Dynamic Web Images With Kylix

by Bob Swart

Last month, I showed you how to write web server applications with Kylix Desktop Developer: that is, without the use of WebBroker. There was one thing that I didn't show you (because my code didn't work when I wrote the original article), which was producing dynamic images out of a web server application written in Kylix.

Since last month, I've learned why my code wouldn't work, won't work in fact, but I have reached a compromise that will often work, and this article will explain to you the troubles, workarounds and other ways to produce dynamic images with Kylix.

BIOLIFE

The old BDE alias `DBDEMOS` contains many tables in Paradox or dBASE format. For Kylix, all these tables were converted to XML or binary CDS files. One of these old tables is biolife.db, which is available as biolife.xml and biolife.cds in Kylix. You can find these files in the demos/db/data directory (which isn't too obvious, if you ask me). The binary biolife.cds file is 1,103,803 bytes, while the somewhat more readable biolife.xml is 1,471,694 bytes. Because I've experienced some problems when storing images in XML files when using Delphi 5, I was glad to see that they apparently solved this issue, since Delphi 6 actually also contains the same two biolife files (apart from the original biolife.db table of `DBDEMOS`).

The trick to producing dynamic images from a web server (CGI or Apache DSO) application consists of two parts. The first part generates the HTML image tag, where the source is in fact making another web server request (passing the `RecNo` so it knows which record to handle), while the second part implements this specific web server request and actually does the work of returning the binary image data. The first and second web server requests can be implemented by two different web server applications or by one web server application that is capable of determining whether or not it should return HTML or the binary image data.

In this article, I've decided to write a single web server CGI application that implements both the original HTML content (including the image tag) and the binary image data production. In order to allow the single application to distinguish from one action to another, I've used a `GET` variable called `IMG` which has the value `yes` when we need to produce binary image data: in that case, we should also find a variable called `RecNo`, containing the specific record number, as well as a variable called `FieldName`, containing the name of the specific field we need to return.

HTML Production

Images are stored in `BLOB` fields (or more specifically, inside graphic fields), so we need to check the `Fields` property of the `DataSet` to see if a particular field is a `TGraphicField`. Note that in case we encounter a graphic field, then we also need to pass the `FieldName` itself as `GET` argument on the URL (because a dataset record may potentially contain more than one `TGraphicField`).

Note that I'm using the `ScriptName` 'trick' from last time in order to make sure that we call 'ourselves' again, this time with the `IMG=yes` argument to indicate that we should produce a binary image. The code to 'dispatch' the right action is very simple, and consists of a simple `if` statement (see the final Listing 4 at the end of this article).

Image Production

Once we're inside the second phase (or should that be second 'face'?) of our web server application, we know the field we must look at. The question is: how do we get the binary data out of a `TGraphicField`? There are actually two ways to do this. Using the high-end WebBroker technology, I've always saved the content of the field in a `MemoryStream`, and returned the `MemoryStream` to the `Response.ContentStream`. But we're not working with WebBroker now, and cannot use streams, I'm afraid. Fortunately, we can use the `AsString` property of the `TGraphicField` to get the content of the bitmap in (binary) string characters. As long as we don't do anything foolish (like converting the string to a `PChar`, which might be terminated by the first #0 in the string), the string can hold everything, including binary data!

The only problem is that the biolife table (and also the biolife.xml and biolife.cds generated files) don't just store the image in the `Graphic` field, but include an 8-byte Paradox graphic header as well. In other words: even if you save the content of

➤ *Listing 1: Record 2 HTML conversion (cross-platform code).*

```
procedure Record2HTML(const DataSet: TDataSet; RecNo: Integer);
var
  fields: Integer;
begin
  for fields:=0 to Pred(DataSet.FieldCount) do
    if DataSet.Fields[fields] IS TGraphicField then
      { GRAPHICS }
      writeln('<b>',DataSet.Fields[fields].FieldName,':</b> ',
        '<img src="',ScriptName,'?IMG=yes&RecNo=',RecNo,'&FieldName=',
        DataSet.Fields[fields].FieldName,'"><br>')
    else
      writeln('<b>',DataSet.Fields[fields].FieldName,':</b> ',
        DataSet.Fields[fields].AsString,'<br>')
end {Record2HTML};
```

```
procedure Table2Img(const TableName, FieldName: String; RecNo: Integer);
const
  Offset = 8; // sizeof Graphic header
var
  DataSet: TClientDataSet;
  Str: String;
  i: Integer;
begin
  DataSet := TClientDataSet.Create(nil);
  try
    DataSet.FileName := TableName;
    DataSetRecNo(DataSet, RecNo); // move to the right RecNo
    Str := (DataSet.FieldByName(FieldName) AS TGraphicField).AsString;
    for i:=Succ(Offset) to Length(Str) do
      write(Str[i]);
  finally
    DataSet.Close;
    DataSet.Free;
  end
end {Table2Img};
```

➤ *Listing 2: Table2Img conversion (cross-platform code).*

```
uses
  JPEG;
var
  Picture: TPicture;
  JPEG: TJPegImage;
begin
  Picture := TPicture.Create;
  Picture.Assign(DataSet.FieldByName(FieldName) AS TGraphicField);
  JPEG := TJPegImage.Create;
  JPEG.Assign(Picture);
```

➤ *Listing 3*

the `Graphic` field from the biolife table to disk, you wouldn't be able to view it. Skip the first eight bytes, however, and you'll end up with a perfect bitmap.

The only disadvantage is that the bitmap is in BMP format. As far as I am aware, this can only be shown inside Microsoft Internet Explorer, and not inside Netscape Navigator. As a result, the biolife table plus images presented by a web server application written in Kylix (and running on Linux) can only be seen correctly... inside Internet Explorer on Windows. Ouch!

Oh well, I guess Borland's marketing department could always decide to call it a cross-platform solution, so no harm done, right?

Bitmap Conversion

In order to try to solve this issue, I tried to fall back to the WebBroker solution. In Delphi, I always used the code snippet in Listing 3 to turn the content of the `TGraphicField` into a `Picture`, which could then be turned into a JPEG image! The interesting part of the Delphi code, for those who are interested, is in Listing 3.

Unfortunately, there are at least two reasons why the above code doesn't work in Kylix. First of all, there is no `JPEG` unit in Kylix (yet: but this may come; one other possibility may be to write the images out to BMP files and use another program to convert from BMP to JPEG: if you know of something suitable, do let me know!). Second, it turns out that as soon as you use a visual CLX component (or unit) inside a Kylix console application, then this will turn your CLX application into an X application. What does that mean? Well, the immediate result is that when you try to run a console application that uses the `QGraphics` unit, for example, inside a terminal or console window (without having started X), then you will get the error message *'could not connect to X server'*. And no matter what I did, I could not get my web server (console) applications to work with any of the visual CLX components or units. So no `QGraphics` unit for my web server applications in Kylix (this also applies to WebBroker, by the way).

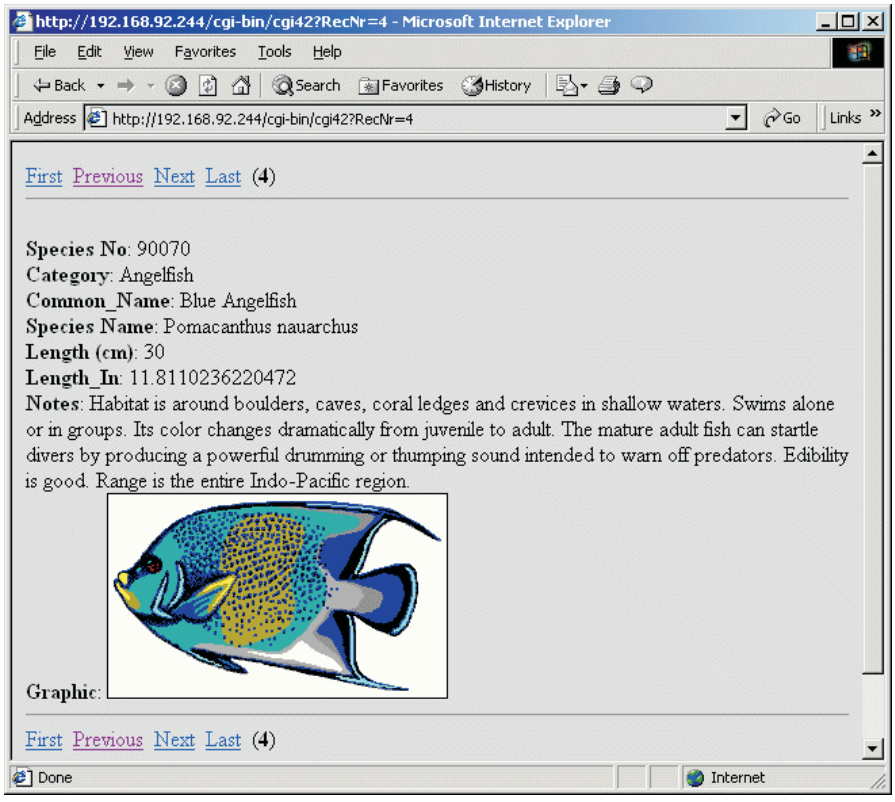
Final Result

I'm afraid that the final result still only produces dynamic bitmap images in the BMP format.

The final Listing 4 also contains the code for state management, so we can navigate through the biolife table and see new images appear as we move through the result set.

And since we use the `ScriptName` variable, it doesn't matter how you name this CGI application, it will always be able to redirect the second action to itself. Note that

➤ *Figure 1*



the DrBobCGI unit was listed last month and is on this month's disk too, and is available from my website at www.drbob42.com.

The result of a single biolife record in HTML, including the graphic and navigator links, can be seen in Figure 1.

Note that although we only call the Record2HTML (to turn a single record into HTML), I've also image-enabled the DataSet2HTML routine, so you could decide to turn the entire biolife table into a grid in your browser (generating multiple <img> tags, resulting in the browser showing multiple images, if you're using Internet Explorer, that is).

Conclusions

There are a number of things I've learned from this exercise. First of all, I didn't know you can't use visual CLX components inside web server applications (and this also

➤ *Listing 4: CGI application producing dynamic images (CLX code).*

```
program DrBob;
{$APPTYPE CONSOLE}
uses
  DrBobCGI, Classes, SysUtils, DB, DBClient;
procedure DataSet2HTML(const DataSet: TDataSet);
  var
    fields: Integer;
    RecNo: Integer;
  begin
    writeln('<table border=1>');
    DataSet.Open;
    write('<tr>');
    for fields:=0 to Pred(DataSet.FieldCount) do
      write('<td bgcolor=ffffff><b>',
        DataSet.Fields[fields].FieldName,'</td>');
    writeln('</tr>');
    DataSet.First;
    RecNo := 0;
    while not DataSet.Eof do begin
      Inc(RecNo);
      write('<tr>');
      for fields:=0 to Pred(DataSet.FieldCount) do
        if DataSet.Fields[fields] IS TGraphicField then
          { GRAPHICS }
          writeln('<td><img src="', ScriptName,
            '?IMG=yes&RecNo=', RecNo, '&FieldName=',
            DataSet.Fields[fields].FieldName, '"></td>')
        else
          write('<td>', DataSet.Fields[fields].AsString,
            '</td>');
      writeln('</tr>');
      DataSet.Next
    end;
    writeln('</table>')
  end {DataSet2HTML};
  procedure Record2HTML(const DataSet: TDataSet; RecNo:
    Integer);
  var
    fields: Integer;
  begin
    for fields:=0 to Pred(DataSet.FieldCount) do
      if DataSet.Fields[fields] IS TGraphicField then
        { GRAPHICS }
        writeln('<b>', DataSet.Fields[fields].FieldName,
          ':</b> ', '<img src="', ScriptName,
          '?IMG=yes&RecNo=', RecNo,'&FieldName=',
          DataSet.Fields[fields].FieldName, '"><br>')
      else
        writeln('<b>',DataSet.Fields[fields].FieldName,':
          </b> ',
          DataSet.Fields[fields].AsString,'<br>')
  end {Record2HTML};
  procedure NavigatorHTML(const DataSet: TDataSet;
    RecNo: Integer);
  begin
    if RecNo = 0 then
      RecNo := 1;
    if not DataSet.Active then
      DataSet.Open;
    write('<a href="',ScriptName,'?RecNo=1">First</a> | ');
    write('<a href="',ScriptName,'?RecNo=', Pred(RecNo),
      '">Prior</a> | ');
    write('<a href="',ScriptName,'?RecNo=',Succ(RecNo),
      '">Next</a> | ');
    write('<a href="',ScriptName,'?RecNo=-1">Last</a> | ');
    write('<a href="',ScriptName,'?RecNo=',RecNo,
      '">Refresh</a> ','(',RecNo,')<br>')
  end {NavigatorHTML};
  procedure DataSetRecNo(DataSet: TDataSet; var RecNo:
    Integer);
  var
    i: Integer;
  begin
    DataSet.Open;
    if RecNo = -1 then begin
      RecNo := 1;
      while not DataSet.Eof do begin
        Inc(RecNo);
        DataSet.Next
      end
    end else for i:=1 to Pred(RecNo) do
      DataSet.Next;
    if DataSet.Eof then begin
      // went past Eof, need to backtrack!
      Dec(RecNo); // one before Eof
      DataSet.First;
      for i:=1 to Pred(RecNo) do DataSet.Next
    end
  end {DataSetRecNo};
  procedure Table2HTML(const TableName: String; RecNo:
    Integer);
  var
    DataSet: TClientDataSet;
  begin
    DataSet := TClientDataSet.Create(nil);
    try
      DataSet.FileName := TableName;
      DataSet.Open;
      DataSetRecNo(DataSet, RecNo);
      NavigatorHTML(DataSet,RecNo);
      writeln('<hr>');
      Record2HTML(DataSet,RecNo);
      writeln('<hr>');
      NavigatorHTML(DataSet,RecNo);
      writeln('<hr>');
      DataSet2HTML(DataSet);
    finally
      DataSet.Close;
      DataSet.Free;
    end
  end {Table2HTML};
  procedure Table2Img(const TableName, FieldName: String;
    RecNo: Integer);
  var
    DataSet: TClientDataSet;
    Str: String;
    i: Integer;
  begin
    DataSet := TClientDataSet.Create(nil);
    try
      DataSet.FileName := TableName;
      DataSetRecNo(DataSet, RecNo);
      Str := (DataSet.FieldByName(FieldName) AS
        TGraphicField).AsString;
      for i:=9 to Length(Str) do
        write(Str[i]);
    finally
      DataSet.Close;
      DataSet.Free;
    end
  end {Table2Img};
const
  Biolife = 'biolife.cds';
var
  RecNo: Integer;
  Dir: String;
begin
  RecNo := StrToIntDef(Value('RecNo'),1);
  if Value('IMG') = 'yes' then begin
    writeln('content-type: image/bmp');
    writeln;
    Table2Img(Biolife,Value('FieldName'), RecNo)
  end else
  try
    writeln('content-type: text/html');
    writeln;
    writeln('<html>');
    writeln('<body bgcolor=ffffcc>');
    writeln(ScriptName,' = ',ParamStr(0),'<br>');
    GetDir(0,Dir);
    writeln('Working Directory: ',Dir,'<br>');
    writeln(RemoteAddress,'<hr>');
    try
      Table2HTML(Biolife, RecNo);
    except
      on E: Exception do
        writeln(E.ClassName,': ',E.Message)
    end
  finally
    writeln('</body>');
    writeln('</html>')
  end
end.
```

applies to some other CLX components, such as `Timers`). Second, it's a bad idea to store graphic fields inside database tables in the BMP format. You're better off storing them in JPG or PNG: this will make it easier to extract them and using PNG does not even compromise the bitmap quality itself.

But finally, I hope to have shown you this time and last month that you can get away with not using WebBroker for your Kylix web server applications. In these two articles, I've produced a set of helpful routines that can help you to produce HTML for a single record, entire datasets, a navigator, images, and more. And I've even shown you how we can handle more than one different request in the same web server application: a very simple action dispatcher, you could say.

By the time you read this, my www.drbob42.co.uk domain, hosted by TDMWeb (highly recommended with very professional web services), will be running on a nice Linux web server. With a cgi-bin directory that will feature a growing number of Kylix web server applications, both using WebBroker and 'plain', so if you're serious about Kylix for web server application development, be sure to check it out!

Bob Swart (aka Dr.Bob, www.drbob42.com) is an @-Consultant, Delphi Clinic Trainer and co-founder of the Kylix/Delphi OplossingsCentrum (www.KDOC.nl) in The Netherlands.