until it successfully deletes it or it reaches the ShutdownDelaySeconds timeout. Houston will not be able to delete the file until the application terminates, because the application has the file open. If Houston reaches the timeout, then it will report an error. If it is able to delete the file, then it will report success.

Automatic Upgrade and Archive

When Houston launches an application, it will first look for a folder in the same folder as the EXE called "Versions". Inside that folder it will look for a folder named "new". If it finds the "new" folder, Houston will upgrade and archive the application. It will first create a folder inside "Versions" named "oldn", where *n* is a number that will make the "old" folder unique inside "Versions". Houston will then recursively copy files from the application's folder into the "old" folder. This is the archive part of the upgrade process.

Once the files in the application's folder, and sub-folders, have been archived, Houston will then recursively copy the files from the "new" folder into the application's folder (and corresponding sub-folders, etc.). Houston will overwrite files when there are duplicates. Because the application isn't running during this process, the files should not be in use and Houston should be able to overwrite them with the versions in the "new" folder. Once the files have been deployed, Houston will delete the "new" folder and then will launch the application, as it was asked to. Only after this entire process is complete will Houston return success or failure to its client (i.e., the DSPProxy).

This is how Houston archives and upgrades your application for you. Because the DSPProxy is waiting for a response for Houston before forwarding requests to this DSP application, requests for this application will be blocked in the DSPProxy while Houston is upgrading your application. Users will see a pause in service, but not a service interruption (i.e., their requests will just take a little longer to process during the upgrade; they will not receive an error).

Note that if you don't want a specific folder or folders archived, then you can create a file, called SkipFolders.txt, in the application's folder. In that file you should list each folder (relative folder name), one per line, that you want skipped during the archive process. To conserve disk space, and to improve response time, you should skip any non-essential folders. For example, if your application has large amounts of static files that are not worth archiving, you should skip them.

Here is a step-by-step procedure for upgrading a DSP application:

1. Create a "new" folder in the "Versions" folder of your application's folder. The "new" folder contains all of the files you want copied into your application's folder during the upgrade process. The folder hierarchy should match the application's folder hierarchy.
2. Point your browser to the DSPProxy's admin page (e.g., <http://www.SomeHost/proxy/admin.dsp>). Click on the Shutdown button next to the application you are upgrading. Wait for the shutdown to complete.
3. Point your browser to any page in your application (e.g., <http://www.SomeHost.com/Sales/Default.dsp>).

In step 2, the proxy will contact Houston and ask it to shutdown the application, as described above. It will block all incoming requests until Houston returns either success or failure.

In step 3, the proxy will attempt to contact the application, which will fail because you shutdown the application in step 2. The proxy will then contact Houston to launch the application. Before launching the application, Houston will see the "new" folder in your application's "Version" folder, and will upgrade and archive the application as described above. Houston will then launch and ping your application, and then return success to the proxy (once it can successfully ping the application). The proxy will retry forwarding the request to the application, which will now succeed. During this entire process, users may have seen a delay, but not an error.

1.6 Configuring Your DSP Application to Run as a Web Server Plug-in

You can build a DSP application as a web server plug-in. When you select File | New | Other | DSP | DSP Application, the New DSP Application dialog will ask whether you want a stand-alone DSP application, or a web server plug-in, such as ISAPI.

If you have an existing stand-alone application that you want to build as a web server plug-in, you can simply save the project under a new name (i.e. File | Save Project As). If you are building an ISAPI module, then you would change the word "program" to "library" and then add the unit "DSIsapiPlugin" to the project's uses clause. Note that the application's INI file must have the same base name as the application executable (i.e. if you change the name of the executable through File | Save Project As, you should also change the name of your INI file to match).

Once you have built your application as a web server plug-in, you can install it into your web server as described in the section of this guide for your web server.

Our first version of DSP (version 1.0) only allowed building a single ISAPI module. While this was simpler to configure initially, we discovered several important drawbacks.

- **Debugging** – debugging an ISAPI application is a horrible experience, so having your DSP application be a stand-alone EXE means it is much easier to debug.
- **Development** – similar to the above point, with a stand-alone EXE, it is possible (in fact, preferable in most cases) to develop and test your DSP application without even having IIS running on your development machine. Just hit F9 inside Delphi, then point your browser to the appropriate URL (or leave the browser open and hit Refresh). Of course, in production, you'll want to use IIS, but that just involves copying the DSP application's EXE file.
- **Deployment** – deploying a new application required shutting down IIS (and sometimes, due to IIS instability, rebooting the computer). With a separate EXE, we can deploy new versions without service interruption. In addition, DSP now offers a plug-in architecture for session management, and the *VDBWebSession* manager provides session management via the RDBMS. This is nice for two reasons: it supports web farms (multiple web/app servers that handle requests in a load-balanced manner); and it is persistent, so that when the application shuts down (as when deploying a new version), no sessions are lost (users don't have to log in again, etc).
- **Configuration** – because our application ran inside IIS, it ran as the SYSTEM user (this cannot be changed), and this did not work well with various RDBMS client libraries we needed to use in the web application. With a separate EXE, you can have your DSP application run as whatever user you wish. An extra benefit of this is that you can, therefore, use Task Manager to kill your DSP application (if need be), which would not be possible if the process is owned by the SYSTEM account.
- **Security** – with a separate EXE, you can run IIS in the DMZ (demilitarized zone) portion of your network, and run your DSP application on your private intranet. This improves security, while at the same time allowing your DSP application to access resources (e.g., RDBMS) on the private network. See <http://img.cmpnet.com/nc/1021/graphics/ws12.pdf> for a nice diagram.
- **Robustness** – some software we use (such as the BDE) has resource leaks, and because IIS is running constantly, we would eventually run into trouble and have to reboot the web server. Now that our DSP applications can be separate EXEs, we have them configured to automatically shut down after a specific period of inactivity (no incoming requests). This way, they automatically restart themselves overnight when few requests are coming in.
- **Scalability** – we wanted a single IIS instance to be able to farm the real work out to a separate application server, potentially running on a different machine (for example, the same machine as the RDBMS, or at least "closer" on the network).
- **Reliability** – any problems in the application affected IIS as a whole, because they were both running in the same process.

While you can build your application into a web-server plug-in, we feel that the combination of the DSPProxy, Houston and the stand-alone DSP application is significantly better for the reasons cited above. The choice is yours.

If you do choose to deploy your DSP application as a web server plug-in, here is a tip on how you can upgrade the application with the minimum of down time.

1. Rename the currently loaded executable. For example, rename Sales.dll to Sales_old.dll. Even though the DLL is loaded by the web server, you can rename it.
2. Copy the new executable (e.g. the new Sales.dll) into the application's folder (i.e. the folder where you just renamed Sales.dll to Sales_old.dll).
3. Restart the web server. For IIS you would right-click on the website and select Restart IIS from the popup menu.

This is a bit heavy handed because the entire website is down for a short period of time. For IIS, instead of step 3, you could try to simply unload the virtual directory (right-click the virtual directory, select Properties and then on the Home Directory tab click the Unload button). We have noticed that IIS does not reliably unload ISAPI DLLs. Your mileage may vary.

Note that this procedure will re-initialize your DSP application. All web sessions stored in memory will be lost. If you want to preserve session state across an upgrade, you can use the VDB session manager, which stores session state in a database. See the comments in *VDBWebSession.pas* for details.

Index

C

Configuring Houston 6

Configuring IIS 5

Configuring the DSPProxy 2

Configuring Your DSP Application to Run as a Web Server
Plug-in 7

Configuring Your DSP Application(s) 1

D

Dimeric Server Pages Deployment Guide 1

I

Introduction 1

