

Borland[®]

Internet Programming with Delphi

by Marco Cantù
(<http://www.marcocantu.com>)

Table of Contents

The Challenges of Internet Programming	1
Where does Delphi Fit?	2
Core TCP/IP Support	3
Client Side Protocols Support	4
Server Side Protocols Support	4
Client Side Web Support	5
Server Side Web Development	6
Generating HTML pages	6
The WebBroker Framework	7
Supporting Microsoft's Windows DNA architecture	9
InternetExpress: XML, ECMAScript, and DHTML	10
Third-Party Web Server Extensions	13
Delphi in Action on the Internet	13
Conclusion: The Delphi Advantage	14

Borland Delphi is known to be a great environment for the development of stand-alone and client-server applications on the Microsoft Windows platform. Its virtues range from full OOP support to visual development, in a unique combination of power and ease. However, the new frontier of development is now Internet programming. What has Delphi got to offer in this new context? Which are the features you can rely upon to build great Internet applications with Delphi? That's what this paper intends to reveal. We'll see that Delphi can be used:

- For direct socket and TCP/IP programming;
- In conjunction with third-party components that implement the most common Internet protocols, on the client or the server side;
- To produce HTML pages on the server side, with the WebBroker and Internet Express architectures;
- As well as to work with Microsoft's core technologies, including MTS, COM, ASP, and ActiveX.

The Challenges of Internet Programming

Internet programming poses new challenges to traditional developer environments and to the programmers using them. There are issues related with the implementation of standard protocols, the use of multiple platforms (Microsoft Windows accounts for most of the client computers on the Internet but only a fraction of the servers), and the licensing schemes of some RDBMS systems.

Delphi

white paper

Most of the problems, however, relate with HTTP development: Turning existing Windows applications into applications running within a Web browser is more complex than it might seem at first sight. The Web is stateless, the development of user interfaces can be quite an issue, and you invariably have to consider the incompatibilities of the most widespread browsers. A new platform specifically aimed at areas of Internet programming (typically the HTTP world) has emerged. These environments favor server side development, often also allowing the inclusion of database information within web pages. A common solution is to write HTML pages with special “scripting” tags included, which are going to be expanded by an engine on the server. Interesting for small projects, most of these technologies have limited scripting power, and force you to mix HTML code and scripting code, and GUI scripting code with database oriented code. On larger applications, this lack of separation among the different areas of a program is considered to be far from a good architecture.

Moreover, Microsoft’s DNA is going to be replaced by the new Microsoft dotNET (or “.NET”) architecture – a new name and approach that seems to imply that the previous architecture had indeed serious limitations. DotNET is apparently going to be more “open” and stresses a lot the importance of XML, including pushing the support for the SOAP (Simple Object Access Protocol) invocation protocol. Another key element of dotNET is that COM is apparently going to be phased out (not a nice idea for people who’ve invested in the approach Microsoft was pushing yesterday).

Even with the advent of dotNet, Microsoft’s DNA architecture, based on ASP for HTML scripting and COM/MTS/COM+ for database manipulation, offers a higher perspective, but is limited to the Windows platform and Microsoft’s own IIS Web server, tends to work primarily with the Internet Explorer browser. The current incarnation of DNA suffers from several limitations, including DCOM unfriendliness with firewalls, complex configuration and administration, some tie-in with Microsoft’s technologies, databases included, and limited scalability. Also, the overall architecture, with the separation of many layers

partial with status and partially stateless, seems to be still limited for the challenges of the Internet.

Where does Delphi Fit?

With this rather complex situation going on, where does a “traditional” development platform like Delphi fit? The goal of Delphi in the Internet age is to bring the some power, flexibility, ease of use, and database support.

- The power of Delphi comes from the fully compiled code, different from many script-based technologies, and from its fully object-oriented architecture (which is not an after thought, but has been the foundation of the language and its libraries since version 1.0). Delphi natively compiled applications are simple to deploy, as they are generally made of a single self-contained executable code (with no extra runtime libraries and custom component files). Actually, dividing the single EXE in multiple packages is a useful option that is offered by Delphi, which programmers can fine-tune to choose the best deployment solution.
 - The flexibility of Delphi comes from a support not limited to HTTP but open to most Internet protocols, as the development environment allows you to write lower level code to support new protocols, as the developers of many native third-party components have done.
 - The ease of use of Delphi comes from the component-based environment. Writing a mail client (eventually bound to a web page) simply implies adding a couple of components to your program, setting a few properties, and writing very little code. Some of the samples found in Delphi and the third party components are basically full-featured email programs!
- With the InternetExpress technology of Delphi 5, the ease-of-use has been extended to allow the visual development of HTML front ends based on data sets.
- Database and client/server support has always been one of the strongest features of Delphi and client/server architectures remain the core of most Web applications and Internet sites. Actually, if you’ve built your Delphi

applications by separating the user interface from the back end (typically using Data Modules for the latter) you are ready to plug in a new user interface to your existing “business rules” code.

This is particularly true for multi-tier MIDAS applications, which separate the business logic and the user interface in two separate applications, running on the client and application server computers. Using the InternetExpress technology, as we’ll see, you can simply build a new front end for a Web server application, and make it available to the client browsers.

Leveraging your existing Delphi code and allowing you to build Windows and browsers based front end for the same core application, are two key reasons to adopt Delphi as your Internet development platform. But they are not the only reasons, as other areas of Internet development are equally served by the technologies included in Delphi.

Finally, with the forthcoming Kylix project (see <http://www.borland.com/linux>), Borland are providing a “Delphi for Linux”, allowing your server side applications to run equally well on Microsoft Windows or Linux operating systems. Delphi will be able to leverage features of the two platforms without any tie-ins to a specific operating system, allowing your Web server applications to run on the two most widespread operating systems for Internet servers.

Core TCP/IP Support

The common factor for all Internet and Intranet applications is communication over TCP/IP sockets. Most of the time the communication is constrained by a set of rules, known as a communication protocol. For example, SMTP and POP3 are two very simple protocols for sending and retrieving mail messages, defined by the Internet standard bodies.

Using Delphi you can:

- Implement the client and the server side of a proprietary protocol, using the TServerSocket and TClientSocket components, found in the Internet page of the component palette. This is handy for distributed applications, but

creates a closed system, in which other programs not written by you cannot interact (which might be an advantage or a disadvantage, depending on the situation). That is, of course, unless you want to define a new protocol and publish the specs for others to “join” you.



Figure 1: The Internet page of Delphi’s component palette, hosting the socket and HTML producer components.

- Implement an existing protocol on the client or on the server side. This can be done again with the generic socket components mentioned above, but its generally accomplished by using protocol-specific Delphi components provided by third parties, some of which are even pre-installed in the Delphi IDE.
- Support the HTTP protocol, the core of the Web, and the HTML file format. As these play such a major role, I’ll cover them separately from the other protocols.

The support for TCP/IP and socket programming in Delphi is as powerful as using the direct API (Winsock, in case of the Windows platform) but far simpler. The socket components, in fact, shield the programmer from some of the complex technical details, but surface the Windows handles and low-level APIs, allowing for custom low-level calls.

Writing simple programs with socket support in plain C calling the Windows APIs requires hundreds of lines of code, while using the Delphi socket components, a few lines of code will suffice, even for complex tasks. That’s the standard advantage of component-based development. Also, building a simple user interface for the program is often trivial in Delphi. With other development environments, you need to program the socket in a low-level language (such as C) and then write the user interface with a different visual tool, integrating the two and requiring knowledge of multiple languages.

Client Side Protocols Support

To develop the client side of Internet applications, Delphi provides you ready-to-use components. There are multiple sets of native VCL components you can adopt, all based on a similar philosophy:

- The NetMaster components are pre-installed in the Delphi environment (see the FastNet page of the component palette), and include client-side support for the most common Internet protocols (including POP3, SMTP, NNTP, FTP, and HTTP).
- The Indy open source components (“Internet Direct”, previously called WinShoes and now “federated” with the Jedi Project) are available on Delphi 5’s Companion CD and from their web site (<http://www.nevrona.com/indy>). It has been announced that Indy will be included by default in Delphi 6 and Kylix (Delphi IDE for the Linux platform).
- The free ICS components (“Internet Component Suite”, available at <http://users.swing.be/francois.piette/icsuk.htm>, include the complete source code) and are designed and maintained by Francois Piette and form another set of very popular Delphi components, supporting most Internet protocols.
- A few other commercial offerings, including IP*Works components (<http://www.dev-soft.com/ipwdlp.htm>) and Turbo Power’s Internet Professional (<http://www.turbopower.com/products/IPRO/>).

Some of these components map directly to their own WinSock wrappers, others also use the WinInet library, a Microsoft system DLL that implements support for the client side of FTP and HTTP protocols. Regardless of the set of components you are going to use, they are really quite simple to work with. If you have an existing application, and want to mail-enable it, just drop a couple of components onto your form (or data module), set their properties (which include the indication of the mail server you want to connect with) and write few lines of code.

For example, to send email with NetMaster’s component, you can use the following simple code:

```
// component properties
```

```
object Mail: TNMSMTP
  Host = 'mail.server.com' // your web
  service
  Port = 25
  PostMessage.FromAddress =
  'marco@marcocantu.com'
end

// code to send the above email
message
Mail.PostMessage.ToAddress.Add
('david@borland.com');
PostMessage.Subject := 'Borland
Community Site';
PostMessage.Body.Add ('Hi David, I
wanted to ask you...');
Mail.Connect;
Mail.SendMail;
Mail.Disconnect;
```

In short, these are the advantages of using Delphi for supporting Internet client applications:

- Choice among various offerings of components (some of which are totally free and open source)
- Easy integration with existing applications
- Easy development of new and specific user interfaces, with Delphi visual and object oriented architecture

Server Side Protocols Support

Besides supporting web protocols in existing applications, or writing custom client programs specifically for them (as a completely custom email program), in a corporate environment you often need to customize Internet server applications. Of course, many of the available pre-built servers can be used and customized, but at times you’ll need to provide something that existing programs do not support.

In that case, you might think of writing your own server, if only it wasn’t so complex. Using Delphi and a set of server side components you can build custom servers with only limited extra effort, compared to a client program, and achieve (or at times exceed) the performance of professional quality Internet server programs.

Server side components were pioneered by Jaadu (<http://www.jaadu.com>), which offers a web server component and are now available in the Indy component set (discussed above). There is also a set of highly optimized native Delphi

components, called DXSock (<http://www.dsock.com>), specifically aimed at the development of Internet server programs. Some of the demonstrations of these component sets are actually full-fledged HTTP, mail, and news servers.

Client Side Web Support

If many Internet protocols are important, and email is one of the most commonly used Internet services, it is undeniable that the Web (that is, the HTTP protocol) is the driving force of most of the Internet development. Web support in Delphi is particularly rich. Here, we are going to start by exploring the features available on the client side (to integrate with existing browsers) and then we'll move to the server side, devoting plenty of time to the Web server development that you can do with Delphi.

- The HTTP components available in most suites allow you to create a custom browser within your application: You can reach existing Web sites and retrieve HTML files or post custom queries. At this point you can send the HTML content returned by the HTTP server to an existing browser or integrate a custom HTML processor or HTML viewer within your application. The ways in which can apply are as follows:
- Sending an HTML file to Microsoft's Internet Explorer or Netscape Navigator, either using it as an external application, by calling the ShellExecute API function:

```
ShellExecute (Handle, 'open',  
'c:\tmp\test.htm', '', '',  
sw_ShowNormal);
```

or integrating the Internet Explorer ActiveX control, surfaced in Delphi as the ready-to-use WebBrowser component.
- Processing HTML in a custom way, to extract specific information; this is useful in case you don't need to show the HTML to a user, but want to process it, eventually extracting specific information from it.
- Showing the HTML within your program using a native Delphi component (so that end users don't need to have Internet Explorer installed). These components are available

by third parties, with the most well known HTML viewer component being offered by David Baldwin (see the Web site <http://www.pbear.com>).

The reverse of integrating a browser within your application, you can integrate your application within the browser. This is rather easy to accomplish by using Internet Explorer and the ActiveX technology. Delphi supports this technology in full using the ActiveForm technology. An ActiveForm is built in the same visual way that a plain Delphi form is constructed, but an ActiveForm is hosted within an HTML page of Internet Explorer. You can even move existing programs to the Web by hosting their main form within an ActiveForm.

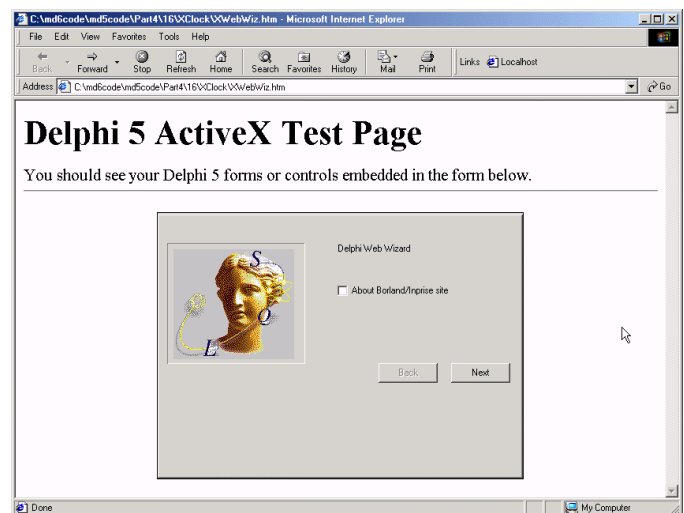


Figure 2: an example of a wizard built with Delphi (it is based on a PageControl component) and deployed within an ActiveForm. The buttons allow you to reach different pages of the form, without moving outside of the browser's page.

This Microsoft specific technology (ActiveX is not supported by other browsers) can simplify the deployment of simple Delphi applications within an Intranet, as users can download the programs they need by pointing their browser to specific pages. The ActiveX technology, however, is not well suited for the Internet, as too many people have different browsers or operating systems, or disable this feature in their browser for

fear of the potential harm caused by the automatic execution of programs that are downloaded from the Web.

- The advantage of Delphi in this area is that, once more, it allows you to customize your existing programs to take advantage of the Internet, and make them work seamlessly with Web browsers. Also, if you choose not to use the ActiveX technology, you won't be tied to any particular browser or platform.

Server Side Web Development

As anticipated, all the development related to Web servers is by far the most important area of Internet development, and again one where many alternative solutions are available. Delphi has offered developers strong server side Web development since version 3, with Delphi 5 being a mature environment for building Web server extensions. Delphi includes multiple technologies to support server side development, so I'm going to cover:

- The HTML Producer components
- The WebBroker technology for building CGI, WinCGI, and ISAPI/NSAPI server side extensions
- The Internet Express technology (introduced in Delphi 5) for building database-oriented server side applications, based on standard technologies such as XML and ECMAScript (formerly known as JavaScript).

Generating HTML pages

Delphi includes several components aimed at the generation of dynamic HTML pages. There are two different sets of producer components, page-oriented and table-oriented ones. When you use a page oriented HTML producer component, such as the PageProducer, you provide the Producer component with an HTML file with custom tags (marked by the # character). You can then handle the OnTag event of the component to replace these custom tags with specific HTML code.

The following is sample code for this OnTag event:

```
procedure
TFormProd.PageProducer1HTMLTag(Sender:
TObject;
```

```
Tag: TTag; const TagString: String;
TagParams: TStrings;
var ReplaceText: String);
var
nDays: Integer;
begin
if TagString = 'date' then
    ReplaceText := DateToStr (Now)
else if TagString = 'expiration'
then
    begin
        nDays := StrToIntDef
(TagParams.Values['days'], 0);
        if nDays <> 0 then
            ReplaceText := DateToStr (Now +
nDays)
        else
            ReplaceText := '<I>{expiration
tag error}</I>';
        end;
    end;
```

This code handles a plain tag, date, which is replaced with the current date, and a parametric one, expiration, which includes a parameter indicating the number of days the information on the page remains valid. The HTML for this custom tag will look like:

```
Prices valid until <b><#expiration
days=21></b>
```

The output will be something like: "The prices in this catalog are valid until 12/24/2000", as you can see in Figure 3.

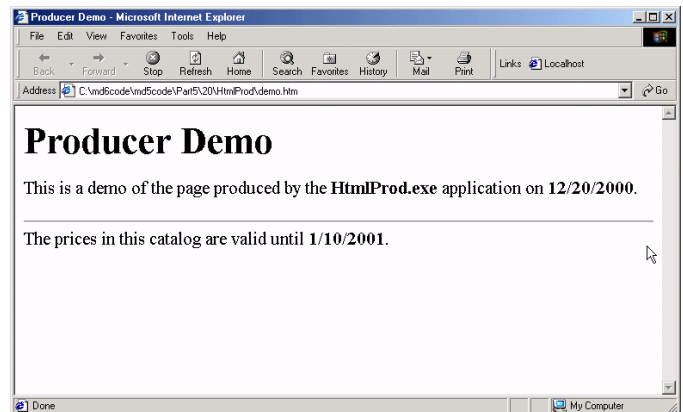


Figure 3: The HTML file generated by a PageProducer component.

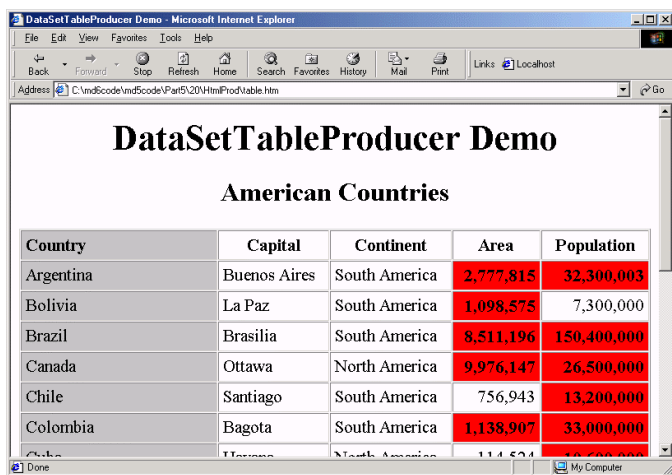
The advantage of this approach is that you can generate such a file using the HTML editor you prefer, simply adding the custom tags. Notice also that the same OnTag event handler can be shared by multiple producer components, so you don't need to

code the same tag expansion multiple times within the same program.

A second component of this group, `DataSetPageProducer`, can automatically replace tag names with the values of the fields of the current record of a dataset.

Another group includes HTML table oriented components. The idea is to convert automatically a dataset (a table or the result set of a query or stored procedure) into an HTML table. Although a standard conversion is supplied, you can add custom tags and styles for the grid, each of the columns, and even specific cells of the table. The customization is similar to that which can be applied to a visual `DBGrid` inside a Windows application. For example, the following code turns all the cells of the second column that have a value exceeding 8 digits red in color (with the effect you can see in Figure 4):

```
procedure
TFormProd.DataSetTableProducer1FormatCell(
  Sender: TObject; CellRow,
  CellColumn: Integer;
  var BgColor: THTMLBgColor; var
  Align: THTMLAlign;
  var VAlign: THTMLVAlign; var
  CustomAttrs, CellData: String);
begin
  if (CellColumn = 1) and (Length
  (CellData) > 8) then
    BgColor := 'red';
end;
```



Country	Capital	Continent	Area	Population
Argentina	Buenos Aires	South America	2,777,815	32,300,003
Bolivia	La Paz	South America	1,098,575	7,300,000
Brazil	Brasilia	South America	8,511,196	150,400,000
Canada	Ottawa	North America	9,976,147	26,500,000
Chile	Santiago	South America	756,943	13,200,000
Colombia	Bagota	South America	1,138,907	33,000,000
Costa Rica	Turkey	North America	114,524	2,200,000

Figure 4: The output of a `DataSetTableProducer`, with custom colors for specific cells.

The second component, the `QueryTableProducer` is specifically tailored for building parametric queries based on input from an HTML search form. The parameters entered in the form are automatically converted by the component into the parameters of a SQL query and the resulting dataset is formatted in an HTML table: all this complex work can be set up with no custom coding!

Advantages

- You can write the basic HTML code with the editor you prefer and simply include custom tags.
- You are not mixing scripting code with the HTML code, but keep them totally separate. The HTML simply includes a placeholder for the code that is going to be generated.
- The script is replaced by full-performance compiled code.
- Using this technique you can easily access database data, and render the result of complex queries in HTML tables with no custom coding!

Further Notes

The HTML producer components can also be used to produce static web pages, that is plain HTML files that can be placed on your web server and not server dynamically by a program. Notice also that beside HTML files, Delphi programs can produce JPEG files, using the `TJPEGImage` component. Again, these files can be placed on a server or produced dynamically from a server extension. The generation of images includes the generation of the complex business graphs, available through the native `TeeChart` components.

The WebBroker Framework

The development of Web server extensions (that is, custom applications seamlessly integrated with a Web server) can be based on multiple competing technologies, including:

- CGI (Common Gateway Interface, common on UNIX boxes),
- WinCGI (the Windows flavor of the same technology),

- ISAPI Internet Server API, libraries specifically tailored for Microsoft's own IIS) and NSAPI (the corresponding API offered by Netscape's web server),
- Apache modules (the same idea, but for the open-source Apache Web server) – this standard is not currently supported by Delphi, but Borland has revealed plans to support it in Kylix, the project for a Linux version of Delphi.

The problem with most of these technologies is that even if they are all based on the HTTP protocol, the way you receive the same information and make it available to the Web server changes substantially. For this reason, Borland has built in the VCL a small object-oriented framework, called WebBroker, which removes those differences. You write all of your code targeting a few generic base classes, and ask Delphi to provide you a specific implementation for, say, CGI or ISAPI. This means you can move your programs (even complex ones) from one of these technologies to another simply by providing a different project source code and a few lines of code.

- Once the “bridge” includes Linux based servers beside Windows ones, the WebBroker technology will be able to bridge a large variety of web servers on multiple operating systems.

Not only does WebBroker provide a bridge among multiple technologies, it also provides a lot of core routines and facilities, to simplify server side development. For example, you can ask for a specific value inside a query string by writing:

```
stringName := Request.QueryString  
[ 'Name' ] ;
```

instead of having to parse a complex string yourself. This is just one simple example, there are a great many timesavers within the WebBroker architecture, to enable you to really speed up development.

Consider also that the WebBroker architecture is generally used in conjunction with the HTML producer components. The development of a program which executes a query on a SQL server, formats it using an HTML table, and returns it from a

server side application takes probably less than 20 mouse clicks and almost no coding!

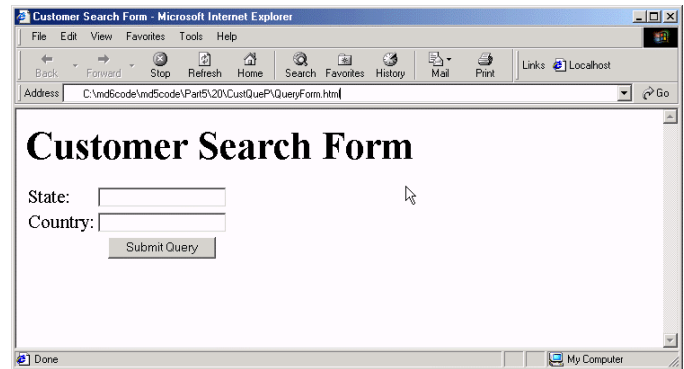


Figure 5: A query form like this one can be directly tied to the parameters of an SQL query, via a QueryTableProducer component and with almost no coding!

For example, if you have the following HTML file with a table (shown in Figure 5), you can hook it with a script (called CustQuery.exe) to process the request. This is the HTML code, with a table having two input fields:

```
<html><head>  
    <title>Customer Search  
    Form</title>  
</head>  
<body>  
  
    <h1>Customer Search Form</h1>  
    <form  
    action="/scripts/CustQuery.exe/search"  
    method="POST">  
    <table>  
    <tr><td>State:</td>  
    <td><input type="text"  
    name="State"></td></tr>  
    <tr><td>Country:</td>  
    <td><input type="text"  
    name="Country"></td></tr>  
    <tr><td></td>  
    <td><center><input  
    type="Submit"></center></td></tr>  
    </form>  
  
</body>  
</html>
```

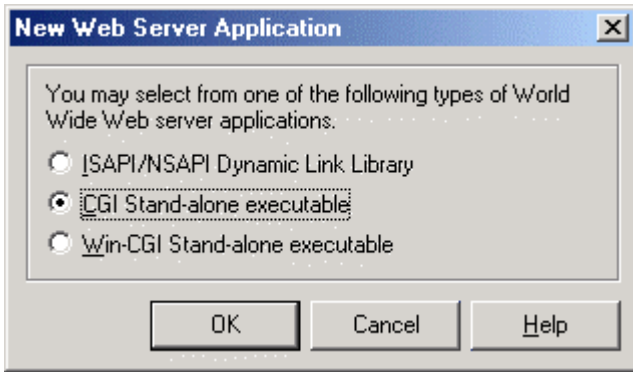



Figure 6: The choices offered by Delphi's Web Server Application Wizard.

Now you can create a Delphi WebBroker application, using the Web Server Application Wizard (see Figure 6), and choosing CGI (or whatever technology you prefer). Inside the WebModule Delphi will create and open for you, you can add an action, by right-clicking on the Actions item of the Objects Tree View (above in Figure 7) or using the add button or local command of the resulting actions list editor (below in Figure 7). You can open the action list editor double clicking on the WebModule itself.

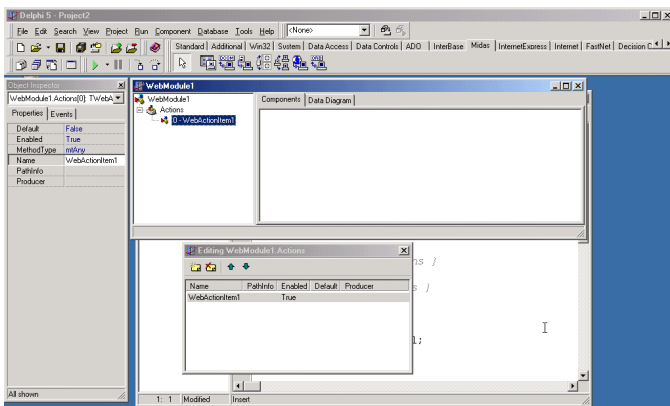


Figure 7: The actions of a WebModule can be seen in the Objects Tree View (at top) and in the actions list editor (at bottom). Their properties are set in the Object Inspector (at left).

Set the action with the “/search” value in the PathInfo property. This can be connected (using the Producer property) with a

QueryTableProducer component added to the data module. This component, in turn, is hooked to a Query component, via its Query property.

The Query component will be executed when the action is invoked, passing to its Params the QueryStrings or ContentStrings parameters of the WebRequest. This means that Delphi will extract the values entered in the HTML input boxes and copy them to the query parameters having the same name. So, we can use a query like the following, with parameters having the same name of the input fields (see again the HTML code above):

```
SELECT Company, State, Country
FROM CUSTOMER.DB
WHERE
    State = :State OR Country = :Country
```

That's all! Even with no Delphi code we've obtained an HTML front end for a database search. By customizing the HTML table output, attaching a style sheet, adding extra code for custom processing, you can build a professional version of this program within a few hours.

In short:

- WebBroker allows you a single source code solution for multiple technologies: CGI, CGI-Win, ISAPI/NSAPI, and also Apache Modules moving forward.
- Combined with the HTML producer components, your server side applications can easily produced HTML pages, particularly showing database data.
- Your WebBroker code will be portable to Linux with little effort, once Kylix is released.

Supporting Microsoft's Windows DNA architecture

Besides talking about the WebBroker framework, and the Internet Express technology I'll discuss later, Delphi has a full and high quality support for the entire Windows's DNA architecture (which will be superseded by the dotNet architecture, but is probably going to remain in use for quite some time). Delphi has traditionally been the first visual

development environment to support ActiveX and MTS technologies, even before Microsoft's own visual tools provided such support.

Not only this, but Delphi's simplified and yet complete COM support is still unparalleled in the industry. Based on this high-quality low-level COM support, Delphi provides support for most COM-related technologies, such as Windows Shell programming, Automation, Active Documents, ActiveX (and also the Web-oriented ActiveForm technology I've already covered), MTS and COM+, and many others.

Using Delphi you can write MTS object defining your business rules and database integration code, provide a layer of ASP-enabled COM objects to generate HTML and user interface elements, and wrap everything in ASP scripts (Microsoft's Active Server Pages technology). This corresponds to embrace Microsoft's proposal in full, with high quality support.

Giving this past track record, we can probably expect a future release of Delphi to fully support COM+ (although I have to say you can already write COM+ applications with Delphi 5, with a little extra effort) and other emerging Microsoft standards.

In short:

- Delphi's support for the Windows platform is complete, including the support for the entire COM architecture and the Windows DNA model.
- Delphi's ability to write low-level COM code makes it possible for you to target new standards without having to wait for Borland support within the development environment. You can hardly say the same for any other visual tool.
- While fully supporting Microsoft technologies, Delphi allows you to avoid the strong tie-ins you'll end up with by using Microsoft tools. With a little engineering effort, you can build Delphi classes which can exposed their functionality in a way suitable to Microsoft's DNA technologies (for example using the Delphi code inside ISAPI servers and COM objects) and be able to port them to other Web Servers and other platforms, such a Linux, by providing a different wrapper to the same core code.

InternetExpress: XML, ECMAScript, and DHTML

Delphi 5 has further extended the traditional Delphi offering in the area of Internet development by providing a brand new technology based on the most recent open standards.

InternetExpress is based on two key components:

- The XMLBroker component can convert an existing dataset (using the MIDAS data stream format) into XML data. You can convert the result of a query, an entire table, and use any of the Delphi dataset components (those based on the BDE, the dataset components based on Microsoft's ADO, or the native InterBase components), to provide data to the XMLBroker and surface it on a web page.
- The MidasPageProducer component is a visual component designed for HTML forms based on the data provided by the XMLBroker. These pages, once made available in a browser, not only allow a user to see the database data, but have full support for editing, deleting, and inserting data in the database.

The user interface construction becomes similar, in its capabilities, to the common Windows user interfaces, although it is a native Web application, capable of running in multiple browsers with no need of any plug-in or custom extension. The reason for this openness lies in the fact that the Internet Express technology is based on open standards, these being:

- XML and an XML DOM
- ECMAScript (the official name of the JavaScript technology), is the only scripting language supported by most Web browsers, which can be used to customize the user interface, apply simple input and editing rules on the client side, make the user interface interact with the XML data.
- Dynamic HTML and CSS (Cascading Style Sheets) allow the development of a modern user interface within a browser, avoiding a tie-in of graphical elements with the HTML and the data rules. Quite the contrary, in fact, as the various

elements (HTML code, business rules, SQL server access) are kept well separate.

As mentioned earlier the InternetExpress architecture is based on Borland's MIDAS 3 technology. In its complete extensions, the architecture has 4 separate layers (see Figure 8), these being:

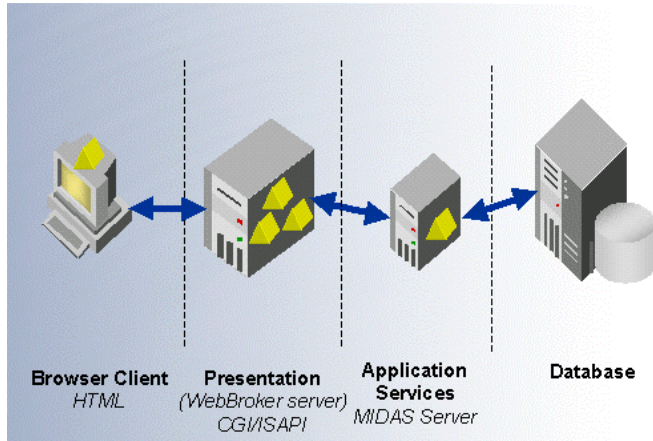


Figure 8: The different layers of a multi-tier Internet Express application.

- The SQL server (any of those supported by BDE, ADO, or native components), eventually running on a separate computer
- A MIDAS application server, which connects to the SQL server, applies specific business rules, and provides the data to the clients
- A WebBroker Web server extension, which ties to the Web server, and converts the data received by the MIDAS server into XML and provides a suitable HTML user interface
- The Web browser, which can be either Internet Explorer or Netscape (or other complying with the standards) and can run on any operating system

In the case of simpler projects, the picture can actually be simplified by merging the MIDAS server and the XML producer components into a single application. The advantage of the overall architecture is that you also gain the benefits of the MIDAS infrastructure, including a proper abstraction of the business logic, transport independence (it can run on top of

TCP/IP, HTTP, DCOM, MTS, and CORBA), and resource pooling (to share SQL server connections).

In practice, let me guide you through the steps you'll need to build a simple Internet Express front end for editing a simple database table. The starting point is again the creation of a new Web Server Application, using the wizard. In the web module you can add an action (see again Figure 7 and the description of the related example) and mark this action as the default action, by setting its Default property to True. Now you need the following components:

- An actual data set, such as a table or query component, connected with a database (using BDE, ADO, or another database access technology). The simplest solution is to use a BDE Table (from the Data Access tab of the component palette), setting its DatabaseName and TableName property. You can use the sample DBDEMOS alias for the DatabaseName property, and choose any of the available tables (from the drop-down menu of the property)
- A data set provider component (from the MIDAS tab of the component palette), hooked with the data set (Table1 in this case) using its DataSet property
- An XML broker component (from the InternetExpress tab of the component palette), hooked directly with the provider (in a "single" tier approach) using its ProviderName property
- A Midas Page Producer component (again from the InternetExpress tab of the component palette), connected with the default action of the web module by setting the Producer property of the action

At this point you can open the Midas Page Producer (by double clicking on it), and use its special editor to prepare the HTML form. For example, you can add a data form, add into it a data grid (connected with the XML broker) and a navigator (hooked to the grid using the XMLComponent property).

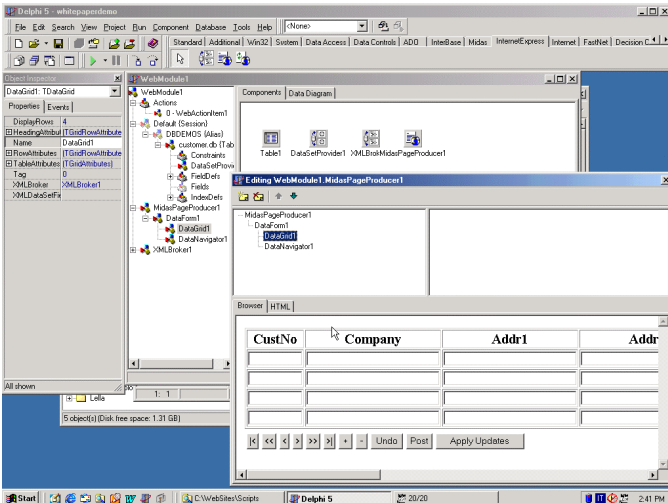


Figure 9: The editor MidasPageProducer component within the WebModule of our InternetExpress application.

After these steps you'll have a complex data module. To summarize its key options, I'm going to list the textual version of its DFM file, an internal Delphi file that reflects the result of visual development actions. You won't ever need to type this code, it is just a summary on the visual setting done in the Object Inspector, very useful for reference and documentation purposes. You can view your own WebModule this way by right clicking on it and then selecting "View as Text" command. You'll see more code than that listed here, as I've extracted from the textual definition of this DFM file only its key elements:

```

object WebModule1: TWebModule1
  Actions = <
    item
      Default = True
      Name = 'WebActionItem1'
      PathInfo = '/MidasPageProducer1'
      Producer = MidasPageProducer1
    end>
  object Table1: TTable
    Active = True
    DatabaseName = 'DBDEMOS'
    TableName = 'country.db'
  end
  object DataSetProvider1:
TDataSetProvider
    DataSet = Table1
  end
  object XMLBroker1: TXMLBroker
    ProviderName = 'DataSetProvider1'

```

```

WebDispatch.PathInfo =
'XMLBroker1'
end
object MidasPageProducer1:
TMidasPageProducer
  HTMLDoc.Strings = (...)
  IncludePathURL = '/include/'
  object DataForm1: TDataForm
    object DataGrid1: TDataGrid
      XMLBroker = XMLBroker1
    end
  object DataNavigator1:
TDataNavigator
  XMLComponent = DataGrid1
  end
end
end
end

```

Notice you have to remember to set the IncludePathUrl property of the Midas Page Producer to a URL referring to a directory where the browser can find the required JavaScript files. Otherwise the browser will show an error message, and no data.

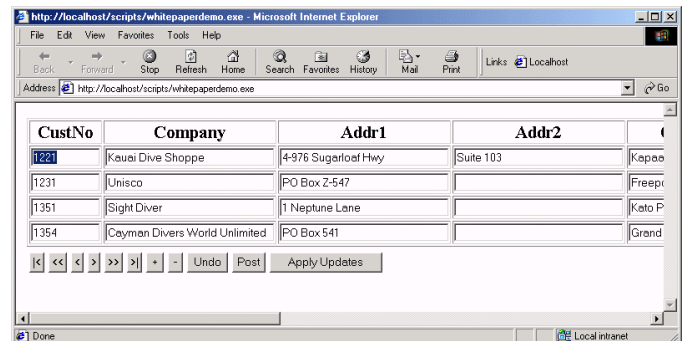


Figure 10: The InternetExpress application we have just built inside a browser (in this case Microsoft Internet Explorer)

This is all! At this point you'll have a complete application, allowing a user not only to see the data inside a browser (any browser!) but also to edit the data and send it back to the server, to be posted to the database. See Figure 9 for an example. And, again, with no coding required we've only chosen the basic options, but with some study (one of the few references, although limited on this regard, is my own book "Mastering Delphi 5") and effort you can build sophisticated program, with a modern browser-based UI.

In short:

- With Delphi InternetExpress architecture you can extend MIDAS multi-tier applications with a browser-based user interface. Relying on HTTP, XML, and ECMAScript, the architecture is not tied to any browser or operating system, which differs from other equally powerful solutions.
- Developing a simple front end for you business data with InternetExpress is really a very fast and completely visual. Even without knowing all the core technologies of the architecture, you can still easily write fully functional and professional-looking web sites.

Third-Party Web Server Extensions

Besides using Borland's own InternetExpress technology, you can use Delphi with some third party components and tools, which support the development of HTML-based server side applications with different approaches. There are many tools in this category, so I'm just mentioning the most popular ones, and not trying to offer a complete picture.

- HREF (<http://www.href.com>) offers the popular WebHub framework, an advanced technology for manipulating HTML snippets and create Web content based on database data. WebHub is capable of handling user sessions, separate the code development from the server side technology used, and helps developers to move to a proper Web-centric approach, instead of adapting existing Windows user interfaces to the Web.
- Nevrona Design offers ND-IntraWeb (<http://www.nevrona.com/intraweb>) which follows an opposite approach: Using a series of custom user interface components it allows you to build a user interface which can work equally well inside a Windows program or a Web browser.
- Marotz Delphi Group offers ASP Express (<http://www.asp-express.com>) offers a set of components designed to encapsulate and simplify the development of Windows DNA applications. It uses ASP, MSXML, COM,

and other technologies, making it easier to combine everything with Delphi code.

In short:

- Custom server side solutions provide ready-to-use complex frameworks for the development of your applications with Delphi. Some of these technologies have been successfully used for the development of large and complex web sites.
- Similarly to the use of components wrapping server side protocols, you can have full control of the entire software on your web server, with no risk of others people bugs creeping into your system.

Delphi in Action on the Internet

Delphi's capabilities in the area of Internet and Web development can be discussed by their technical merit, as I've done in this paper, but can also be evaluated based on their success. Delphi is used for the development of many Internet-products, ranging from simple shareware utilities to huge e-commerce Web sites. Although the press is all about other languages and environments, Delphi has found its inroads in the Internet era, and is in widespread use right now.

Borland provides a long list of success stories (see <http://www.borland.com/about/cases> and <http://www.borland.com/delphi/cases>), but there are a few worth highlighting. Among the web sites powered (at least for the database oriented portions) by Delphi there are:

- Autobytel.com, a US on-line car dealer
- The National Trust (<http://www.nationaltrust.org.uk>), a UK organisation for preserving historical buildings
- iVillage.com, the women's network web site
- Dulux Trade Paints (<http://www.dulux.com>), a web site for choosing paint colors
- CalJobs (<http://www.caljobs.ca.gov>), the State of California Internet system for linking employer job listings and job seeker resumes.
- Travel.World.Net, a complete travel services management system by Australian World.net.

Speaking of Internet utility programs, almost half of the Internet-related shareware programs on the market are built with Delphi. Some worth mentioning, taken from different categories, are:

- The powerful email manager The Bat! (<http://www.ritlabs.com>) and the freeware mail client Mail Warrior (<http://pages.infinet.net/kaufman>), top rated in many software web sites
- The acclaimed HTML editor HomeSite (<http://www.allaire.com/homesite>)
- The MERAK Mail Server (<http://www.icewarp.com>)
- Two of the most popular IRC clients, Pirch (<http://www.pirchat.com>) and Virc (<http://www.megalith.co.uk/virc/>)

There are also newsgroup readers, FTP front ends, XML editors, chat programs, and applications for the client and the server side almost any possible Internet protocol.

Another area of success in Delphi development is the creation of high-quality ASP components. Among the most popular ASP add-ons mentioned on Microsoft's web site

(<http://msdn.microsoft.com/workshop/server/components/catalog.asp>) there are quite a few written in Delphi, including the ASP components written by Dimac (<http://www.dimac.net>) and including the popular JMail component, described as "the leading SMTP-component for ASP-coders".

You can find a rather complete (albeit unofficial) list of Internet applications and Web sites powered by Delphi on the "Built with Delphi" area of the Baltic Solution web site

(<http://www.balticsolutions.com/bwd>).

Conclusion: The Delphi Advantage

After this long detailed paper it is not easy to summarize in only a few words why you should use Delphi as a core Web development tool, within your organization. I can certainly say that Delphi delivers fast-performance applications built with a rapid development environment for Windows and the Web. It has optimal Client/Server support and the ability to write good-quality object-oriented code, both for the building of a complex

application structure and also to delve deep into low-level programming tasks.

A great tool for the entire Internet needs of any organization.

And a tool you can use today on the Windows platform and get ready to extend to the Linux operating system with a visual and high-performance development environment, the ideal solution for all those who like programming "The Delphi Way".