So You Want To Be A Web Wizard...

Tools for building web apps in Delphi

Reviewed by Brandon Smith

Developing a web application is a simple three-stage process. First pick one or more of the following acronyms: HTTP, HTML, PING, POP3, SMTP, NNTP, TCP/IP, UDP, ISAPI, CGI, NSAPI, XML, TELNET, FTP, ASP, DCOM, CORBA, etc. Next select your language of choice, such as Visual Basic, Java, Delphi, JavaScript, VBScript, Perl, NetRexx, C++, or whatever. Finally, pull up your favorite internet search tool, such as WebFerret, type in your selections from stages one and two, and press `Enter`. You will now have a list of several hundred sites, most of which use one or more of your selections, and some of which actually provide what we're really looking for: a component suite that you can mate with your language of choice to produce a web application.

This article is not a conventional review, rather it's a discussion of some of my experiences with specific projects using a selection of tools. But, nevertheless, it should prove useful in your own decision process.

What's In The Box...

When I first got the assignment to convert an existing client/server Windows 3.1 application to a web browser client going against an NT/IIS ISAPI server application in Delphi 3, I did not do any searching. Delphi Client/Server 3 (and 4) comes with an excellent set of web application development components based on a special form, `TWebModule`. The Netmaster suite of internet components which comes with Delphi 2, 3 and 4 is *not* excellent in my view, although the original set, WinShoes, is. `TWebModule` is an entirely different proposition: an HTTP-aware form which hosts components such as `TPageProducer` to make producing dynamic HTML almost trivial. Within a week, I had the logon sequence completed and was translating the client/server logic behind the initial search and selection screens to the new HTML 3.2 client environment.

During my initial burst of work, I'd also learned the basic secret to developing ISAPI web server extensions: do them first as plain CGI, so that you have an EXE file you can run in the debugger with ease, and also so that when you exit the program the web server doesn't have to be stopped before you can make any changes. ISAPI applications are actually specialized DLLs which are loaded by the web server when first called, and which thereafter belong to the server and cannot be unloaded without unloading the server itself. Also, an access violation in your code will probably crash the server and perhaps even your whole system. The Delphi `TWebModule` requires you specify whether you are making a CGI or an ISAPI application when you create one. There are a few programming differences between doing CGI or ISAPI, but for the most part anything you build as a CGI with Delphi's `TWebModule` you can convert to ISAPI without any changes other than the include files at the top of the project file.

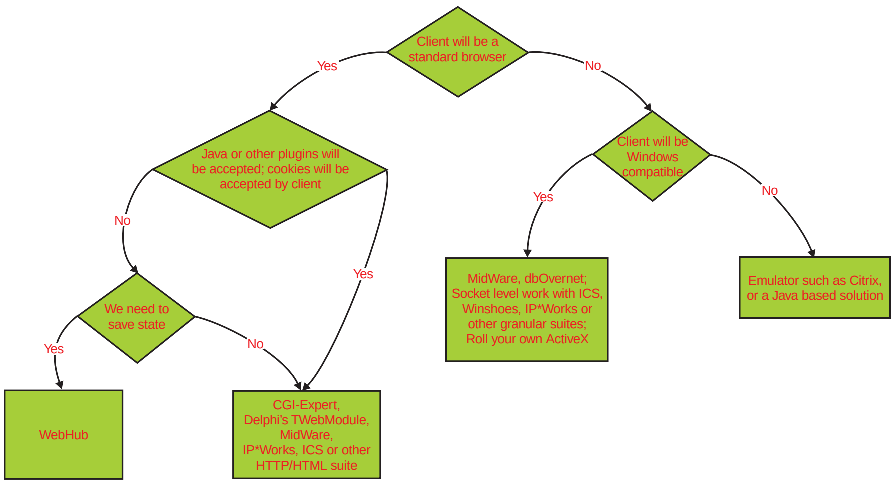
There was one aspect of working out this second chunk of programming that bothered me. I had to somehow pass the user name and password back and forth between the client and the server each and every time control moved between them. This is due to the fact that HTTP is stateless. When a web server receives a request, it sends a response based on the header and contents of the request and then promptly forgets the whole thing and begins waiting patiently for the next request. HTTP 1.1 is supposed to allow saving state, but only the latest browsers support this. Out on the bleeding edge of web technology there are many more methods of overcoming this state-saving problem, but even now one cannot count on the customer having the appropriate level of web browser.

Cookies are an easy way to overcome the problem, being nearly universally supported by the browsers one is likely to find. However, cookies have unjustly earned a less than good reputation and some people turn off their cookies. While it was highly unlikely that any of our customers would believe that the government was going to take over their computer and use it to listen in on their private dinner conversations, it was entirely possible that some of them would have turned their cookies off.

I had worked out a method for encoding the user name and password into a hidden form variable, and this technique would certainly work, though it also means that anyone who knows about the `View | Source` option on the browser menu could lift the encoded name and password and use them (assuming they figured out what they were) for their own nefarious purposes. By including date and time information in the encoding scheme, the danger from that can be minimized, but the more I worked on it, the more I realized there must be a better way to approach the whole 'web browser client with web server server' architecture. So I went online and searched for alternatives.

...And What's Not

Since Delphi was the language of choice for this assignment, I found several component sets which fell roughly into two basic categories: highly granular or highly integrated. The granular component sets generally offer many components, each customized to one specific protocol, while the integrated suites usually offer fewer but more powerful components. Cutting across that categorization is another two-way split in what is offered to Delphi developers: socket programming or HTTP integration. Socket programming is what you want to use when you need 100% control over the client: you can have a Delphi executable at both ends, and in fact you are generally not using the HTTP protocol. If you want the client to be an off-the-shelf browser, however, you need a suite which is centered on the HTTP protocol. The decision chart in Figure 1 summarizes these categories and their relationship to your upfront design decisions.


Client will be a standard browser
Yes
No
Java or other plugins will be accepted; cookies will be accepted by client
No
Yes
We need to save state
Yes
No
WebHub
CGI-Expert, Delphi's TWebModule, MidWare, IP*Works, ICS or other HTTP/HTML suite
Client will be Windows compatible
Yes
No
MidWare, dbOvernet; Socket level work with ICS, Winshoes, IP*Works or other granular suites; Roll your own ActiveX
Emulator such as Citrix, or a Java based solution

◆ *Figure 1: Decisions, decisions...*

Piette's Internet Component Suite And MidWare Suite

The highly granular component sets generally offer a multitude of highly focused individual components. For example, Piette's Internet Component Suite, or ICS, (freeware, available at www.rtfm.be/fpiette) contains 22 files in the DPK file as part of a package (not counting C++ support) containing some 250 source code files. Certainly an excellent place to start if you want to build specialized web applications. Much as I wanted to take the time and effort to learn web programming from the ground up, my schedule did not allow me this kind of luxury. To give you a feel for the kind of detail and flexibility you get with ICS, browse through the list from the readme.txt file in Table 1.

In addition to the components he provides plenty of sample applications using them. This set of components also offers both kinds of programming support for Delphi developers. You can bypass the whole web business and work directly with sockets, or you can work through the web using the HTTP protocol.

Next I turned to another offering from Monsieur Piette, his freeware MidWare suite. This is an extension of the ICS along with the sample application programs near and dear to the heart of anyone venturing out into n-tier land. Again, a great place to start from the ground up. But I would be using the Client/Server version of Delphi and did not need to build an object request broker from scratch. If you do not have the Delphi Client/Server edition, MidWare would be an excellent choice.

dbOvernet

dbOvernet, a commercial n-tier development tool, falls into the second category: fewer, but more powerful components. On the client side there is an `AppServerClient`, a `Dataset Buffer` and a `Dataset`; on the server side is an `ApplicationServer` and a `RequestBroker` along with support components, including a `Socket`, a `RequestBroker`, and some SQL-specific support stuff. This package is an excellent alternative to the rather high cost of the Delphi Client/Server, but it is a socket solution and does not address the HTTP actions I needed to use a browser client. If you are planning to build your 'web' application without using the web, this one is a winner.

CGI-Expert

Lars Akerman's CGI-Expert (www.cgiexpert.com), despite its name, also

◆ *Table 1: Piette's Internet Component Suite.*

WSocket.pas	Winsock component: TCP, UDP, DNS,...
HttpProt.pas	HTTP client protocol: used by the web
FtpCli.pas	FTP client protocol: file transfer
FtpSrv.pas	FTP server protocol: file transfer
Ping.pas	ICMP echo protocol: ping a host
Pop3Cli.pas	POP3 client protocol: get mail from mail server
MimeDec.pas	MIME component: decode file attach, use with POP3
SmtpCli.pas	SMTP client protocol: send mail to server
NntpCli.pas	NNTP client protocol: send/receive newsgroup messages
TnCnx.pas	TELNET client protocol: terminal emulation protocol)
TnScript.pas	TELNET client protocol: with automation
EmulVT.pas	ANSI terminal emulation in a control
TnEmulVT.pas	TELNET and ANSI terminal emulation combined
FingCli.pas	FINGER client protocol: Find information about user
Wait.pas	A kind of progress bar

does ISAPI quite easily, with nearly the same flexibility of switching between a CGI development project to an ISAPI deployment project as is provided in the Inprise WebModule. The price is considerably better, however. He offers a freeware version that does only the most basic HTTP request and response stuff, though it does feature both THTTPFileFilter and THTTP MemoFilter components which behave just as flexibly as, though differently to, the PageProducer components Inprise offers. I would tend to categorize this offering as highly granular and HTTP oriented, though the degree of integration is higher than I found in ICS. A friend who owns a small web farm was looking for a tool to hit the Internic name registry site and find out if a name was available. I was able to build a CGI that did the job for him in a few hours using CGI-Expert, though I did have to buy a license because the freeware offering was missing the piece that would let me send off an HTTP request to somewhere else on the web while still in the process of building a response to an immediate request. The problem of state did not come up in this little application, and the only way to save state with this set of components would be to use cookies, or the technique I had been considering earlier.

IP*Works

IP*Works from DevSoft (at www.dev-soft.com) is another highly granular offering of components, more extensive than either ICS or CGI-Expert, but also costing more. By this time, I was looking only at HTTP support and how the state saving issue was handled. IP*Works seemed to have the kind of HTTP support I would need, but offered only cookie management for saving state.

WebHub

Then I turned to WebHub from HREF Tools (www.href.com). This Delphi Web application development environment is highly granular and highly integrated at the same time. You could do socket level stuff with what's available, but the primary focus is taking over an HTTP Server and turning it into an extension of Delphi.

At first glance, especially after installing it and discovering the huge number of new components on your palette, it would seem HREF are following the highly granular approach. However, WebHub does quite a bit more, and in fact is the *only* tool I'd recommend for a heavy duty client/server type of application. However, the learning curve is somewhat akin to climbing a mountain, and building a web application in WebHub requires a full time concentrated effort.

Before discussing WebHub in more detail, though, I'd like to touch on three other areas. One is the actual solution we used for our project, the conversion of a Windows 3.1 client/server app to use an HTML 3.2 browser as the client. I'd also like to touch on Active Server Pages and discuss briefly the design constraints of writing a web application in Delphi.

Diversion: Citrix And ASP

After downloading and installing trial versions of most of the above products and working though the various approaches to saving state, I'd recommended we go with WebHub. However, one of the higher-ups was not pleased with the idea that his code shop would now be maintaining one set of code for users directly attached to the network and an entirely different set of code for users coming in through the internet. Enter Citrix and WinFrame. This product is essentially an emulator: once the user is connected to the Citrix server, what shows up on the user's screen is an exact graphical image of the application running on Windows, whether they come in via a network connection or via an internet connection. In addition, it doesn't matter whether the end-user's machine is Windows or MacIntosh. The screen, keyboard and mouse respond as if they were sitting in front of a Windows machine dedicated to running your app.

The downside is that one has to be willing to sink rather large amounts of money into hardware costs and licensing fees. One would need a dedicated person to handle the Citrix installation and maintenance, and perhaps a different person to figure out the licensing requirements. I didn't do the numbers, but I wouldn't be surprised if it turns out that even with these huge hardware and licensing costs, they are still cheaper than software maintenance for two sets of code over something like a ten or twenty year projection. Visit www.citrix.com if you are interested in taking it further.

An entirely different approach to web applications is Active Server Pages (ASP), which are at present limited to Microsoft IIS servers and would appear at first glance to require JavaScript or Visual Basic and all kinds of special Microsoft tools (as an aside, I came across something called Chilisoft, visit www.chilisoft.com, just before sending in this article: they promise ASP support for HTTP servers other than IIS).

The only requirement is that JavaScript or a related dialect is required to activate an ASP object, but the object itself can be written in Delphi and, from within the object, one has access to all of IIS through COM interfaces. In addition, Microsoft is building (and probably will be building for several years yet) support structures to support web applications: the Global.asa file, for example, can be used to set up both application and session specific objects and common functions, although the session control at this time is still cookie based. All this support infrastructure is visible to the Delphi ASP object through COM interfaces. Best of all, one does not need to require the user to have a browser that understands JavaScript, one can instead use server-side Java script to activate your Delphi written ASP objects. Once one has imported the appropriate type libraries, one's Delphi program can take over a website and control nearly 100 percent of what goes on, from within the Delphi code. But not 100 percent, because what must arrive at the user's browser is a text file with appropriate HTML tags.

Which leads to the constraints of designing a web application powered by Delphi (or Visual Basic, or C++, for that matter). The central constraint is that the user input has to be controlled from the browser, an interface which is not under the control of Delphi. All user input will come in through input fields within the HTML `<FORM>` tag. There are some other ways to get user input; for example, the user can type a URL with arguments, but in any case, one does not have an event based environment and one does not have access to keystrokes or mouse events. If you are able to require your users to have the latest browser, you do have applets, client side JavaScript, DHTML, XML and all the other neat things out on the bleeding edge. However, you will soon find yourself maintaining code in several different languages, several of them confusingly similar. Plus you get to have the fun task of writing code to figure out what kind of browser is currently using your application, and having at least two branches of code to handle Netscape versus MSIE.

Back To WebHub

In practical terms, the most difficult hurdle to overcome is realizing that even the simplest logical branching, such as *'If user selects A, then show page 23 else show page 24'*, cannot be put into your Delphi code. The first part, the *'selects A'*, has to be read from the HTTP `Request` object when it arrives from the user. You'll then need some Delphi code to translate what arrived from the user into a `Boolean` expression before you can get to the point when you can send page 23 or 24 back. Object orientation is basically out of the door and you are back to tedious and detailed procedural processing. You will, of course, wrap everything in neat and orderly classes, but most of your maintenance work is going to be very much step-by-step tracing back and forth between the contents of an HTTP `Request` header and your program logic.

Then you pick up the WebHub manuals and start to get excited as you see they have done most of the wrapping for you. Your HTML, instead of being in separate text files, is not only contained within your web app, but you can change it on the fly. Not only during design-time, but even while your web application is live and doing live transactions over the internet!

On top of that, session tracking is very neatly handled by a special engine which WebHub calls a runner. The runner does much more than act as a central traffic cop, making sure that each user is tracked separately without having to bother the user's machine with cookies. The runner mechanism also allows you to have multiple Delphi applications running simultaneously, each servicing a different aspect of your website. I don't have enough space to describe even half of the features of WebHub, but the home site, www.href.com, has plenty of resources and the authors are very responsive in helping users get over the hurdles. WebHub is a complete web application development environment and includes their own special dialect of HTML that handles server side manipulations. I'm convinced that if you are willing to put in the time to master them, the tools you get when you buy into WebHub will enable you to do anything you want, and with a responsiveness that will make your customers more than happy.

One aspect of WebHub's session tracking mechanism deserves special mention for those working with a commercial database back end. All of the heavy duty database back ends charge licensing fees based on some formula that means you pay for each connection to, or transaction between, your application and the database. It's easy to open a new connection for each stateless iteration of a user-server transaction, but this will drive your licensing costs sky high in a very short time. You need to keep track of who the current request is coming from, and you need to have some special Delphi code in place to link that user with the same database connection you initiated when the user first signed on. More than likely, this will involve threads and other delicate operations in addition to the functional logic of your program. WebHub's runner engine and session saving mechanism can be set up so that this linkage is taken care of by WebHub. There are other ways of minimizing licensing costs, such as pooling database connections, but then you lose any auditing mechanism built into the database.

Conclusions

Your choice of after market Delphi web development components depends on what kind of application you want to put on the web as well as what kind of coding you want to do. If you want, for example, to bypass the entire HTTP/HTML muddle and use the internet as your network connection, then you probably want something based on socket level code and you will find MidWare, dbOvernet, or IP*Plus to be the kind of tool you want. On the other hand, if you want to take advantage of the HTTP/HTML magic that has taken the world by storm, and you have a relatively simple application, CGI-Expert is probably your cup of tea. On the third hand, if you have a complex database application to 'webify', you can either use the HTTP pieces of something like MidWare or IP*Plus, or you can jump in and let WebHub handle HTTP for you and concentrate on Delphi code and customized HTML.

Brandon Smith works in Jefferson City, Missouri for Rose International, on Delphi object infrastructure building. Find him at delphi@synature.com

www.itecuk.com

News, Contacts, Back Issues, What's Coming and much more, with a full site search capability so you can find what you need quickly and easily