Under Construction: Delphi 5 InternetExpress, 3

by Bob Swart

Delphi 5 InternetExpress combines the WebBroker and MIDAS technologies, producing HTML and XML as the end result for ultra-thin clients. This is the third, and (for now) final, article in this series, explaining how to make best use of InternetExpress.

The Story So Far

Two months ago, we saw how to build a master-detail relationship on a MIDAS application server, and provide it (using a `WebConnection` component) to a client application. The client consisted of an `XMLBroker` and `MidasPageProducer`, extending the existing WebBroker Technology to produce a web server application. XML was used in two ways: both as a data format for the packets that were sent from the `DataSetProvider` to the `XMLBroker` (and back), and as a data format for the actual data embedded inside the (D)HTML web pages that we could see inside the web browser.

Last month, we saw how to limit the amount of data being sent from the web server to the browser, and how to handle update errors (also called reconcile errors). And I also showed you how you could use the `InternetExpress` component without actually needing a MIDAS licence... but, ahem, check the sidebar on page 21 for an update on that issue.

This Time

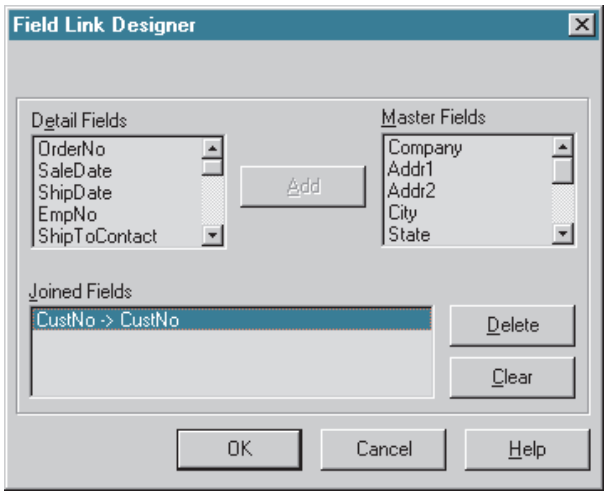
This time (in the third and, for now, last instalment of this topic), we'll see how we can extend InternetExpress, by exploring some custom add-on components that plug right into the Web Editor. We'll also explore ways to enhance the layout of the resulting web pages and see techniques to debug InternetExpress applications: not an easy task once you realise the tricks that your web server can play with you.

But before we start, I'd like to return to the place where we left off last time: the topic of using InternetExpress in a single-tier application.

Non-Remote InternetExpress

Some of you may have noticed it already, but last month I presented a solution for a non-remote `InternetExpress`, where we used two tables, two `DataSetProviders` and two `XMLBrokers` to feed a `MidasPageProducer` inside a WebBroker application. Unfortunately, this does not give us a working master-detail relation: the master and detail data is not connected. You can see what I mean if you execute the CGI51 project from last time in your browser (if you have no web server on your machine, read on for a quick-and-dirty solution for that part). Inside your browser you will see that the master table (`customer`) is not linked to the detail table (`orders`). You can navigate through the master table, but no matter which master record is selected, the detail table will show all the orders.

➤ *Figure 1: Field Link Designer.*



The reason for this is, of course, the fact that we used two standalone `XMLBroker` components, which are in no way connected to each other (they don't even know the other one exists), so the resulting XML data packets are in no way aware of each other's existence. In a multi-tier InternetExpress application, we could simply use one `XMLBroker` and get our hands on the detail table (orders) through the `NestedTable` `DataSet` field. The funny thing, as you'll see in a moment, is that this field was *not* available from the `XMLBroker` component when building a non-remote InternetExpress application. But let's start from the beginning.

Start a new WebBroker application (`File | New - Web Server Application`), select an ISAPI/NSAPI Dynamic Link Library (we'll use this example to debug later in this article) and save as project name ISAPI52.DPR (with WebMod.pas for the Web Module itself). Drop two table components on (BDE, ADO or whatever). I'll use ADO here, as I personally think that ADO, as the next generation ODBC, will be a more and more commonly used database connectivity technology. So, apart from the two `TADOTable` components, we need a `TADOConnection` component as well. After you've connected the `ADOConnection` (see last month), make sure the two tables point to the `customer` and `orders` `tables`, and rename them to `ADOTableCustomer` and `ADOTableOrders`. Next, we need a `DataSource` component connected to the `ADOTableCustomer`. This `DataSource` is used to create the master-detail

relation between the ADOTableCustomer and ADOTableOrders (because the Visual DataModule Designer has a bug that prevents it to work with ADO tables, remember?). Set the MasterSource property of ADOTableOrders to the DataSource, and click on the ellipsis next to the MasterFields property to get the Field Link Designer dialog for an ADO master-detail relationship.

Note that ADOTables provide a bit less information to the Delphi IDE than regular BDE tables, so we don't see index names, for example. That should not be a problem, however. Just select the CustNo field on both sides and add the master-detail relationship (see Figure 1).

Now, go to the MIDAS tab, and drop one DataSetProvider on the Web Module. Set its DataSet property to the ADOTableCustomer. Last time, we used another DataSetProvider to connect to the ADOTableOrders, but this time we pass the records of ADOTableOrders as a nested table (in a DataSetField) instead.

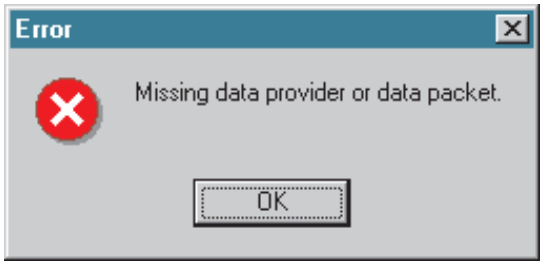
As the next step, go to the InternetExpress tab, and drop the XMLBroker and MidasPageProducer components. Set the ProviderName of the XMLBroker to the DataSetPageProducer component. Note that we don't want or need to specify the RemoteServer property of the XMLBroker component; instead, we need to wire the XML packets from the DataSetProvider directly to the XMLBroker.

Custom InternetExpress Components

Before we continue, let's make sure the 'InternetExpress sample components' design-time package has been loaded and activated for our current project. In case you didn't load it last time, it can be found at:

```
C:\Program Files\Borland\
  Delphi5\demos\MIDAS\
  InternetExpress\INetXCustom\
  DCLINetXCustom.dpk
```

(where the runtime package INetXCustom.dpk can also be found). After compilation and installation, the dclinetxcustom.bpl file can be found in the Delphi5\bin directory next to the other design-time packages. Apart from the ReconcilePageProducer (which we saw and used last time), it also contains numerous components that are not immediately visible, but available in the Web Page editor of the MidasPageProducer...



➤ *Figure 2: Missing data provider or data packet.*

MidasPageProducer

So, right click on the MidasPageProducer to start the Web Page editor, in order to visually build the master-detail relationship. Like we've done before, we can start with a DataForm. However, the dclinetxcustom.dpk package contains a number of additional components that we can use here, such as a TitleDataForm. The TitleDataForm component has three new properties (compared to the DataForm), namely Caption, CaptionAttributes (Custom, Style and StyleRule) and CaptionPosition. We can use this to specify a caption that can appear on top, bottom, left or right side of the other content of the TitleDataForm. The Caption itself can contain HTML codes, such as <H1>The Delphi Magazine #52</H1>.

If we select the TitleDataForm, we can add new components again, such as a FieldGroup (for the master record) and a DataGrid (for the detail records). We also need a DataNavigator. In this case, I'd like to take two ImgDataNavigator components (also introduced by the dclinetxcustom package). ImgDataNavigators have the same functionality as regular DataNavigator components, but instead of buttons with > and < caption text on them, the buttons consist of images. A set of sample images (applyupdates.gif, delete.gif, first.gif, insert.gif, last.gif, next.gif, nextpage.gif, post.gif, prior.gif, priorpage.gif and undo.gif) is provided in the same directory where you can find the source code of the dclinetxcustom.dpk package. These should be made available on your web server, and each ImgDataNavigator must be able to locate them by specifying the location of the images in the ImagePathURL property of the ImgDataNavigator (like http://192.168.91.201/images in my case). Sometimes, the images don't show up spontaneously. In that case, I need to right click on the ImgDataNavigator, and explicitly add the ImgButtons I want to see. Close the Web Editor dialog, re-open it, and then they'll show up. A bit funny, but it's the first time of Dr.Bob's rule that says: *'don't take anything for granted: create explicitly!'*.

Now, connect one ImgData- Navigator to the FieldGroup, and the other to the DataGrid (to get rid of the

```
ImgDataNavigatorX.XMLComponent
  = nil
```

warnings. Click on the FieldGroup, and specify XMLBroker1 as XMLBroker component, and do the same with the DataGrid. This gets rid of the two remaining design time warnings, and shows the output at design-time almost as we want it. The only thing that needs to be done is make sure that the DataGrid points to the detail records (the nested table) instead of the master records inside the XMLBroker. For that, we need to assign the name of the detail dataset to the DataGrid1.XMLDataSetField property.

Unfortunately, we cannot just open up the XMLDataSetField property of the DataGrid1 component. Somehow, this immediately yields the error message *'Missing data provider or data packet'*.

The combobox for the XMLDataSetField doesn't even open

up for us, so enough people will just give up here. In fact, I almost gave up myself, if it wasn't for Dan Miser (www.execpc.com/~dmiser) who was also convinced that this is a bug in the `XMLDatasetField` property editor. However, since we did set up our datasets with a nested dataset, we should be able to assign the property manually. In this case, it means that we need to type in the name of the dataset field manually, we can't select it from the drop-down list.

To make things even worse: each keypress will trigger the *'Missing data provider or data packet'* error. That is, until we complete the typing (of the name `ADOTable-Orders`) and press `Enter`. Then all of a sudden the connection will be made, and the detail records (in the nested dataset) are assigned to the `DataGrid`.

Of course, no-one at Inprise ever tried to use `XMLBroker` in a single tier, so it's not strange to see the `XMLDataSetField` property editor complaining about the lack of a Remote Server (instead of just the 'server-less' data provider). Fortunately, Dan presented me with the workaround, which is really a joy to demonstrate at sessions or conferences, I must say (especially when you hit the final Enter and everything suddenly works fine).

Before we can run the current standalone ISAPI52 web server DLL, we must make sure that we've thought of a few important things.

First, since we used ADO, make sure the `LoginPrompt` property of the `ADOConnection` component is set to `False` (otherwise, the web server will 'show' this dialog, which will result in a hung ISAPI52 DLL). Then, as we used a `Midas-PageProducer` producing XML, we must make sure that we've set the `IncludePathURL` to the location of the JavaScript libraries.

Next, since we are creating a WebBroker application we must make sure to create at least one `WebItemAction` and either assign its `PageProducer` to a valid `TPage-Producer` component, or write code in the `OnAction` event handler to producer output in the `Response` parameter.

Finally, because we're creating a web server application, we should set the output directory to the (web server) scripts directory on our machine, so we can recompile and re-run the WebBroker application. Note that this won't work with ISAPI DLLs, which remain loaded by IIS, but we'll get back to that in a moment.

When we've finished the above steps, it's time to deploy and test our ISAPI52.DLL with a web server such as IIS. Strangely enough, the output is empty! All we see is a title caption and two rows of image buttons, but no data whatsoever. It appears we didn't listen closely enough to Dr.Bob's rule (as defined earlier in this article): *'don't take anything for granted: create explicitly!'*

In this case, we should go back to the `FieldGroup` and `DataGrid`, and explicitly create the individual fields that we want to display. Note that usually with `DataSets`, we get all fields by default, and we have to use the Fields Editor to explicitly add or delete fields. We had a similar problem a while back with the WebBroker `Table-Producer` components, that also only allow us to remove fields after we've added them all.

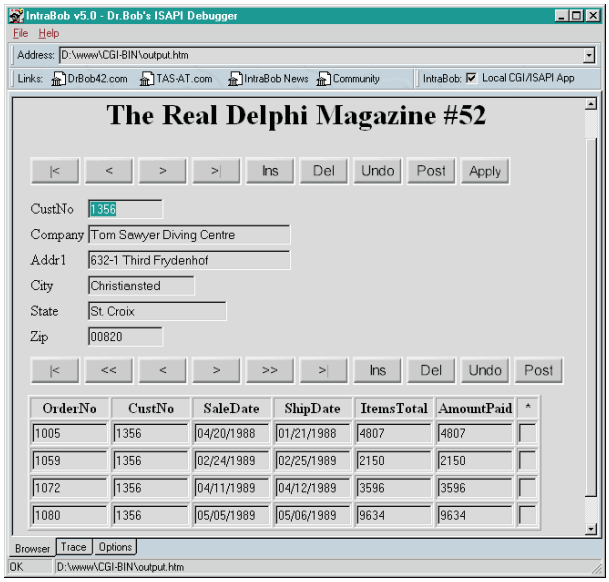
Anyway, just go back to the `MidasPageProducer` Web Editor, right click on the `FieldGroup` and `DataGrid` and in both cases explicitly add all the fields you want. This time, when we re-compile and re-run the application, we see the output as expected (including the connection between the master record and the detail records), as can be seen in Figure 3.

IntraBob v5.0

Note that the screenshot of Figure 3 wasn't taken using a regular browser, although internally it's the IE5 ActiveX control (ie the WebBrowser component as found in Delphi 5). Instead, the browser that I'm using is IntraBob, which was first developed in these very pages of *The **Real** Delphi Magazine* in order to test CGI and debug ISAPI applications, and later WebBroker apps (with full support for the `PathInfo` field). At this time, IntraBob version 5.0 also supports InternetExpress including XML packages and JavaScript libraries. To illustrate the usefulness of this tool, we can load the project of last month that contained the *'Next set of %d records (currently showing %d-%d)'* button. Frankly, without IntraBob, it would have been a lot harder to debug that project and see where the project failed (and when the table cursor was positioned to the first record again). Using IntraBob, we can debug ISAPI DLLs (which include WebBroker and InternetExpress), by specifying IntraBob as the host application in the `Run | Parameters` dialog (Figure 4).

Inside the ISAPI DLL, we are free to set any breakpoint we want (including new Delphi 5 features such as breakpoint groups, actions, etc). As soon as we hit the `Run` button, IntraBob is started, showing a `HOME.HTM` file that should contain a CGI form and an action item to launch the ISAPI DLL. We can set specific options in the `Options` tab, and just hit on `Submit` to execute the ISAPI DLL. By the way, I owe big thanks to Brian Long for his help in deciphering

➤ *Figure 3: Output of 1-Tier InternetExpress.*



the OLE `Variant` data pointer that was generated by the browser when the user clicks on the `Submit` button. Thanks to Brian's help, I can now 'kidnap' this package, and load the ISAPI DLL as a local process, instead of a remote one (loaded by the web server). This also makes sure ISAPI DLLs do not stay loaded when run by IntraBob (so it's a lot easier to re-compile and re-run them: you don't need to shut down your World Wide Web Service every two minutes).

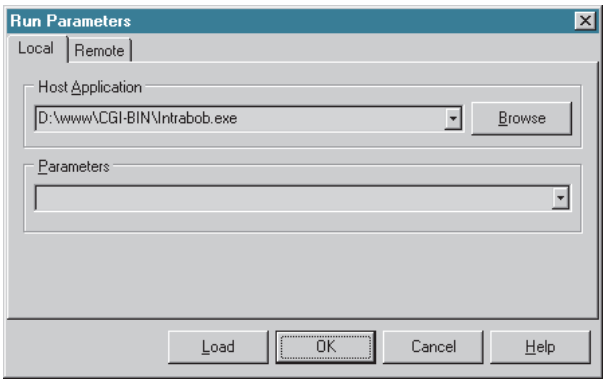
Although I've made it myself, I can only say that I can hardly imagine InternetExpress or any WebBroker development without the use of IntraBob. It's free at www.drbob42.com/tools (the version of IntraBob that can be found on the *Gold Issue* CD-ROM is a slightly older one which does not support InternetExpress nor XML/JavaScript).

Template Design

The final topic is, as so often, the final touch. So far, we've seen how to create HTML, XML and JavaScript output that consists of groups of fields, grids and navigators (with or without images as buttons), but from a non-developer perspective, the output still looks useless. We may be amazed that the browser does not flicker when we click the `Next` button (and the next record is fetched by the JavaScript from the XML data that's already part of the current web page), but most of our clients only see an otherwise ugly web browser, with ugly fonts and not even a scrollbar in the grid. Well, I can't blame them, and I can't give them a scrollbar in the grid either, but I can make the output a little more user-friendly.

The key to this all is the `HTMLDoc` property of the `WebMidasProducer`, which contains a pre-defined set of HTML lines:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#INCLUDES><#STYLES><#WARNINGS>
  <#FORMS><#SCRIPT>
</BODY>
</HTML>
```



➤ *Figure 4: Run | Run Parameters Host Application.*

We can add any HTML here, or remove any HTML code from here. The five special # tags are used by InternetExpress (some at design-time only, such as `WARNINGS`). We need to leave at least one tag in our HTML file to indicate where InternetExpress should insert forms, JavaScript, etc. The easiest way is to insert a new tag, namely `<#BODYELEMENTS>` in our HTML file. This tag is shorthand for the five `<#INCLUDES>` `<#STYLES>` `<#WARNINGS>` `<#FORMS>` and `<#SCRIPT>` tags.

By the way, you can find a more detailed explanation of the tags in the help (search for `BODYELEMENTS` and look at the help page titled *'Customizing the MIDAS page producer template'*).

Some editions of Delphi 5 Enterprise (I believe the US edition only, although I received a copy myself as well) included HotMetal Pro 5, which has built-in support for these tags! But usually I use a plain text editor to edit my HTML files. For our example, I've modified it a little bit to show a nice background, use friendly colours and a much nicer set of fonts (Comic Sans MS for the title and Verdana for the regular text, which are great fonts for websites). To make things more flexible, we can decide to give the `HTMLFile` property a value (pointing to a filename that contains the HTML source), which will override the value of the `HTMLDoc` property. Of course, you have to make sure the `HTMLFile` points to a valid file even when the ISAPI DLL is uploaded to the web server itself, but it means that you can update the look-and-feel of the

MIDAS Licensing Correction

In my *Under Construction* column in Issue 51 I wrote that I may have found a way to use InternetExpress in a single-tier 'web server app' that may not require a MIDAS licence.

It appears I have been wrong. The rule is as follows: a deployment licence does not need to be purchased *only* if the data packet does not pass machine boundaries.

Now, the XML data packet between the DataSetProvider and the XMLBroker stays on the same machine (the web server). However, the resulting HTML page that's being generated and shown on other machines (the web browser clients) also contain XML data, which, according to John Kaster of Inprise R&D, is also considered a MIDAS XML data packet.

So, we don't need a licence for MIDAS *only* if the XML packet *really* stays on the same machine, like a standalone browser that shows the data from the MidasPageProducer. But then why not use a Windows GUI?

The upside: according to John Kaster's reply at the InternetExpress chat, it appears the price for an unlimited Server licence for MIDAS 3 is now 'only' $2,500 (half the price of MIDAS 2's $5,000 which is still displayed on the website). For that amount, you can put any number of MIDAS Application Servers on one machine (and any number of clients using those servers, including internet clients).

Hmm, so how about sharing a web server amongst developers...?

web page by modifying the template, while the application itself doesn't have to be recompiled again. This means that our customers can make these changes themselves (using tools like the aforementioned HotMetal Pro, for example). And then we can blame our customers when the user interface looks bad...

Anyway, the `HTMLFile` variable points to my template, which is defined in Listing 1.

At design-time, this looks good. It even shows the right font for the field names and the grid titles, although it doesn't use the specified `SIZE=2` but the default `SIZE=3` instead. I don't know why, but Verdana looks good at size 2 and less good at size 3. Furthermore, at runtime we won't see the right font at all when using Netscape Navigator (the design-time browser is based on the Internet Explorer control). To make sure that the right font is used for the fieldgroup and datagrid as well, we must make sure to set either the correct style, or insert HTML code to set the font again.

For each field in the `FieldGroup`, I specify `ALIGN=RIGHT` in the `CaptionAttributes Custom` property, so that the field labels are right-aligned. I also append a colon to each `Caption` itself, and prefix the `Caption` with `<FONT FACE="Verdana" SIZE=2>`. When working through the list of fields, you will note that the correct font and font size will be used at design-time as well. And while working on the field captions, let's change `Addr1` to the more friendly `Address` instead. The same trick can be applied with the `Captions` of the `DataGrid`.

Furthermore, we can specify more custom `CaptionAttributes` for the `DataGrid`, such as the background colour of the column titles with `BGCOLOR=FFFF00` (to get a yellow background to the column heading). And while we're at it, note the `ReadOnly` property which is part of every field, whether part of a `FieldGroup` or `DataGrid`. This is quite useful to indicate that a given field cannot be changed (such as the `CustNo` or `OrderNo` fields, or when just browsing through the data). Setting the `ReadOnly` property to `True` makes sure that the end user cannot accidentally change the data at all.

Web designers can even edit the template in `HTMLFile` to include menus (to jump to other web pages), more images, and so on. But I think you will agree that even without those, the screenshot in Figure 5 looks better already compared to the one in Figure 3 (at least I think so).

➤ *Figure 5: Template based MidasPageProducer output.*

Next Time

Next month in the one and only real *Delphi Magazine*, we take a small step back to examine the enhancements that have been made in the WebBroker framework.

As you may well know, I'm quite fond of the WebBroker Technology, and I love the new InternetExpress stuff. But that was not the only recent Delphi 5 enhancement or change made in WebBroker, and some of them are very much worthy of our attention, so check back next month for more internet development in *Under Construction* (what else is new?)...

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

➤ *Listing 1*

```
<HTML>
<HEAD>
<TITLE>Under Construction #52 : InternetExpress (3)</TITLE>
</HEAD>
<BODY BACKGROUND="http://192.168.91.201/images/back.gif">
<FONT FACE="Verdana" SIZE=2>
<IMG SRC="http://192.168.91.201/images/bobhoed.gif">
<#BODYELEMENTS>
<P><HR><CENTER>
<FONT SIZE=1>This page &copy; 1999 by Bob Swart (aka Dr.Bob - www.drbob42.com)
</BODY>
</HTML>
```