Under Construction: Delphi 5 InternetExpress

by Bob Swart

Delphi 5's InternetExpress combines the WebBroker technology with MIDAS, producing HTML and XML as the final results, for ultra-thin clients. InternetExpress actually results in a 4-tier application: database server (SQL DBMS), middleware data server (MIDAS), web server (WebBroker) and finally the HTML/XML inside a web browser. Note that the fourth tier can actually be deployed on any machine, even on Linux or an iMac (as long as the browser supports XML, or at least JavaScript).

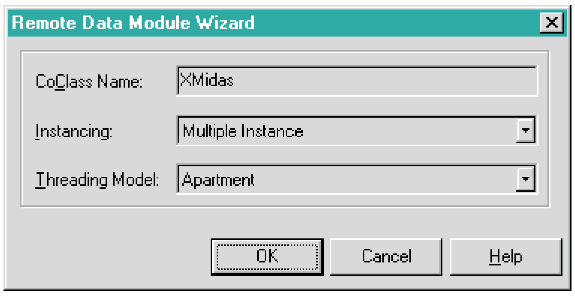
MIDAS Server And Remote Data Module Designer

Before we start, a little word on MIDAS 3. This version of MIDAS, that ships with Delphi 5, has undergone a few changes. Probably most important is the fact that MIDAS is now stateless, meaning the middleware data servers no longer keep track of the clients and their state: we now have to keep track of our own clients' state. This may seem cumbersome, but it also enables us to use MIDAS with other stateless solutions, such as CGI and ISAPI web server applications.

Master-Detail Example

Since we are building a 4-tier application, we first need to build a middleware data server. In our example, I'd like to focus on the customer and orders master-detail relationship. This set of tables is available to anyone with Delphi 5.

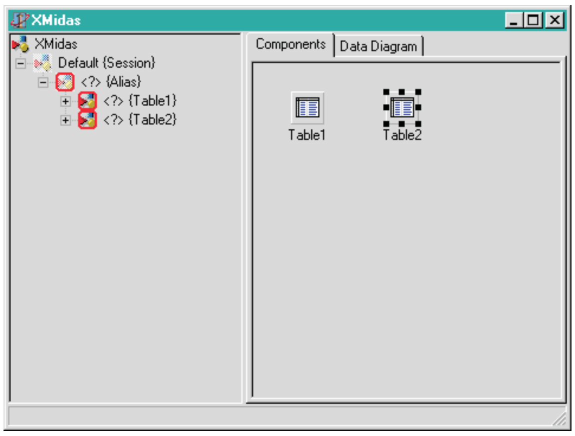
First, start Delphi 5, which opens up a new default project. Save this project in a sensible place (for example in the C:\TDM50\Server directory) as IxServer.dpr (and main.pas for the form), and make sure you put some additional components on the default main form to identify the MIDAS server later. Now, do `File New` and add a new `Remote Data Module` from the `Multitier` tab in the object repository, and call it `XMidas` (leave the default choices of `Multiple Instance` and `Apartment`, as usual), as in Figure 1.



CoClass Name: XMidas
Instancing: Multiple Instance
Threading Model: Apartment

➤ *Figure 1*

This will generate a new remote data module for us, and enters us in the new visual Data Module Designer. Here, we can drop two tables, and immediately see some visual (design-time) cues that we need to perform some additional work on before we can actually use them (Figure 2).

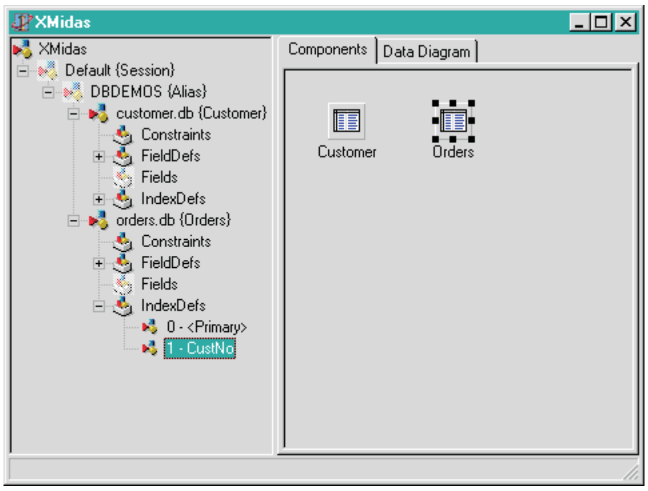


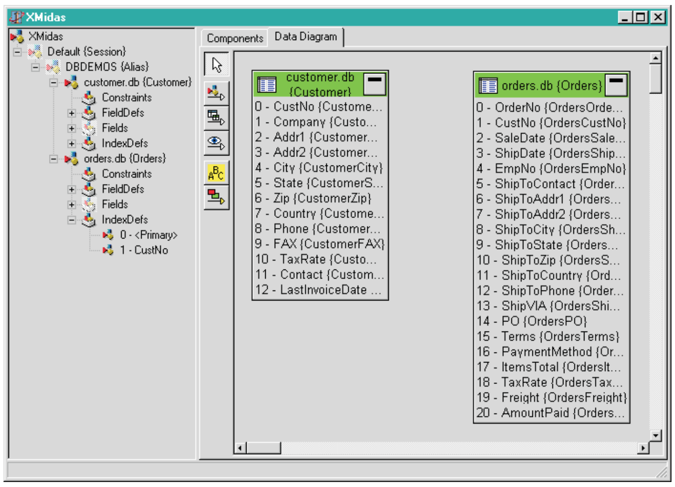
➤ *Figure 2*

The red squares around the alias and two tables in the left frame (you noticed them too, didn't you?) are a useful indication that we need to specify some additional information, like the `AliasName` and the `TableNames`. To get rid of these 'error' indicators, we simply need to solve the 'problems', or complete the specifications. So, click on `Table1` and specify `DBDEMOS` as the `DatabaseName`. Now, we see `Table1` which belongs to the `DBDEMOS` alias, while Table2 still hangs under the red alias. We can now either specify the `DatabaseName` for Table2 to be `DBDEMOS` just like Table1, or simply drag and drop Table2 onto the `DBDEMOS` alias. Now, at least the alias is known for both tables, but we still need to specify the two `TableNames` themselves before the final two red indicators around the tables will disappear.

In order to do just that, select `customer.db` as the `TableName` for `Table1`, and `orders.db` as the `TableName` for `Table2` (and rename the tables as `Customer` and `Orders`, so we know what data we are working on). We can click on the + signs next to `Customer` and `Orders` in the left frame to drill down to detail information on the tables, such as `Constraints`, `FieldDefs`, `Fields` and `IndexDefs`. Note that actual information for the `FieldDefs` won't be available until we open the table (set the `Active` property to `True` for both tables). The `IndexDefs` information becomes available after we select the `Update Table Definition` option with a click with the right mouse button on the table components (this also gives us the `FieldDefs` information, in case we didn't open the table, so you can shortcut this by just executing the `Update Table Definition` if you want). See Figure 3.

To get a list of the actual fields, we can either start the fields editor the old way, or just right-click on the `Fields` and select the `Add all fields` option directly (again, on





customer.db {Customer}
0 - CustNo {Custome...
1 - Company {Custo...
2 - Addr1 {Customer...
3 - Addr2 {Customer...
4 - City {CustomerCity}
5 - State {CustomerS...
6 - Zip {CustomerZip}
7 - Country {Custome...
8 - Phone {Customer...
9 - FAX {CustomerFAX}
10 - TaxRate {Custo...
11 - Contact {Custom...
12 - LastInvoiceDate ...

orders.db {Orders}
0 - OrderNo {OrdersOrde...
1 - CustNo {OrdersCustNo}
2 - SaleDate {OrdersSale...
3 - ShipDate {OrdersShip...
4 - EmpNo {OrdersEmpNo}
5 - ShipToContact {Order...
6 - ShipToAddr1 {Orders...
7 - ShipToAddr2 {Orders...
8 - ShipToCity {OrdersSh...
9 - ShipToState {Orders...
10 - ShipToZip {OrdersS...
11 - ShipToCountry {Ord...
12 - ShipToPhone {Order...
13 - ShipVIA {OrdersShi...
14 - PO {OrdersPO}
15 - Terms {OrdersTerms}
16 - PaymentMethod {Or...
17 - ItemsTotal {OrdersIt...
18 - TaxRate {OrdersTax...
19 - Freight {OrdersFreight}
20 - AmountPaid {Orders...

➤ *Above: Figure 3*

➤ *Right: Figure 4*

both tables). As soon as the actual fields are available, we can remove the fields we don't want. This is just the old behaviour of Delphi, except that we can now see a list of fields right there in the left frame of the visual data module designer. Quite helpful again, if I may say so.

Master-Detail

Now the really interesting part starts: it's time to define the relationship between the `Customer` and `Orders` tables. The old-fashioned way would be to add a datasource component, connect it to the `Customer` table, and use this as the `MasterSource` for the `Orders` table. Then, we'd need to click on the `MasterFields` property inside the Object Inspector to define the `CustNo` master-detail relationship between the `Customer` and `Orders` tables.

Fortunately, we can do it visually this time, using the `Data Diagram` tab of the visual Data Module Designer. Just click on this tab to see the (still empty) `Data Diagram`. Now, drag the `Customer` and `Orders` tables from the left frame over to the `Data Diagram`. This will immediately show a lot of visual detail of these two tables (Figure 4).

Now, click on the master-detail button (third from the top), and then take the master table (`Customers`) and drag it to the detail table (`Orders`). Note that the Delphi 5 on-line help *incorrectly* states that we need to drag the detail to the master, which works the other way around. This was reported as a problem in the Delphi 5 Release Candidate 2 that I used for this article, but I'm not sure it's fixed in the final version, so beware! You won't notice something's wrong until you actually want to use the exported master-detail relation from a remote client, only to discover that there's no detail available.

Fortunately, you can check the final result, since the line between the master and the detail uses a big square to indicate the master and a smaller one for the detail. After we drag the master to the detail table, we get the Field Link Designer dialog, where we specify which field (or fields) make up the master-detail relationship. In this case it's `CustNo`, see Figure 5.

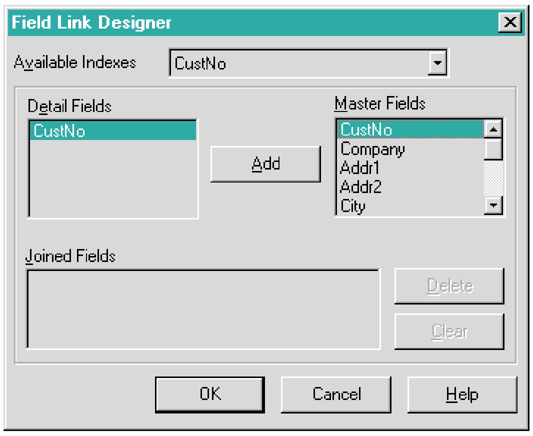
After we click on the `Add` and `OK` buttons, the master-detail relationship is defined. Note the `DataSource1` component, which was created for us to act as `MasterSource` for the `Orders` table, just as we would have done in the old-fashioned way. See that the line ends in two squares: a big one (the master) and a little one (the slave), and has a label identifying the field connecting the two tables together. If we want to remove the master-detail relationship, we should go to the `Components` tab and remove the `DataSource` component (which is the connecting component for this relationship). Just removing the relation itself on the `Data Diagram` will disconnect the `DataSource`, but not remove it, since we may be using it for other purposes, leaving you with an unused `DataSource` component. And they can quickly add up, if you try adding and removing relations just to see what will happen. See Figure 6.

Apart from master-detail relationships, we can also use the Data Diagram to visually define lookup fields, add our own relationships, custom comments, etc. And we can print the Data Diagram or export it to a WMF or EMF file to include in our project documentation. This is really helpful. And of course it also works with existing, pre-Delphi 5, data modules. Just open your project, drag and drop your tables/queries and whatever over to the Data Diagram tab, and you'll see the fields and the relations, etc. Wonderful.

Exporting DataSetProvider

Now, click on the `Components` tab of the Data Module Designer. Move over to the `MIDAS` tab on the Component Palette, and drop a `TDataSetProvider` component on

➤ *Figure 5*

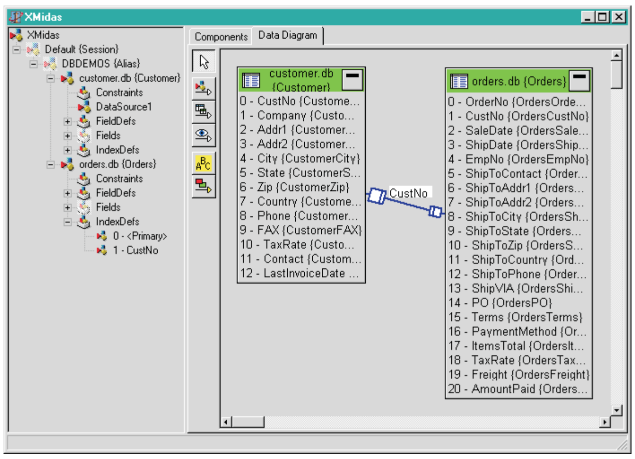


Available Indexes: CustNo

Detail Fields
CustNo

Master Fields
CustNo
Company
Addr1
Addr2
City

Joined Fields



```
XMidas
XMidas
  Default {Session}
    DBDEMOS {Alias}
      customer.db {Customer}
        Constraints
        DataSource1
        FieldDefs
        Fields
        IndexDefs
      orders.db {Orders}
        Constraints
        FieldDefs
        Fields
        IndexDefs
          0 - <Primary>
          1 - CustNo
```

```
customer.db {Customer}
0 - CustNo {Custome...
1 - Company {Custo...
2 - Addr1 {Customer...
3 - Addr2 {Customer...
4 - City {CustomerCity}
5 - State {CustomerS...
6 - Zip {CustomerZip}
7 - Country {Custome...
8 - Phone {Customer...
9 - FAX {CustomerFAX}
10 - TaxRate {Custo...
11 - Contact {Custom...
12 - LastInvoiceDate ...

orders.db {Orders}
0 - OrderNo {OrdersOrde...
1 - CustNo {OrdersCustNo}
2 - SaleDate {OrdersSale...
3 - ShipDate {OrdersShip...
4 - EmpNo {OrdersEmpNo}
5 - ShipToContact {Order...
6 - ShipToAddr1 {Orders...
7 - ShipToAddr2 {Orders...
8 - ShipToCity {OrdersSh...
9 - ShipToState {Orders...
10 - ShipToZip {OrdersS...
11 - ShipToCountry {Ord...
12 - ShipToPhone {Order...
13 - ShipVIA {OrdersShi...
14 - PO {OrdersPO}
15 - Terms {OrdersTerms}
16 - PaymentMethod {Or...
17 - ItemsTotal {OrdersIt...
18 - TaxRate {OrdersTax...
19 - Freight {OrdersFreight}
20 - AmountPaid {Orders...
```

➤ *Figure 6*

the data module (note that unlike Delphi 4, we no longer have a `TProvider` component, it's all using `TDataSetProvider` now). Rename the `DataSetProvider` to `CustomerOrdersProvider`, and connect it to the `Customer` table. The `Orders` table will be attached to the `Customer` table as a so-called nested dataset field (as we'll see later on).

Again in the 'old days' (using Delphi 4), we had to right-click on the `CustomerOrdersProvider` to export it from our remote data module. Using Delphi 5, however, this step is no longer needed: the provider is already exported by default. We can change this by setting the value of the `Exported` property in the Object Inspector (so no right-clicking any more). This also implies that we should be able to dynamically define whether or not a provider is being exported to the outside world. This is something we'll test next month: it should prove quite useful at times; for example, depending on certain security details, you don't want to 'show' or 'make available' certain providers to everyone.

The only thing left to do with the InternetExpress MIDAS server is to save it, compile it and run it once on our machine, so it is registered and available for use by the InternetExpress clients (Figure 7).

MIDAS Client

After you've run the IxServer project, close the executable and open the Project Manager to start a new project: our InternetExpress MIDAS client. This is a WebBroker application. In the Project Manager, right-click and select `Add New Project`. Then, select a `Web Server Application` in the Object Repository, and click on `OK`. This will bring up the New Web Server Application wizard. Just leave the default choice of ISAPI/NSAPI and click on `OK` to create our new web module application. Save this project as IxClient.dpr (and use WebMod.pas for the web module itself).

Remember that a Remote Data Module is the most flexible of all remote data modules when it comes to communication protocols. Where a CORBA Data Module would require an ORB, and an MTS Data Module would require MTS, a standard Remote Data Module can communicate using DCOM, sockets (TCP/IP) and now with Delphi 5 even using stateless HTTP. As you may remember, we can always export a standard remote data module using CORBA, to be able to use all these protocols!

For our example this time, we'll focus on stateless HTTP communication (which can go through a firewall or a proxy server). So instead of the `DCOMConnection` component we'd usually select for a MIDAS client, we'll now use the new `WebConnection` component from the `MIDAS` tab of Delphi 5. Under this component, drop an `XMLBroker` component (from the `InternetExpress` tab) and finally a `MidasPageProducer` component (also from the `InternetExpress` tab). As we've seen earlier, the Data Module Designer indicates a number of our components aren't ready yet, with red squares. So let's fix these problems (Figure 8).

First, click on the `WebConnection` component. This has to connect to the IxServer we just created. The `WebConnection` component can make this contact using the HTTP protocol. However, to use a `WebConnection`, we must make sure that WININET.DLL is installed on the client system (which is available if you have Internet Explorer 3 or higher installed). The server must also have IIS version 4 or higher or Netscape Enterprise version 3.6 or higher, and finally we must install HTTPSRVR.DLL (found in the DELPHI5\BIN directory) in a scripts directory on the web server that the `WebConnection` component uses to connect to. HTTPSRVR.DLL is responsible for launching the MIDAS middleware server, and will marshal all calls from the client to the application server interface.

As a direct consequence, the `URL` property of the `WebConnection` component must point to:

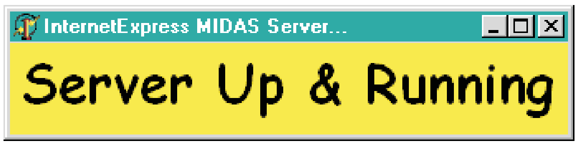
```
http://localhost/cgi-bin/
  httpsrvr.dll
```

which points to the scripts directory on my local machine. I could also have used:

```
http://www.drbob42.com/cgi-bin/
  httpsrvr.dll
```

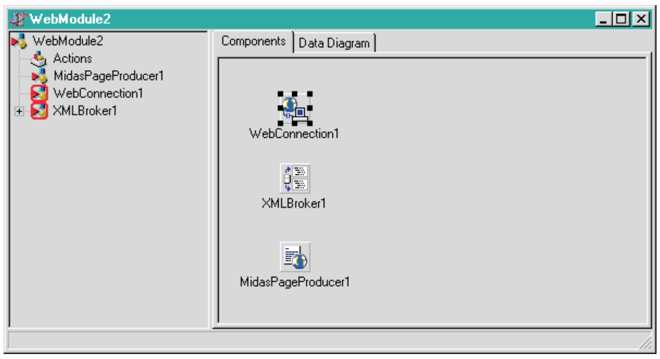
of course. Next, we can click on the `ServerName` property, open up the list of available MIDAS servers and select the `IxServer.XMidas` middleware server. This fixes the red square around the `WebConnection` component. We can make sure that the connection actually works by double-clicking on the `Connected` property. If the value turns into `True`, we're OK. Note that we don't actually see the MIDAS server running. That's because HTTPSRVR (started by the web server) is actually activated by another user (the default internet user), and as a result we don't see any visual representation of the middleware server at this time (as we were use to when using a DCOM or CORBA middleware server). To make sure the server is actually running, we can always take a look

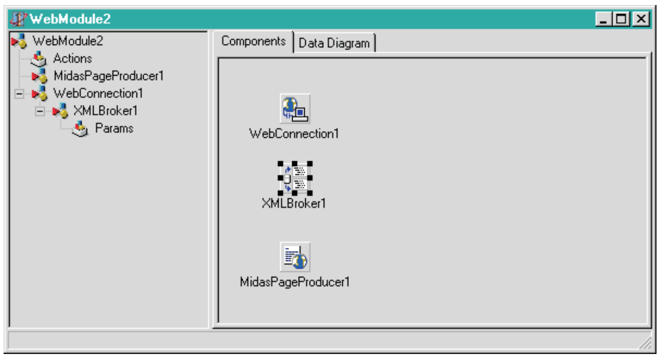
➤ *Figure 7*



InternetExpress MIDAS Server...

Server Up & Running





➤ *Above: Figure 8*

➤ *Right: Figure 9*

at the Task Manager, of course. Make sure to disconnect the server before we continue.

Now click on the XMLBroker component and point its RemoteServer property to the WebConnection component. Next, it's time to pick the DataSetProvider we exported from the remote server. To do this, click on the little arrow on the right of the ProviderName property in the Object Inspector. This will bring down a list of available providers from our remote server.

Of course, the only reliable way for Delphi to show this list of providers is simply to ask for a list from the running middleware server. So, when we click on the arrow, the first result will be that the IxServer middleware server will be started (but again, we don't see a visual cue), followed by the list of available providers that the Object Inspector received from this MIDAS server. In this case, we only see the CustomerOrders-Provider, since we only exported one provider from the remote data module, the master Customer table with the detail Orders table 'embedded' as nested dataset field. See Figure 9.

InternetExpress

It's time to move on to the InternetExpress part of this article with the MidasPageProducer. First, notice that the HTMLDoc property of this component already contains a default template inside. Don't mess with that (yet). Instead, right-click on the MidasPageProducer and select the Web Page Editor. This brings us into the heart of the InternetExpress visual designer. The upper left frame lists the components currently in use, the upper right frame lists the child components of the selected parent component (in the left frame).

The lower frame has two views: Browser and HTML. The former actually uses the TWebBrowser component (wrapped around the IE control) to give a visual representation of the HTML source code which is shown in the HTML tab, of course.

We now can design our resulting web page, by right-clicking on components in the left frame and adding new components as we need them. Note that on every level we'll only be able to create new components that are actually relevant. For selected components in the upper left or right frames, we can always use the Object Inspector to modify their property values. We'll get immediate visual feedback in the Browser page, which is quite exciting to watch!

For our master-detail example, we probably want to have a list of master fields, followed by a grid showing all the detail fields. So, let's see how to do that.

The first time we select New Component, we can add a DataForm, QueryForm or LayoutGroup. Let's select a DataForm. Now, click on the DataForm, and again create new components. This time, we can choose between a DataGrid, a DataNavigator, a FieldGroup or a LayoutGroup.

For our example, add a DataNavigator, a FieldGroup, another DataNavigator (for the detail) and a DataGrid. We can even go some levels deeper, by specifying the fields we want to show inside the FieldGroup or DataGrid, or by specifying the buttons we'd like to see in the DataNavigator components.

Also, like the visual Data Module Designer, we now get warnings that show us that certain properties haven't been assigned, yet (so all this isn't complete, yet). The warnings will remain there until we fix them, of course. See Figure 10.

First, let's connect the two DataNavigators to a XMLComponent. The first one to the FieldGroup, the second one to the DataGrid. Second, connect the FieldGroup to the XMLBroker component. Once we assign the XMLBroker property of the FieldGroup, the IxServer middleware server is launched again. This enables the InternetExpress Web Page Editor to provide us with detailed information and actual live data (at design-time).

As usual, we implicitly get all fields (from the CustomerOrders-Provider) from the XMLBroker. To get them explicitly, and be able to remove certain fields, we have to right-click on the FieldGroup component, and select Add All Fields. We can now remove all fields we don't want to see (such as Addr2, and so on) by selecting them in the right frame and deleting them from there. There's one special field, called FieldStatus, which holds the status information for the current record (M for modified, for example, as we will see in detail next month, when we also focus on update and reconcile errors).

The final warning about the DataGrid1.XMLBroker can be resolved by assigning the XML-Broker and XMLDataSetField properties of the DataGrid1 component. XMLBroker must be set to XMLBroker 1 (there's no other choice) after which we can set the XMLData-SetField property to Orders. This will ensure that the `DataGrid` displays the detail `Orders` table. By this time, the final design-time warning has disappeared as well.

Like before, we can right-click on the `DataGrid` to `Add All Fields` and then remove the fields we don't want to see in the grid, like `ShipToAddr2`, etc. See Figure 11.

Action!

We're almost ready. We only need to connect the `MidasPageProducer` to a `WebActionItem`. We can create the latter the usual way: right-click on the Web Module to see the Actions Editor, and right-click inside the Actions Editor to add a new action. Set the default to `True` (and don't bother with the `PathInfo`). In order to connect the `MidasPageProducer` with this action, we used to have to write code for the `OnAction` event handler, but with Delphi 5 we can simply assign the `MidasPageProducer` to the `Producer` property of the `WebActionItem`. Note that this will automatically enter some information in the `PathInfo` field as well, which we would like to clear again, but are unable to: a side effect of this way of action-producer coupling.

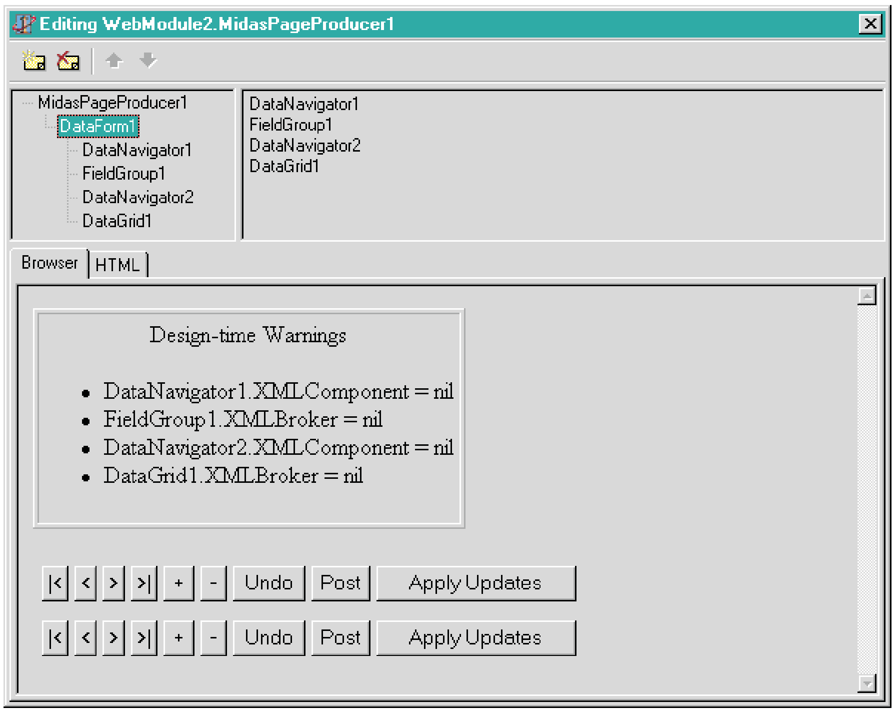
Deployment

So, we're almost ready for the final test. Save and compile our final WebBroker client, and put it in the cgi-bin (scripts) directory of your web server (IIS in this case). We should also make sure to deploy the JavaScript libraries from the DELPHI5\SOURCE\WebMidas folder, and specify this location in the `IncludePathURL` of the `MidasPageProducer` (http://localhost/cgi-bin in my case). Finally, we should deploy MIDAS.DLL (the renamed DBCLIENT.DLL).

A big advantage of using the `WebConnection` component is that it can be used to connect through a firewall and use SSL security. Note that this applies to the WebBroker application connecting to the MIDAS middleware application server. The WebBroker can be outside of the firewall (where everyone can 'touch' it), while the MIDAS middleware application can be safely inside the firewall (maybe even on another machine).

When a `WebConnection` connects to the MIDAS middleware application server, it supplies the values of the `UserName` and `Password` properties so that it can log on to the host or proxy. The value of `UserName` and `Password` can be left empty if neither the host nor proxy requires authentication.

If we make any changes and post them, the status will be `M` (for modified). If we check the internal source code for the generated page, we see a lot of XML inside for the specified records. In fact, all master-detail records are presented using raw XML in the web page, which uses the included JavaScript code to parse and


Editing WebModule2.MidasPageProducer1

```
MidasPageProducer1
  DataForm1
    DataNavigator1
    FieldGroup1
    DataNavigator2
    DataGrid1
```

```
DataNavigator1
FieldGroup1
DataNavigator2
DataGrid1
```

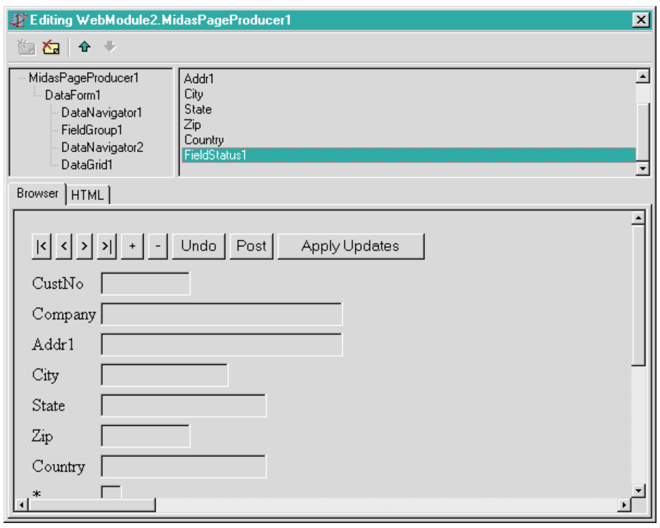
Design-time Warnings

- DataNavigator1.XMLComponent = nil
- FieldGroup1.XMLBroker = nil
- DataNavigator2.XMLComponent = nil
- DataGrid1.XMLBroker = nil

Browser
HTML
|< < > >| + - Undo Post Apply Updates
|< < > >| + - Undo Post Apply Updates

➤ *Figure 10*

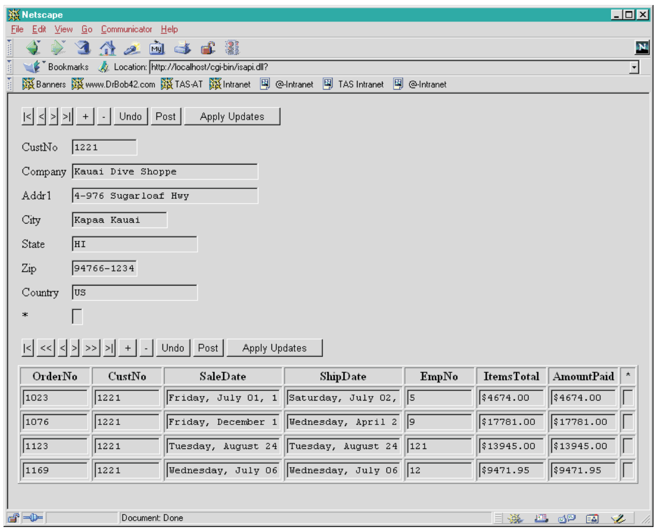
➤ *Below: Figure 11*


Editing WebModule2.MidasPageProducer1

```
MidasPageProducer1
  DataForm1
    DataNavigator1
    FieldGroup1
    DataNavigator2
    DataGrid1
```

```
Addr1
City
State
Zip
Country
FieldStatus1
```

➤ *Right: Figure 12*


Netscape

File Edit View Go Communicator Help
Bookmarks
Banners www.DrBob42.com TAS-AT Intranet @-Intranet TAS Intranet @-Intranet
|< < > >| + - Undo Post Apply Updates

Location: http://localhost/cgi-bin/isapi.dll?

CustNo	1221
Company	Kauai Dive Shoppe
Addr1	4-976 Sugarloaf Hwy
City	Kapaa Kauai
State	HI
Zip	94766-1234
Country	US
*	

|< << < > >> >| + - Undo Post Apply Updates

OrderNo	CustNo	SaleDate	ShipDate	EmpNo	ItemsTotal	AmountPaid	*
1023	1221	Friday, July 01, 1	Saturday, July 02,	5	$4674.00	$4674.00	
1076	1221	Friday, December 1	Wednesday, April 2	9	$17781.00	$17781.00	
1123	1221	Tuesday, August 24	Tuesday, August 24	121	$13945.00	$13945.00	
1169	1221	Wednesday, July 06	Wednesday, July 06	12	$9471.95	$9471.95	

Document: Done

display the data embedded in XML inside the controls that we see in the Figure 12. Quite ingenious, as the resulting web page doesn't even need to refresh (or flicker) when we move from one record to another). Truly a powerful thin client, and in my view one of the most powerful new features of Delphi 5!

Next Time

InternetExpress is a hybrid mix between the WebBroker and MIDAS Technologies. Using InternetExpress, we can generate N-tier but yet true thin-client web applications that can be deployed in any XML/JavaScript supporting browser.

We've left a number of things uncovered. Like how to handle update errors. Or how to use the InternetExpress components (`XMLBroker` and `MidasPageProducer`) without actually using MIDAS. This and more will be covered next time, including a way to limit the number of XML-packages records with the `MaxRecords` property of the `XMLBroker` component, something that won't work in the current example, *so stay tuned...*

Bob Swart (aka Dr.Bob, visit www.drbob42.com) is an IT Consultant for TAS Advanced Technologies and a freelance technical author.

***Editor's Note**. Together with Marco Cantù, Bob Swart recently received the 1999 Spirit of Delphi award from Inprise. Congratulations to them both!*