Under Construction: Delphi 5 InternetExpress, 2

by Bob Swart

Delphi 5 InternetExpress combines the WebBroker technology with the MIDAS technology, producing HTML and XML as the final result for ultra-thin clients.

Last Time

Last month 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 the data format for the packets that were sent from the `DataSetProvider` to the `XMLBroker` (and back), and as the data format for the actual data embedded inside the HTML web pages that we could see inside the web browser.

This Time

So we are left with a few loose ends to explore. Like how to limit the amount of (XML) data being sent from the web server to the browser, and how to handle update (reconcile) errors. And I also want to show you ways to use the `InternetExpress` component without actually using a MIDAS licence (wouldn't that be fun).

AutoSessionName

Before we start I'd like to apologise for something that I forgot to mention last time (but hopefully readers that started to play with InternetExpress noticed it themselves). As usual with multi-user, multi-threaded and multi-session BDE applications, we need to place a `TSession` component on the (remote) data module and set the `AutoSessionName` to `True`. I completely forgot to mention that last time, and since I never ran the application more than once (something we *will* do this time), I never got into trouble. However, with a `Session` and `AutoSessionName` set to `True`, at least we know for sure that each individual request will be handled in the context of its own session, so no session conflicts will occur inside the BDE.

Alternatively, we could have used an `ADOConnection` component and two `ADOTables`, but then it wouldn't have been possible to demonstrate the Visual Data Module Designer. Due to a bug in the latter, we can't see the fields of `ADOTables`, nor 'draw' master-detail relationships between ADO tables in Delphi 5. This bug has been confirmed by Inprise and is said to be fixed in an internal build of Delphi 5 (so I guess it's only a matter of time until the first Delphi 5 Update Pack will be available).

Unlimited Data

Apart from the `Session` (which wasn't present), our master-detail demonstrated in the last issue presented us with a web page that didn't flicker or need a refresh to show other data. In fact, all the data was already provided (in XML packets) inside the web page. If we run the web server again and save the generated web page in a local file, it turns out that this local file is almost 100Kb in size. And that's only for a small customer-orders master-detail relationship. Do you remember Issue 49, where we measured distributed efficiency? It looks like we're about to face the same problem here: if we don't pay attention, all the available data from the MIDAS application server will be used to generate a (huge) web page for the client browser.

For a `TClientDataSet` component, all we had to do was change the `PacketRecords` property from -1 (meaning: 'send me all records as data packets') to a more sensible value of 20, and pay attention to `DBGrids` and `DBNavigator` components (who could jump to the 'last' record in the table by sending the entire table over the wire again). In Delphi 5's InternetExpress, the equivalent of a `ClientDataSet` is the `XMLBroker` component. And this component has a `MaxRecords` property which, again, is set to -1 by default (meaning: 'send all records as XML packets'). We can set this property to 20, and compile and execute the web server application again. `MaxRecords` indeed limits the number of records that are requested by the `XMLBroker` component, and we only get the first 20 customer (master) records and all their orders detail records. However, whenever we reach the end of this list in the browser (by clicking on the `Next` button 19 times or clicking on the `Last` button), there seems to be no way to get the next set of 20 records. And if we think about that, it's not too difficult to understand why: the `MidasPageProducer` requests all available data (as XML packets) from the `XMLBroker`, and produces a web page to browse through these records (with the data again packaged in XML format). The `MidasPageProducer`, however, does not know anything about the fact that the `XMLBroker` might have access to more than 20 records, let alone the fact that the `XMLBroker` was limited in the number of records in the first place. And since MIDAS 3 is now stateless (at the server side), there's no help from the application server either.

In fact, since the `IAppServer` interface (which controls all communication between the `DataProvider` on one side and `ClientDataSets` and `XMLBrokers` on the other side) is now stateless, we need to maintain state at the client side, and send the state information back to the server side when needed.

With TClientDataSet, such an event has been prepared for using the OnBeforeGetRecords events, in which we can put special (location) information in an OwnerData variable, which will be passed to the DataSetProvider OnBeforeGetRecord event. This way, we can position a DataSet to a certain location right before new records are provided. This works like a charm using TClientDataSet.

We can use a similar approach using XMLBrokers. Unfortunately, it's a bit more complex and involves more steps. First of all, we must make sure that our InternetExpress application somehow makes a callback to the web server application (to trigger the action that will request the next *n* records). Then, we need to implement the above behaviour of the OwnerData inside the OnBeforeGetRecords events of the XMLBroker and DataSetProvider in order to position the DataSet and return the next batch of *n* records. In theory, it should all work. In practice, however... just watch and wonder with us.

The FieldGroup is accompanied by NavigatorButtons. However, none of these buttons can trigger an external event: they all 'connect' to JavaScript code only. In order to trigger an external event, I would either have to modify the JavaScript code (a topic for another day), or somehow create a new button to trigger this action. The latter solution can be implemented by creating a QueryForm in the Web Editor/Design of the MidasPageProducer. The QueryForm also gets a special button: a QueryValueButton. This means whenever we click on the button, an event (Action) will be triggered, passing a special value inside a hidden field on the form. This is exactly what we need.

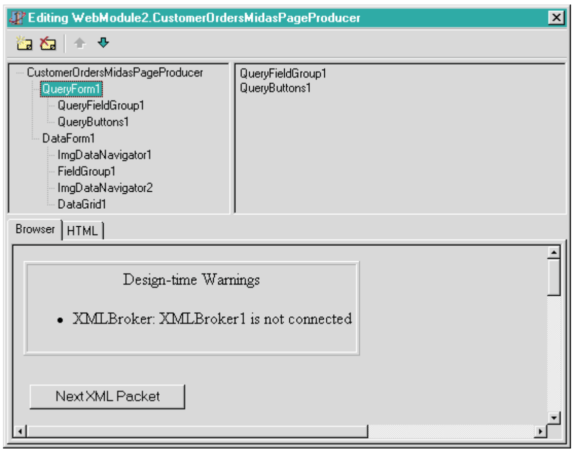
Specifically, we need to take the master-detail design from last month (one DataForm with a FieldGroup, DataNavigator, DataGrid and DataNavigator) and add a QueryForm to it. Figure 1 shows what this would look like at design-time.

By the way, never mind the design-time warnings that say XMLBroker1 is not connected. I've deliberately disconnected the XMLBroker at this time, because I always want to make sure to disconnect all MIDAS 'connection' components at design-time before I close a project. That way I can always open a client project again, even if the server is unavailable (for whatever reason).

The QueryForm contains a QueryFieldGroup and QueryButtons. The QueryFieldGroup consists of one QueryHiddenText field (with the name HiddenRecNo, to identify the fact that it contains the RecNo).

➤ *Listing 1*

```
procedure TWebModule2.WebModule2WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  { Log is a simple debug statement that takes a String argument }
const
  Str = 'Next set of %d records (currently showing %d-%d)';
var
  RecNo: String;
  RecNr: Integer;
begin
  RecNo := Request.QueryFields.Values['HiddenRecNo'];
  if RecNo <> '' then
  try
    RecNr := StrToInt(RecNo)
  except
    RecNr := 1
  end;
  NextXMLPacketButton.Caption := Format(Str,[XMLBroker1.MaxRecords,
    RecNr,RecNr+XMLBroker1.MaxRecords-1]);
  HiddenRecNo.Text := IntToStr(RecNr+XMLBroker1.MaxRecords-1);
  Log('HiddenRecNo #1: '+HiddenRecNo.Text);
  if DComConnection1.Connected then
    Log('DCOMConnection = True')
  else
    Log('DCOMConnection = False');
  { next statement may cause XMLBroker to Request Records }
  Response.Content := CustomerOrdersMidasPageProducer.Content;
end;
```



Editing WebModule2.CustomerOrdersMidasPageProducer

Design-time Warnings

- XMLBroker: XMLBroker1 is not connected

➤ *Figure 1: QueryForm with QueryFieldGroup and QueryButtons.*

The QueryButtons contains one button: a TSubmitQueryButton to submit the query and trigger an action from the web module. The SubmitQueryButton triggers the Action from its Parent QueryForm component, or more specifically the value of its Action property (a full URL that specifies the name of the web module as well as any pathinfo, should that be relevant). In our case, QueyForm.Action points to

```
http://localhost/cgi-bin/
  Client51.exe/
  CustomerOrdersMidasPageProducer
```

or just the same Web Module application, with the same

```
/CustomerOrdersMidasPageProducer
```

path information. In other words, we just start the same Action again. The only difference is that this action is started with a hidden value of RecNo. And this value was not present the very first time that we executed the URL above. In all subsequent events, the hidden field RecNo would contain something, namely the current value of the last RecNo that we've seen.

The SubmitQueryButton is visible at design-time (see Figure 1 again), and contains a caption, that currently says Next XML Packet. That's just something to show at design-time. At runtime, I'd like to show something like Next set of n records and possibly something like `Currently showing Y-Z`. Since the `Method` property of our `QueryForm` is set to `fmGet` (so we're using the `Get` protocol), it's easy to use the `QueryFields` to see if the `HiddenRecNo` field contains a value. If so, then we need to set the caption of the `NextXMLPacketButton` and increase the `HiddenRecNo` value by the number of `XMLBroker.MaxRecords`, see Listing 1.

Now, whenever we click on the `Next n Records` button, the action will fire, and as a consequence, the `XMLBroker` component needs to obtain a new set of records from the (remote) `DataSetProvider`. Only in this case, we'll get an event first, giving us the chance of positioning the `DataSet` (referenced by the `DataSetProvider`) to the exact location of the new batch of `XMLBroker1.MaxRecords` records.

Inside the `OnBeforeGetRecords` event of the `XMLBroker` we can write the code in Listing 2 to assign a `String` value to `OwnerData` (containing the `RecNo` value where the next batch of `XMLBroker1.MaxRecords` records should start).

Note that `HiddenRecNo` was already advanced by the number of `XMLBroker1.MaxRecords`, which is actually one time too many, so we must decrement it again before we assign it to the `OwnerData` variable. The `OwnerData` variable is then passed to the `OnBeforeGetRecords` event of the `DataSetProvider`, which it can use to position its `DataSet` component to the right `RecNo`, see Listing 3.

```
procedure TWebModule2.XMLBroker1RequestRecords(Sender: TObject;
  Request: TWebRequest; out RecCount: Integer; var OwnerData: OleVariant;
  var Records: String);
begin
  OwnerData := IntToStr(StrToInt(HiddenRecNo.Text) - XMLBroker1.MaxRecords);
  Log('HiddenRecNo #2: '+HiddenRecNo.Text);
end;
```

➤ *Listing 2*

Now, all we need to do is lie back and wait for the (correct) records to be sent in. Unfortunately, it doesn't quite work that way. For some reason (like I hinted last time), every time we hit the `Next n Records` button we increase our internal state (the button will show the next few numbers), but the `DataSet` seems to start at position one again, like always.

If you've played along with me so far, you're ready to try and debug this behaviour. So, let's start looking for a place to fix. Since I didn't believe anything was amiss in the InternetExpress Client, I focused mainly on the Server application (and the `DataSetProvider`) instead. To see what was going on, I placed some debug code on my `CustomerTable`, specifically on the `OnBeforeScroll` event (so I could see where the table would be scrolling to and when). This showed that the table was actually positioning itself correctly during the `OnBeforeGetRecords` event. Only to position itself at the very first record again at the actual `GetRecords` event. Ouch! A 'reset' was performed no matter where I positioned my `DataSet` component. Surely, this is not the way InternetExpress was intended to work, unless I completely misunderstood the use of the `OwnerData` variable and the text in the *Delphi 5 Developer's Guide* on pages 14 to 27, which demonstrates the above technique for 'regular' `ClientDataSets`, which works just fine, of course.

```
procedure TTDM.CustomerOrdersProviderBeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
var
  RecNo: Integer;
begin
  if OwnerData <> '' then begin
    RecNo := OwnerData;
    Log('DataSetProvider.OnBeforeGetRecords: '+IntToStr(RecNo));
    with (Sender AS TDataSetProvider) do begin
      DataSet.First;
      while (RecNo > 0) and not DataSet.Eof do begin
        Dec(RecNo);
        DataSet.Next
      end
    end
  end
end;
```

➤ *Above: Listing 3*

➤ *Below: Listing 4*

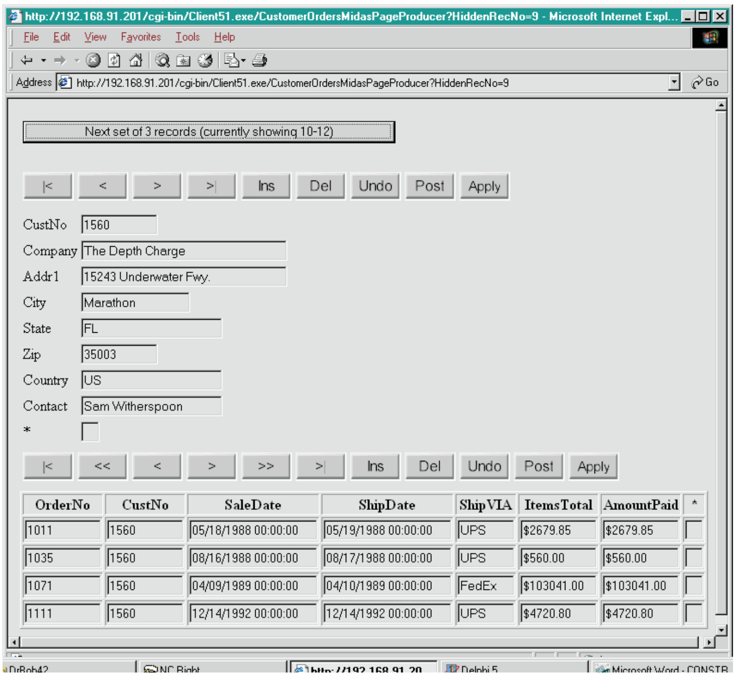
```
function TXMLBroker.GetXMLRecords(var RecsOut: Integer;
  var OwnerData: OleVariant; XMLOptions: TXMLOptions): string;
var
  ByteArray: OleVariant;
  Options: TGetRecordOptions;
begin
  Options := [grMetaData, grXML, grReset]; // remove grReset here !!
  RecsOut := 0;
  ByteArray := AS_GetRecords(MaxRecords, RecsOut, Byte(Options),
    '', PackageParams(Params), OwnerData);
  Result := FormatXML(VariantArrayToString(ByteArray), XMLOptions);
end;
```

Dr.Bob's Quick Fix

I did a lot of digging, and found a few ways to solve the problem. Possibly the best way is to take a look at the call to `TXMLBroker.GetXMLRecords` (Listing 4) which generates a call to the `AS_GetRecords` method (ie the remote application server and `DataSetProvider`) with the `grMetaData`, `grXML` and `grReset` options. The latter option is the one causing the problems, as it resets the `DataSet` we've just positioned. When I remove that option, things work fine again for me.

Note that I'm not 100% sure that this fix has no unwanted side effects. It works fine for me, in the situation presented in Listing 4, but I haven't tested it against every other InternetExpress application. In other words: use it at your own risk. And I welcome all feedback on this at drbob@chello.nl (thanks in advance).

If we apply the modified function `TXMLBroker.GetXMLRecords`, and recompile our project with the updated `XMLBrokr` unit, then we get the same result as last month, only this time we have an extra button that says (on the screenshot in Figure 2) `Next set of 3 records (currently showing 10-12)`, and indeed we see records 10-12 including the details. So it appears


http://192.168.91.201/cgi-bin/Client51.exe/CustomerOrdersMidasPageProducer?HiddenRecNo=9 - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Address http://192.168.91.201/cgi-bin/Client51.exe/CustomerOrdersMidasPageProducer?HiddenRecNo=9
Go
Next set of 3 records (currently showing 10-12)
|< < > >| Ins Del Undo Post Apply
|< << < > >> >| Ins Del Undo Post Apply
DrBob42
NC Right
http://192.168.91.20
Delphi 5
Microsoft Word - CONSTR

CustNo	1560
Company	The Depth Charge
Addr1	15243 Underwater Fwy.
City	Marathon
State	FL
Zip	35003
Country	US
Contact	Sam Witherspoon
*	□

OrderNo	CustNo	SaleDate	ShipDate	ShipVIA	ItemsTotal	AmountPaid	*
1011	1560	05/18/1988 00:00:00	05/19/1988 00:00:00	UPS	$2679.85	$2679.85	□
1035	1560	08/16/1988 00:00:00	08/17/1988 00:00:00	UPS	$560.00	$560.00	□
1071	1560	04/09/1989 00:00:00	04/10/1989 00:00:00	FedEx	$103041.00	$103041.00	□
1111	1560	12/14/1992 00:00:00	12/14/1992 00:00:00	UPS	$4720.80	$4720.80	□

➤ *Figure 2: InternetExpress showing 3 records at a time.*

to work! Of course, a similar technique can be used for the previous set of records, but I'm sure you get the idea by now.

So, why spend so much time on what is, at first sight, such a simple feature? Mainly because of what I pointed out at the beginning of this topic: distributed efficiency is becoming more and more important these days. Especially for real applications (which typically require more than a few dozen 'customer-orders' records). And although the above solution may need a little while to get adjusted to (we need to explain to users that they can browse through a subset of records, and request new records 'a set at a time'), it certainly reduces the bandwidth problems.

In time, I'll probably modify the JavaScript behind the buttons to automatically revert to a trigger of the web module action once we've reached the last record of the current XML packet, but that's a story for another day.

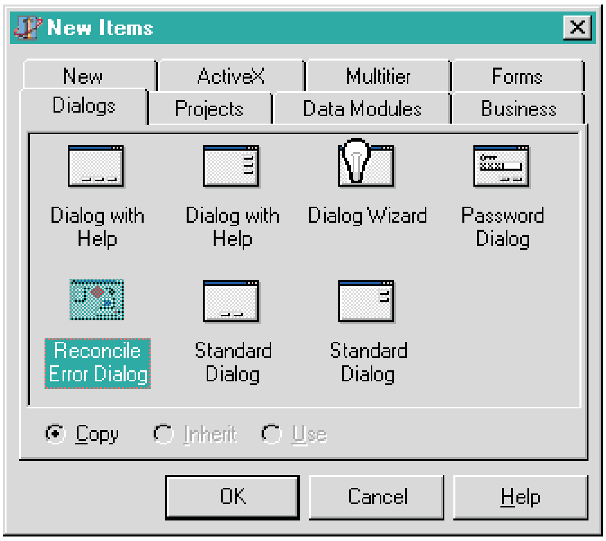
➤ *Listing 5*

```
procedure TRemoteDataModule.ClientDataSetReconcileError(DataSet: TClientDataSet;
  E: EReconcileError; UpdateKind: TUpdateKind; var Action: TReconcileAction);
begin
  Action := HandleReconcileError(DataSet,UpdateKind,E)
end;
```

Reconcile Error

Any multi-user application faces the potential danger of update conflicts. Where user A and user B both update the same record at the same time, user A changes one or more fields and user B also changes some records (maybe the same records). Posting the changes only 'saves' them locally, and with MIDAS we need to call the `ApplyUpdates` method (of `TClientDataSet`) to actually apply the updates to the remote dataset. So, the first user (say A) can call `ApplyUpdates` without a problem. However, when the second user (B) calls `ApplyUpdates`, an update error will occur: the original values in the record (passed on together with the new values) do not match the current values in the record. Oh, no! Time for some human intervention!

Experienced MIDAS developers will know that Borland has provided us with a special `Reconcile Error Dialog` to use in these situations. OK, I admit, you have to know where to look for it, but it's there anyway. Open the Object Repository (with `File | New`) and go to the `Dialogs` tab. There, you'll see the `Reconcile Error Dialog` in between the other (more common) dialogs (see Figure 3).



● Copy ○ Inherit ○ Use

➤ *Figure 3: Finding the MIDAS Reconcile Error Dialog.*

If you select this dialog and press `OK`, you'll find you get more than just a dialog: there's quite a lot of code 'behind the scenes' that comes with this repository item too. To use the dialog we simply need to add a call to `HandleReconcileError` in the `OnReconcileError` event handler of `TClientDataSet`, as in Listing 5.

The only other thing to keep in mind is that we should make sure that the `Reconcile Error Dialog` is not one of the Auto-created forms (which is now an option in Delphi 5, see the `Preferences` tab of the `Tools | Environment Options` dialog). Apart from that, using the `Reconcile Error Dialog` is as simple as one line of code, and it gives the user the ability to select a possible follow-up action (skip, abort, merge, correct, cancel or refresh). See Figure 4.

It should go without saying that InternetExpress web clients can also get into the same 'update trouble' (just like any other MIDAS client), but obviously we cannot show a `Reconcile Error Dialog` from inside a web browser. Fortunately, Delphi 5 Enterprise introduces the notion of a `ReconcilePageProducer`. Take a look at the `ReconcileProducer` property of the `XMLBroker`, where we can stick any 'producer' component. Unfortunately, a regular `PageProducer` or a `TableProducer` won't do the job (and neither will a `MidasPageProducer`, of course), we need an actual `ReconcileProducer`. So where do we find a specific `ReconcileProducer`? It's quite a bit hidden, actually. In the DEMOS/MIDAS/InternetExpress/INetXCustom directory, you will find two packages: inetxcustom.dpk (a runtime package) and dclinetxcustom.dpk (the design-time package) named *InternetExpress Sample Components* that contain additional InternetExpress components, including the `TReconcilePageProducer`, which is quite similar to the `Reconcile Error Dialog`.

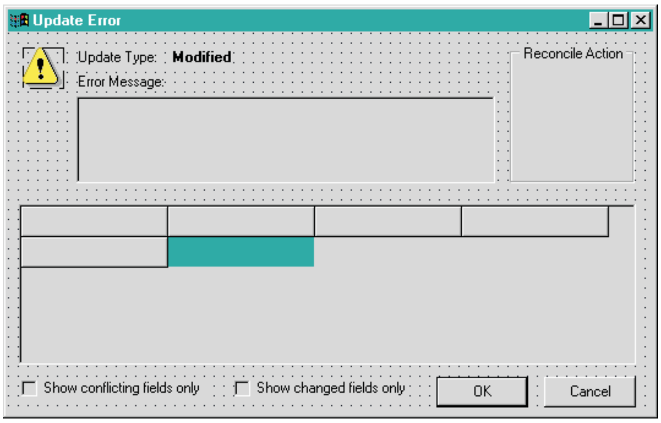
The `HTMLDoc` property of the `ReconcilePageProducer` contains a pre-defined reconcile error dialog. And although I have not yet been abel to test it for myself, I am sure that the `ReconcilePageProducer` also works fine with the special err.html HTML template file which can be found in the very same directory that holds the JavaScript files (the SOURCE\WEBMIDAS directory).

Golden InternetExpress

MIDAS 3 offers a lot of functionality, but at a price. Deployment fees, to be specific. There's one way to avoid this, and that's the standalone scenario. See

```
http://www.borland.com/midas/
  papers/licensing
```

for more information about when you need to buy a MIDAS licence. If I understand this document correctly (and it still needs to be updated for MIDAS 3), then in the standalone scenario, we can use either the `TClientDataSet` or the `TDataSetProvider` to manipulate data packets as long as they remain on the same machine.



Update Type: Modified

Error Message:

➤ *Figure 4: MIDAS Reconcile Error Dialog.*

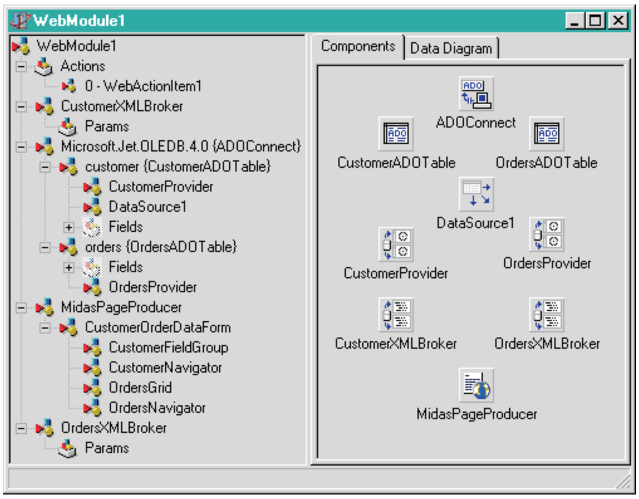
Well, that's what we'll be doing here: we can create a standalone InternetExpress application, without using a `ClientDataSet` but with using a `TDataSetProvider` and `XMLBrokers`, that probably doesn't require a MIDAS licence (but if you want to be really sure, you'd better contact your local Inprise office).

If you take a look at Figure 5, you'll see what I mean. It's a simple WebBroker CGI application. We start with a `ADOConnection` component, and connect it to our local DBDEMOS.udl file (with contains the well-known customers and orders tables). Next, drop two ADOTables on the Web Module, and connect them to the ADOConnection component. Select customer and orders as the TableNames. Now, drop a DataSource component, connect it to the CustomerADOTable, and use it to define a master-detail relationship between the customer and orders tables (remember that due to a bug in Delphi 5 it isn't possible at this time to define this relationship using the Data Diagram tab of the Visual Data Module Designer).

Now, after we have set up the master-detail relationship, we would normally need to export the tables using a DataSetProvider. That's what we'll do, but in this case we won't be needing to actually export them. Drop two DataSetProviders on the Web Module, and connect them to the CustomerADOTable and Orders-ADOTable. Now, instead of exporting them, drop two XMLBroker components on the same Web Module, and connect these two XMLBroker components directly to the DataSetProviders. Without needing a connection (or connection protocol) between them. Since we won't be needing a connection, we won't transfer any data from one 'tier' to another, so I believe I can safely state that this is a fair example of 'standalone use' of the InternetExpress technology. Which should be royalty-free.

The final step consists of adding a MidasPageProducer, and setting up the web page like we've been used to. This time, the master XML records will come from the CustomerXMLBroker component, while the detail XML records will not come from a nested dataset (if we try to open the XMLDataSetField property editor we'll get an error message), but instead ar eavailable in the second XMLBroker component: OrdersXMLBroker.

There's one thing to be aware of: inside the Web Editor (of the MidasPageProducer), we should explicitly add all fields we want to see inside a FieldGroup or DataGrid. Otherwise, we'll see some data and fields at design-time, but nothing at all at runtime (if is not explicitly created, it's not really there at all!).



➤ *Figure 5: Standalone InternetExpress Application.*

Finally, I want you to realise that this technique (of using InternetExpress and two logical tiers without actually needing two phyical tiers) can be used with ActiveForms or 'normal' Windows applications as well, of course. And using ADO, you are still spared from having to use the BDE *[Hooray! Ed]*.

Next Time

In the past two columns we've seen how to use InternetExpress to create distributed internet applications. Next time (in the third and, for now, last instalment of this topic), we'll see how we can extend InternetExpress, by producing 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. Yes, InternetExpress is fun and powerful, but you ain't seen nothing yet, *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.