Under Construction: Database Web Server Extensions

by Bob Swart

In this article we'll turn our real live database application on the internet from the last issue into a web server extension DLL, using core ISAPI techniques. As with last month's article, we'll do this without Web Modules, so you don't need the Client/Server Edition of Delphi 3 to try this for yourself! Along the way, we'll encounter and overcome some BDE limitations.

CGI Versus ISAPI

Last month, we encountered a significant problem with CGI database applications: performance. For every request, the CGI application has to be loaded, the entire BDE has to be loaded, the table has to be opened, the current record must be found, followed by an optional update of the current record (if the data appears to have changed), performing the Action (one of the buttons the user clicks in the HTML form), dynamically generating the HTML CGI Form again with all the information, and finally shutting down the CGI application and the BDE. Whew! The load and unload times especially weigh heavily on the response times for even our simple example.

This month, we'll find out how we can improve the performance of the CGI application by turning it into an ISAPI application. Technically, this should mean only a few changes: the application must become a DLL, conform to the ISAPI API and should probably use a slightly different method of obtaining the values of the CGI variables. We'll also find out how we must change the way we work with the BDE to prevent clashes between multiple users. But first, let's examine some ISAPI basics.

The architectural difference between CGI applications and ISAPI web server extensions can be depicted as in Figure 1 (these two figures are taken from a paper by Diane Rogers, which can be found on the Inprise website).

ISAPI

ISAPI stands for the Internet Server API, and offers us the ability to write server-side DLLs that are just like CGI applications, but they run in the same memory space as the web server (thereby extending the web server's capabilities, hence the name web server extensions). ISAPI is implemented on the Microsoft Internet Information Server (IIS) and other ISAPI-compliant web servers. In practice, I only have experience with IIS 3.0 and IIS 4.0, but these seem to work just fine with the techniques I used.

There are two references on Microsoft's website for more information on ISAPI. The first one is an overview of ISAPI and can be found at www.microsoft.com/win32dev/apiext/isapimrg.htm. The second is the reference manual for the API, which is at www.microsoft.com/win32dev/apiext/ isapiref.htm.

Another helpful article from the Microsoft website is simply called *ISAPI Programming*, and is from the *Microsoft Interactive Developer*, January 1997 (at www.microsoft.com/mind/0197/isapi.htm). This says that '*server-side CGI scripting is inefficient; a process is started every time, and on Windows NT this technique is very costly. But you can achieve a dramatic improvement over CGI scripting with the Internet Server API framework.*'

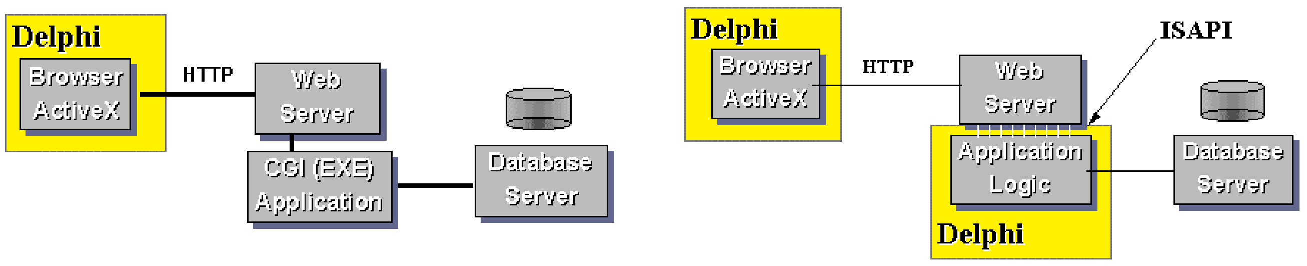
ISAPI DLLs live in the same address space as the HTTP server. They can therefore directly access the HTTP services available from the server. They load into memory more quickly and have much less overhead when it comes to making a call from the server. This can be particularly helpful if you are working under a heavy load. Unfortunately, this also means that the ISAPI DLL can bring down the web server if it goes wrong.

You can control when the DLL is loaded or unloaded. For instance, it is possible to preload DLLs for fast access on the first try or unload the ISAPI Applications DLLs that are not being used in order to free system resources. Unfortunately, you must sit at the web server console to unload ISAPI DLLs: I found no way to unload them from a remote location.

Delphi Support

Delphi 3 includes two support units for ISAPI: in C:\Program Files\Delphi 3\Source\RTL I found a unit ISAPI.PAS as well as a unit

➤ *Figure 1*



```
function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL; stdcall;
function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD; stdcall;
function TerminateExtension(dwFlags: DWORD): BOOL; stdcall;
```

➤ *Listing 1*

```
function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL; stdcall;
begin
  Ver.dwExtensionVersion := $00010000;  // we are interested in HTTP 1.0 support
  Ver.lpszExtensionDesc := 'Delphi 3 ISAPI DLL'; // Description of our ISAPI DLL
  Result := True
end {GetExtensionVersion};
```

➤ *Listing 2*

```
PEXTENSION_CONTROL_BLOCK = ^TEXTENSION_CONTROL_BLOCK;
TEXTENSION_CONTROL_BLOCK = packed record
  cbSize: DWORD;            // = sizeof(TEXTENSION_CONTROL_BLOCK)
  dwVersion: DWORD;         // version info of this spec
  ConnID: HCONN;            // Context ID (unique) Do not modify!
  dwHttpStatusCode: DWORD; // HTTP Status code
  // null terminated log info specific to this Extension DLL
  lpszLogData: array [0..HSE_LOG_BUFFER_LEN-1] of Char;
  lpszMethod: PChar;              // REQUEST_METHOD
  lpszQueryString: PChar;         // QUERY_STRING
  lpszPathInfo: PChar;            // PATH_INFO
  lpszPathTranslated: PChar;      // PATH_TRANSLATED
  cbTotalBytes: DWORD;            // Total bytes from client
  cbAvailable: DWORD;             // Available number of bytes
  lpbData: Pointer;               // pointer to cbAvailable bytes
  lpszContentType: PChar;         // CONTENT_TYPE
  GetServerVariable: TGetServerVariableProc;
  WriteClient: TWriteClientProc;
  ReadClient: TReadClientProc;
  ServerSupportFunction: TServerSupportFunctionProc;
end {TExtensionControlBlock};
```

➤ *Listing 3*

```
function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD; stdcall;
var
  Str: string;
  StrLen: Integer;
begin
  ECB.lpszLogData := 'Delphi DLL Log';
  ECB.dwHTTPStatusCode := 200;
  if ECB.lpszMethod = 'GET' then
    Str := 'Method: GET<BR>Data: [' + StrPas(ECB.lpszQueryString) + ']'
  else
    Str := 'Method: POST<BR>Data: [' + StrPas(PChar(ECB.lpbData)) + ']';
  Str := '<HTML>'+ '<HEAD>'+ '<TITLE>Dr.Bob''s ISAPI DLL</TITLE>'+ '</HEAD>' +
    '<BODY BGCOLOR=A7B7C7>' + '<H1>Dr.Bob says...</H1>' + '<HR><P>' + Str +
    '</BODY>' + '</HTML>';
  Str := Format('HTTP/1.0 200 OK'#13#10+'Content-Type: text/html'#13#10+
    'Content-Length: %d'#13#10+ 'Content:'#13#10#13#10'%s', [Length(Str), Str]);
  StrLen := Length(Str);
  ECB.WriteClient(ECB.ConnID, Pointer(Str), StrLen, 0);
  Result := HSE_STATUS_SUCCESS
end {HttpExtensionProc};
```

➤ *Listing 4*

called ISAPI2.PAS. These are for versions 1 and 2 respectively of the HTTP Server Extension Interface. Since an ISAPI version 2 compliant server will also support version 1, I decided to use the basic ISAPI.PAS file for this article. The original Delphi 3 Professional did not include these units, by the way, but they can be found on Inprise's website (www.inprise.com, go to Developer Support, Delphi, Downloads, file WEBAPI.ZIP).

The unit ISAPI.PAS can also be used with Delphi 2, although you either have to remove the line with `{$WEAKPACKAGEUNIT}`, or put it between `{$IFNDEF VER90}` and `{$ENDIF}` compiler directives.

ISAPI.PAS contains the interface to ISAPI, including the three functions that serve as entry points to ISAPI DLLs: see Listing 1. The first two are mandatory, while the third one, not defined in ISAPI.PAS by the way, is optional.

In our ISAPI DLL, we must export the first two functions in Listing 1 (I usually skip the `TerminateExtension` clean up function, as I do the clean up in the `HttpExtensionProc` itself anyway) to make sure the web server can load the ISAPI DLL and use `GetProcAddress` to get to the functions `GetExtensionVersion` and `HttpExtensionProc`.

GetExtensionVersion

The exported function `GetExtensionVersion` (Listing 2) is used to tell the web server who we are, and which level of HTTP support we expect. In our case, we expect HTTP 1.0 support.

TEXTENSION_CONTROL_BLOCK

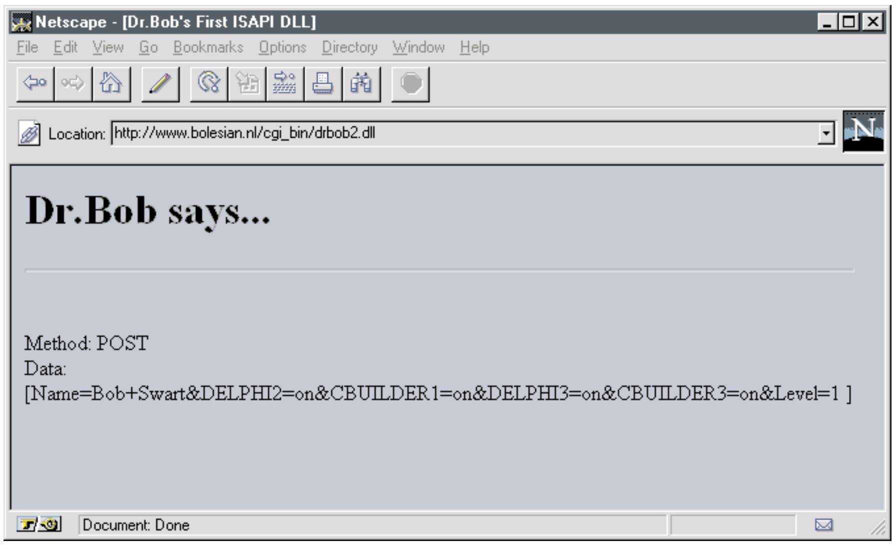
The exported function `HttpExtensionProc` is the main entry point for the DLL. This is where the work is done. The function gets one argument, `ECB`, of type `TEXTENSION_CONTROL_BLOCK`, defined in Listing 3.

The first parameter of this record is set to the size of the `TEXTENSION_CONTROL_BLOCK`, the second contains the HTTP version info, while the third contains our unique connection ID. We are not allowed to change any of these first three fields: they are filled by the web server upon calling the ISAPI DLL.

In a CGI Application, we access DOS environment variables to get the `REQUEST_METHOD` and after that the `CONTENT_LENGTH` (for `POST`) or `QUERY_STRING` (For `GET`). In an ISAPI DLL, we can use the argument `ECB` of type `TEXTENSION_CONTROL_BLOCK` we get in `HttpExtensionProc`, and look at the field `lpszMethod` to see if we're dealing with `POST` or `GET`. For the `POST` protocol, we need to look at `lpdData` (which contains `cbAvailable` bytes), while for `GET` we can get the input query from `lpszQueryString`. If the `POST`ed data contains more than 48Kb, then we should call ECB's `ReadClient` to get the remaining data from the web server (only the first 48Kb are put in `ECB.lpbData` or `ECB.lpszQueryString` fields). The ECB member function `ReadClient` is defined as:

```
function ReadClient(ConnID:
  HCONN; Buffer: Pointer;
  var Size: DWORD) :
  BOOL stdcall;
```

Apart from the ECB fields that already embed the `REQUEST_METHOD`, `QUERY_STRING`, `PATH_INFO`, `PATH_TRANSLATED` and `CONTENT_TYPE`, we can


Netscape - [Dr.Bob's First ISAPI DLL]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://www.bolesian.nl/cgi_bin/drbob2.dll
Document: Done

Dr.Bob says...

Method: POST
Data:
[Name=Bob+Swart&DELPHI2=on&CBUILDER1=on&DELPHI3=on&CBUILDER3=on&Level=1]

➤ *Figure 2*

get other 'environment variables' with a call to EBC's `GetServerVariable` function, defined as:

```
function GetServerVariable(
  hConn: HCONN;
  VariableName: PChar;
  Buffer: Pointer;
  var Size: DWORD) :
  BOOL stdcall;
```

We will skip the `ServerSupportFunction` for now, but it's important to know that `WriteClient` should be used to send dynamic HTML back to the web server (which then sends it back to the browser):

```
function WriteClient(ConnID:
  HCONN; Buffer: Pointer;
  var Bytes: DWORD;
  dwReserved: DWORD) :
  BOOL stdcall;
```

HttpExtensionProc

Now that we've seen what the ECB parameter given to the `HttpExtensionProc` routine contains, it's time to look at a sample implementation. As an example, let's consider a first ISAPI DLL that returns the Method (`GET` or `POST`) and the query string or data sent to the DLL itself (Listing 4).

To use our first ISAPI DLL, we can put it in the same scripts directory that we place our CGI applications in. Just like CGI applications, ISAPI DLLs can be called either directly (in a `<A HREF...>...</A>` link) or by using it as `METHOD` in an HTML Form. If we use the first ISAPI DLL (from the combined Listings 1 to 4, full source code is on the disk of course), then we get the result shown in Figure 2.

As we can see, it's easy to get our hands on the input data, and we can use the functions `Value` and `ValueAsInteger` from the last issue to use just like we've been used to.

With respect to the above output, it's of course easy to modify this ISAPI DLL to report more fields of the ECB structure. The difficult part is actually replacing the ISAPI DLL on the web server with the new version, as we can't overwrite an active DLL. We need to shut down the WWW service of the web server to de-activate the DLL and only then we can overwrite it with a new version. So it's not easy to take the RAD approach when developing ISAPI DLLs, and that's not even the biggest difference between CGI and ISAPI.

CGI ISAPI Differences

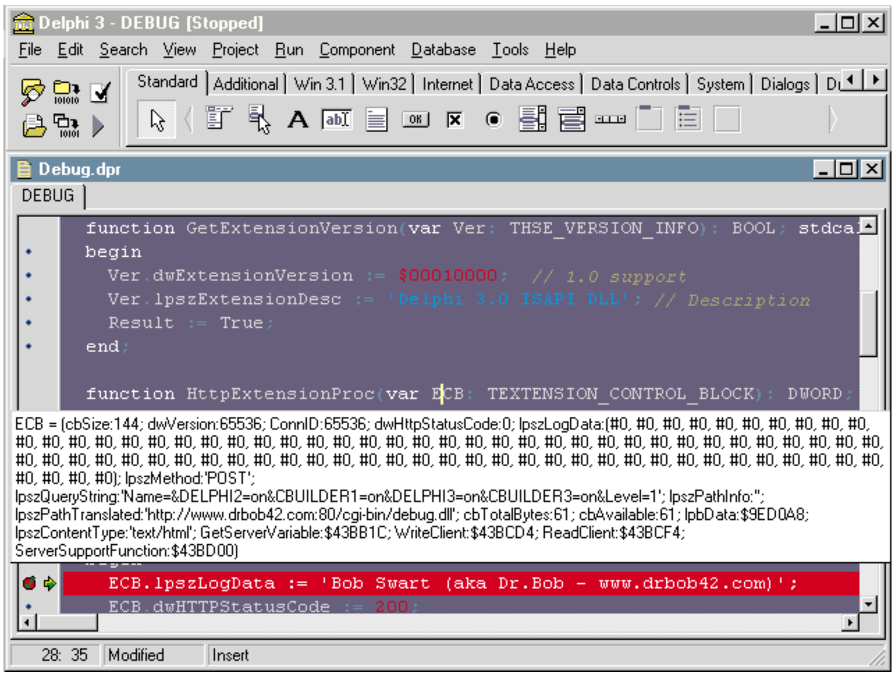
Inside a standard CGI application, we could just write the HTML codes to standard output. However, an ISAPI DLL has no standard output, and we must use the `WriteClient` member functions of the ECB to send HTML codes back to the client. So, if we want to port CGI applications to ISAPI, this means that we need to change every `writeln` statement in order to 'collect' the data into one big string that can be sent using `WriteClient` when processing is done. Alternatively, we can call `WriteClient` for every previous `writeln`, but this may inflict a performance penalty on the web server (all I/O will be done synchronously, so we'd better do a small number of I/O operations so as not to delay other ISAPI processes).

Fortunately, with the Delphi `AnsiString` type, we can just use a single variable and add pieces of the response to this long string. Not unlike the technique used in Delphi 3 Web Modules, where we assign an HTML response to the `Response.Content` property.

Another difference between CGI and ISAPI that we saw just earlier is the fact that the DLL is only loaded once, at the time of the first request. So, the first time won't be any faster than a CGI application, but because the DLL stays loaded, further requests are more efficient. A rogue ISAPI DLL can of course bring down the web server (as I found out, the hard way).

IntraBob for ISAPI

The fact that ISAPI DLLs are hard to debug was the main reason I decided to enhance IntraBob, my CGI Debugger) with ISAPI capabilities as well. IntraBob v3.0, included on the disk with this issue, is now also able to load ISAPI DLLs, that can be debugged from within the Delphi 3 IDE (just specify IntraBob to be the 'Host Application' for the ISAPI DLL). If you set it up this way, you can specify breakpoints inside your ISAPI DLL, and once inside a breakpoint, you can even use Code Insight to get a tooltip with the current value of the ECB (Figure 3). Great! Debugging ISAPI without a web server or browser: quite handy, if I may say so myself *[I can feel the heat of the satisfied glow from here! Ed]*.



```
function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL; stdca
begin
  Ver.dwExtensionVersion := $00010000;  // 1.0 support
  Ver.lpszExtensionDesc := 'Delphi 3.0 ISAPI DLL'; // Description
  Result := True;
end;

function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD;
```

```
ECB = (cbSize:144; dwVersion:65536; ConnID:65536; dwHttpStatusCode:0; lpszLogData:(#0, #0, #0, #0, #0, #0, #0, #0, #0, #0,
#0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0,
#0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0, #0,
#0, #0, #0, #0); lpszMethod:'POST';
lpszQueryString:'Name=&DELPHI2=on&CBUILDER1=on&DELPHI3=on&CBUILDER3=on&Level=1'; lpszPathInfo:'';
lpszPathTranslated:'http://www.drbob42.com:80/cgi-bin/debug.dll'; cbTotalBytes:61; cbAvailable:61; lpbData:$9ED0A8;
lpszContentType:'text/html'; GetServerVariable:$43BB1C; WriteClient:$43BCD4; ReadClient:$43BCF4;
ServerSupportFunction:$43BD00)
```

```
  ECB.lpszLogData := 'Bob Swart (aka Dr.Bob - www.drbob42.com)';
  ECB.dwHTTPStatusCode := 200;
```

➤ *Figure 3*

Multithreading

Another, very important, difference between CGI and ISAPI is that each invocation of an ISAPI DLL should be considered a thread within a single application. The *Delphi 3 Developer's Guide*, Chapter 3, discusses multithreaded database applications. Specifically, it says that they require multiple sessions. We'll examine this in detail in the next section.

Apart from sessions, multithreading means that we should avoid the use of global variables, or wrap them inside critical sections. The functions `Value` and `ValueAsInteger` from unit `DrBobCGI`, presented in the last issue, will have to be modified to work on local data (on the stack of the `HttpExtensionProc` routine for example) as opposed to global data that is shared by all sessions. This also means they have to be embedded inside any ISAPI DLL we write and cannot be part of a global unit (unless you want to pass the data itself as an argument, I don't).

Another important observation on this topic is that pre-initialised variables (in the global data segment) don't work as expected in an ISAPI DLL. Of course, they have the right value the first time we encounter them, but after we've changed them, subsequent threads will get the updated value instead of the initial value!

Finally, it seems that Microsoft's IIS creates a thread for the ISAPI DLL using standard Windows API calls and, as a consequence, the Delphi `BeginThread` RTL function is not called. If this routine is not called, to create a thread within the context of a Delphi DLL, then the variable `IsMultiThread` is never set to `True` and memory allocations by Delphi's memory manager are not thread-safe! The fix is easy: add the line `IsMultiThread := True` to the initialization of the ISAPI DLL.

ISAPI Wrapper

If we temporarily skip the actual generation of HTML code (inside a single Long String variable called Str) and focus on the ISAPI wrapper for BERT, the Bolesian Error Report Tool we designed and implemented as a CGI application last time, we come to the source code in Listing 5.

Note that this wrapper doesn't even access the BDE or any database components. That's all left for the last part of this article, in the included file BERT.INC.

ISAPI And The BDE

Given the information provided so far, it should be obvious that we can write just about any ISAPI application (based on the ISAPI wrapper in the previous listing, for example). However, there's still one more area we need to cover and that's database ISAPI applications or, more specifically, ISAPI and the Borland Database Engine. First of all, it's very important to have the latest version of the BDE installed, which may be found on Inprise's website at www.inprise.com/bde/. Currently, I'm using the BDE version 4.51, with the patch from the website.

Apart from that, we have already learned from the manuals (we always read the manuals, right?) that multithreaded database applications require multiple sessions. This is easy to forget, and at first an ISAPI DLL that uses only one (default) session seems to work fine. Until you perform a second request, that is, which just fails, returning an empty result page. It's not even possible to catch exceptions here, as the BDE will simply detect that the session is already in use, terminating the request, resulting in a lost connection and no result set. Only after a while, when the original request (and session) is de-activated (ie completed), does the session become available again. Once you realise that each thread must have its own session with the AutoSessionname set to True (to get a unique session name, which can then be assigned to the TTable components inside this particular ISAPI thread).

The maximum number of concurrent BDE sessions is 32. This means that the BDE may not be

➤ *Listing 5*

```
library BERT;
{.$DEFINE DEBUG}
uses
  Windows, SysUtils, ISAPI, IniFiles, DB, DBTables;
function GetExtensionVersion(var Ver: THSE_VERSION_INFO):
  BOOL; stdcall;
begin
  Ver.dwExtensionVersion := $00010000;  // 1.0 support
  Ver.lpszExtensionDesc := 'Delphi 3.0 ISAPI DLL';
  Result := True;
end {GetExtensionVersion};
function HttpExtensionProc(
  var ECB: TEXTENSION_CONTROL_BLOCK): DWORD; stdcall;
var
  Data: AnsiString; // Contains ISAPI input data
  function Value(const Field: ShortString): ShortString;
  var
    i: Integer;
    len: Byte absolute Result;
  begin
    Len := 0;
    i := Pos('&'+Field+'=',Data);
    if i = 0 then begin
      i := Pos(Field+'=',Data);
      if i > 1 then i := 0
    end else
      Inc(i); { skip '&' }
    if i > 0 then begin
      Inc(i,Length(Field)+1);
      while Data[i] <> '&' do begin
        if not (Data[i] in [#10,#13]) then begin
          { ignore CR/LF }
          Inc(Len);
          Result[Len] := Data[i]
        end else begin
          { CR/LF -> #32 }
          if (Len = 0) or (Result[Len] <> #32) then begin
            Inc(Len);
            Result[Len] := #32
          end
        end;
        Inc(i)
      end
    end;
    while (Len > 0) and (Result[len] = #32) do
      Dec(len)
  end {Value};
  function ValueAsInteger(const Field: ShortString):
    Integer;
  begin
    try
      Result := StrToInt(Value(Field))
    except
      Result := 0
    end
  end {ValueAsInteger};
{$I BERT.INC}
{ see next listing for procedure
  GenerateContents(var Str: String); }
  var
    i: Integer;
    Str: AnsiString;
  begin
    Str := 'Hello, world!';
    try
      try
        // parse ECB input data
        if StrPas(ECB.lpszMethod) = 'POST' then
          Data := StrPas(ECB.lpbData)
        else
          Data := ECB.lpszQueryString;
        if (Length(Data) > 1) and (Data[Length(Data)] = #0)
          then Delete(Data,Length(Data),1);
        i := 0;
        while i < Length(Data) do begin
          Inc(i);
          if Data[i] = '+' then Data[i] := ' ';
          if Data[i] = '%' then begin
            { special code }
            Str := '$00';
            Str[2] := Data[i+1];
            Str[3] := Data[i+2];
            Delete(Data,i+1,2);
            Data[i] := Chr(StrToInt(Str))
          end
        end;
        if i > 0 then
          Data[i+1] := '&'
        else
          Data := '&';
        // initialize ECB output data
        ECB.lpszLogData :=
          'BERT - Bolesian Error Report Tool';
        ECB.dwHTTPStatusCode := 200;
        // create the dynamic HTML webpage here inside STR
        try
          GenerateContents(Str);  // see BERT.INC
          Str := '[' + Data + ']<P>' + Str
        except
          on E: Exception do
            Str := Str + '<P>IN<P><HR><P>' + E.ClassName +
              ' ' + E.Message
        end;
      except
        on E: Exception do
          Str := Str + '<P>OUT<P><HR><P>' + E.ClassName +
            ' ' + E.Message
      end;
    finally
      // finalize ECB output data
      Str := Format('HTTP/1.0 200 OK'#13#10+
        'Content-Type: text/html'#13#10+
        'Content-Length: %d'#13#10+
        'Content:'#13#10#13#10'%s', [Length(Str), Str]);
      i := Length(Str);
      ECB.WriteClient(ECB.ConnID, Pointer(Str), i, 0)
    end;
    Result := HSE_STATUS_SUCCESS
  end {HttpExtensionProc};
exports
  GetExtensionVersion,
  HttpExtensionProc;
begin
  IsMultiThread := True
end.
```

suitable for high traffic sites unless you work out some way of scheduling the connections (which, by the way, is implemented by Delphi 3 Web Modules).

The include file BERT.INC contains the local procedure `GenerateContents`, which in its turn can call the local routines `Value` and `ValueAsInteger` (that work on the `Data` variable on the local stack for this thread). Note that we need to store the dynamic HTML that we generate in a single `Long String` (called `Str`), which is sent back to the client using a single `WriteClient`, see Listing 6.

For each request, which translates into a thread, we dynamically create a `TSession`, activate it (to get the unique `SessionName`), create a `TTable`, assign its `SessionName` property to the `Session.SessionName`, and free both the `TTable` and the `TSession` afterwards. The main performance gain lies in initialization of the BDE, which will be available after the very first request.

If we test both BERT.DLL and BERT.EXE (from last month) online, we'll find that BERT as an ISAPI DLL starts with the same speed (or lack thereof) as the CGI application. However, after the initial load of the BERT.DLL and BDE files, all subsequent requests are much faster. Even with multiple users walking through a single table, updating records, deleting or inserting records, it's fast. As long as we don't have more than a few dozen connections we shouldn't get into problems.

If we ever need more than the BDE's maximum 32 sessions I guess that means it is time to investigate a three tier solution using the Inprise middleware tools like MIDAS, Entera or CORBA and the VisiBroker Object Request Broker. At least we'll know for sure that these tools will have no problem integrating in our current and future Borland Delphi environment (as we shall see in a future article, I promise...).

Framework

All in all, the end result from this month's column is not a truly generic source framework, like that we produced for Delphi CGI applications, but we can of course still use the final source code as a template for future ISAPI database DLLs.

As long as we keep in mind that ISAPI DLLs are multithreaded and each thread requires a unique `Session`, we shouldn't encounter too many problems along the way. Until we need to upgrade to a three tier solution, that is.

Remember also that it's always helpful to be able to test and debug an ISAPI DLL on a local machine, using IntraBob 3.0, in order to avoid an accidental programming error bringing down the web server. Even if it's an accident, your web server administrator may sometimes forgive but seldom *forget* these bad experiences!

Next Time

Last month, I also promised to design some nice HTML CGI-Forms to find a given record in the table, or to perform a dynamic SQL query and display the results as well. Time and space constraints unfortunately prevent me from reporting on these two issues. However, I'm sure the ISAPI techniques I've presented in this paper are more than enough to keep you going for a little while.

➤ *Listing 6*

```
procedure GenerateContents(var Str: String);
const IniFile = '.\report.ini';
procedure DataSetTable(DataSet: TDataSet; NewRec: Boolean);
  { NEW RECORD, Actions: POST,CANCEL }
  { BROWSE RECORD, Actions: FIRST,PREV,NEXT,LAST,INSERT,DELETE,REFRESH}
  const
    Int: Array[1..9] of Char = '123456789';
  var
    i,j,col,items: Integer;
    option: ShortString;
  begin
  {$IFDEF DEBUG}
    Str := Str + '<P>';
    Str := Str +
      'Debug Action: <INPUT TYPE=TEXT NAME=Action>'#13#10;
    Str := Str + '<P>';
  {$ENDIF}
    if NewRec then begin
      Str := Str +
        '<INPUT TYPE=SUBMIT NAME=Action VALUE=Post>'#13#10;
      Str := Str +
        '<INPUT TYPE=SUBMIT NAME=Action VALUE=Cancel>'#13#10
    end else begin
      Str := Str +
        '<INPUT TYPE=SUBMIT NAME=Action VALUE=First>'#13#10;
      Str := Str +
        '<INPUT TYPE=SUBMIT NAME=Action VALUE=Prev>'#13#10;
      Str := Str +
        '<INPUT TYPE=SUBMIT NAME=Action VALUE=Next>'#13#10;
      Str := Str +
        '<INPUT TYPE=SUBMIT NAME=Action VALUE=Last>'#13#10;
      Str := Str + '&nbsp;'#13#10;
      Str := Str + '<INPUT TYPE=SUBMIT NAME=Action ‘+
        ‘VALUE=Insert>'#13#10;
      Str := Str + '<INPUT TYPE=SUBMIT NAME=Action ‘+
        ‘VALUE=Delete>'#13#10;
      Str := Str + '&nbsp;'#13#10;
      Str := Str +
        '<INPUT TYPE=SUBMIT NAME=Action VALUE=Find>'#13#10;
      Str := Str +
        '<INPUT TYPE=SUBMIT NAME=Action VALUE=Query>'#13#10;
      Str := Str +'&nbsp;'#13#10;
      Str := Str + '<INPUT TYPE=SUBMIT NAME=Action ‘+
        ‘VALUE=Refresh>'#13#10;
    end;
    Str := Str + '<INPUT TYPE=RESET VALUE=Reset>'#13#10;
    Str := Str + '<P>'#13#10;
    with DataSet do begin
      if NewRec then
        Str := Str + '<INPUT TYPE=HIDDEN NAME="' +
          Fields[0].FieldName+'" VALUE="-1">'#13#10
      else
        Str := Str + '<INPUT TYPE=HIDDEN NAME="'+
          Fields[0].FieldName+'" VALUE="'+
          Fields[0].AsString+ '">'#13#10;
      Str := Str +'<TABLE BGCOLOR=BBBBBB BORDER><TR>'#13#10;
      col := 0;
      with TIniFile.Create(IniFile) do
        try
          for i:=1 to FieldCount-1 do begin
            { first field was hidden }
            if Fields[i].DataType = ftMemo then begin
              Str := Str + '</TR><TR><TD COLSPAN=3>';
              col := 3;
            end else
              if Fields[i].Size > 99 then begin
                Inc(col,2);
                if col > 3 then begin
                  Str := Str + '</TR><TR>';
                  col := 2
                end;
                Str := Str + '<TD COLSPAN=2>'
              end else begin
                Inc(col);
                if col > 3 then begin
                  Str := Str + '</TR>'#13#10'<TR>';
                  col := 1
                end;
              end;
            Str := Str + '<TD>'
          end;
          Str := Str +'<B>'+ReadString(Fields[i].FieldName,
            'Name',Fields[i].FieldName)+'</B><BR>';
          items := ReadInteger(Fields[i].FieldName,
            'Items',0);
          if items = 0 then begin
            if Fields[i].DataType = ftMemo then begin
              Str := Str + '<TEXTAREA NAME="'+
                Fields[i].FieldName+'" ROWS=6 COLS=72>';
{ ** CONTINUED ON FACING PAGE ---> ** }
{ ** CONTINUED FROM FACING PAGE ** }
              if not NewRec then
                Str := Str + Fields[i].AsString;
              Str := Str + '</TEXTAREA>'
            end else begin
              if Fields[i].Size > 99 then
                Str := Str + '<INPUT TYPE=text NAME="'+
                  Fields[i].FieldName+'" SIZE=64'
              else
                if Fields[i].Size = 0 then
                  Str := Str + '<INPUT TYPE=text NAME="'+
                    Fields[i].FieldName+'" SIZE=30'
                else
                  Str := Str + '<INPUT TYPE=text NAME="'+
                    Fields[i].FieldName+'" SIZE='+
                    IntToStr(Fields[i].Size);
              if not NewRec then
                Str := Str+
                  ' VALUE="'+Fields[i].AsString+'"';
              Str := Str + '>'
            end
          end else begin
            Str := Str + '<SELECT NAME="'+
              Fields[i].FieldName+'">';
            for j:=1 to items do begin
              option := ReadString(Fields[i].FieldName,
                'Item'+Int[j],Int[j]);
              if (not NewRec) and
                (option = Fields[i].AsString) then
                { selected }
                Str := Str + '<OPTION SELECTED VALUE="'+
                  option+'">'+option+' '
              else
                Str := Str + '<OPTION VALUE="'+
                  option+'">'+option+' '
            end;
            Str := Str + '</SELECT>'
          end;
          Str := Str + '</TD>'
        end;
        Str := Str + '</TR>'#13#10
      finally
        Str := Str + '</TABLE>'#13#10;
        Free
      end
    end
  end;
const
  {no alias: ChDir and use current directory on web server}
  _DatabaseName = '';
  _TableName = 'report.db';
  Action: String[7] = '';
var
  Table: TTable;
  Session: TSession; { IMPORTANT }
  Report,i: Integer; { key field }
  NoChange: Boolean;
begin
  Str := '';
  Action := '';
  ShortDateFormat := 'DD/MM/YYYY';
  GetDir(0,Str);
  if IOResult <> 0 then
    { skip };
  Str := Str + '<HTML>'#13#10;
  with TIniFile.Create(IniFile) do
  try
    Str := Str + '<HEAD>'#13#10;
    Str := Str + '<TITLE>'+ReadString(_TableName,'Name','')+
      '</TITLE>'#13#10;
    Str := Str + '</HEAD>'#13#10;
    Str := Str + '<BODY BGCOLOR=AAAAAA>'#13#10;
    Str := Str + '<CENTER>'#13#10;
    Str := Str + '<H1>';
    Str := Str + '<IMG SRC="'+
      ReadString(_TableName,'Bitmap','')+'">';
    Str := Str + ReadString(_TableName,'Name','');
    Str := Str + '</H1>'#13#10;
    Str := Str + '<FORM METHOD=POST ACTION="'+
      ReadString(_TableName,'Action','')+'">'#13#10
  finally
    Free
  end;
  Session := TSession.Create(nil);  // IMPORTANT
  Session.AutosessionName := True;  // IMPORTANT
  Session.Active := True;           // IMPORTANT
  Table := TTable.Create(nil);
  Table.SessionName := Session.SessionName;
  with Table do
  try
    Active := False;
    TableType := ttParadox;
    { DatabaseName := _DatabaseName; }
    TableName := _TableName;
    Open;
    First;
    { locate current record }
    Report := ValueAsInteger('Report');
    if Report > 0 then
      FindKey([Report])
    else
      First;
```

```
    { update record if data has changed }
    NoChange := True; { assume no change }
    if (Value('_'+Fields[0].FieldName) <> '') and
      (ValueAsInteger(Fields[0].FieldName) <> -1) then begin
      NoChange := True; { assume no change }
      for i:=0 to FieldCount-1 do
        NoChange := NoChange AND
          (Value('_'+Fields[i].FieldName) =
          Value(Fields[i].FieldName));
      if not NoChange then begin
        { update record. check if data in table is still same }
        NoChange := True;
        for i:=0 to FieldCount-1 do
          NoChange := NoChange AND
            (Value('_'+Fields[i].FieldName) =
            Fields[i].AsString);
        if not NoChange then being
          { table changed!! }
          Str := Str + '<B>Error: value of record changed ‘+
            ‘before your update was made!</B>';
          Action := 'Refresh' { force refresh }
        end else begin
          { go ahead! }
          Str := Str + '<FONT SIZE=2>Note: ';
          Edit; { set Table in Edit-mode }
          for i:=0 to FieldCount-1 do begin
            if (Value('_'+Fields[i].FieldName) <>
              Value(Fields[i].FieldName)) then begin
             {$IFDEF DEBUG}
              Str := Str + IntToStr(i)+'['+Value('_'+
                Fields[i].FieldName)+ ']-{'+
                Value(Fields[i].FieldName)+ '} ';
             {$ENDIF}
              Fields[i].AsString :=
                Value(Fields[i].FieldName) { new }
            end
          end;
          Post { Post data in Table };
          Str := Str + ' previous record updated in table’+
            ’</FONT><P>'#13#10
        end
      end
    end;
    { determine action }
    if Action = '' then
      Action := Value('Action');
    if Action = '' then
      Action := 'First';
    { perform action }
    if Action = 'First' then
      First
    else if Action = 'Next' then
      Next
    else if Action = 'Prev' then
      Prior
    else if Action = 'Last' then
      Last
    else if (Action='Find') or (Action='Query') then begin
      // TODO: special query CGI-Form
    end else if Action = 'Delete' then
      Delete
    else if Action = 'Insert' then
      { skip }
    else if Action = 'Post' then begin
      { insert record }
      First;
      Report := 0;
      while not Eof do begin
        if Fields[0].AsInteger > Report then
          Report := Fields[0].AsInteger;
        Next
      end;
      Inc(Report);
      Insert;
      Fields[0].AsInteger := Report;
      for i:=1 to FieldCount-1 do
        Fields[i].AsString := Value(Fields[i].FieldName);
      Post
    end else if Action = 'Cancel' then
      { cancel }
    else
      { Refresh };
    Str := Str + '<P><B>' + Action + '</B><P>';
    for i:=0 to FieldCount-1 do
      Str := Str + '<INPUT TYPE=HIDDEN NAME="_'+
        Fields[i].FieldName+ '"VALUE="'+
        Fields[i].AsString+ '">'#13#10;
    Str := Str + Fields[0].AsString+' - '+IntToStr(RecNo)+
      '/'+IntToStr(RecordCount)+ '&nbsp;&nbsp;'#13#10;
    { generate HTML CGI-Form with fields }
    DataSetTable(Table,Action = 'Insert');
    Close
  finally
    Str := Str + '</FORM>'#13#10;
    Str := Str + '</BODY>'#13#10;
    Str := Str + '</HTML>'#13#10;
    Free
  end;
  Session.Free;    // IMPORTANT
  Session := nil;  // IMPORTANT
  Table := nil     // IMPORTANT
end;
```

Next month, we'll explore yet another user feedback question: *how can we send an email message directly from a web server, without human intervention?* Not that I want you all to become email 'spammers' of course, but there are many other legitimate and useful applications of such a technique.

This will also be the first step along the way to Robot-Bob: the automatic 'intelligent agent' on the web server, that can search the web for specified items, and report its findings by email.

Bob Swart (aka Dr.Bob, visit www.drbob42.com) is a professional knowledge engineer technical consultant using Delphi, C++Builder and JBuilder for Bolesian (www.bolesian.com) in The Netherlands. In his spare time, Bob likes to watch video tapes of *Star Trek Voyager* and *Deep Space Nine* with his 4 year old son Erik Mark Pascal and his 1.5 year old daughter Natasha Louise Delphine.