

# Dr Bob's Prescriptions

## Nº.1 DataSetPageProducer

*Dr Bob's Prescriptions* is the title of a new column for the UK-BUG Developer's Magazine, in which I'll try to give you a regular prescription or recipe for a useful technique or tweak for the existing internet and web development toolsets in Delphi, C++Builder and Kylix: most notably WebBroker and the new WebSnap.

### WebBroker PageProducers

As regular WebBroker developers will already know, a TPageProducer has two properties to specify (or contain) an HTML template: HTMLFile and HTMLDoc. HTMLFile points to an external HTML file. This is useful if we want to be able to change our webpage template without having to recompile our entire application itself. The HTMLDoc property, on the other hand, is of type TStrings and contains the HTML text itself (i.e. this will be hard coded in the .dfm or .xfm file).

The HTML template of a TPageProducer component can contain any HTML code as well as special #-tags. These #-tags are invalid HTML tags, so they will be ignored by browsers, but are accepted by the OnHTMLTag event of the TPageProducer itself. Inside this event, we can change an encountered TagString and replace it with a ReplaceText. For more flexibility, #-tags can also contain parameters, right after the name itself (like a parameter *Format=YY/MM/DD* to specify the format in which to convert and represent the date).

**Warning:** the HTMLDoc and HTMLFile properties cannot be used at the same time! When you enter a value in the HTMLFile property (and move the focus to another property or component to post the change in the HTMLFile property), then the content of the HTMLDoc property will be cleared. So, if you decide to move the content of your HTMLDoc property to an external file specified by HTMLFile, make sure to copy the contents before you enter the filename (been there, done that, got to retype it all over again).

### TDataSetPageProducer

The TDataSetPageProducer component is derived from the TPageProducer. Instead of just replacing #-tags with a regular value, the TDataSetPageProducer has a DataSet property, and will try to match the name of the #-tag with a fieldname inside the DataSet property, and (if found) will replace the #-tag with the current value of the field. Quite useful for a detailed view of all fields in a DataSet - if you know the fieldnames, that is.

### DataSet and PageProducer

Since the combination of a PageProducer and a DataSet is very powerful one, I will now spend the remainder of the column showing how to use a DataSet and use it to prepare a DataSetPageProducer to list all fields of the DataSet.

The result will be a routine that takes a DataSetPageProducer as its argument, and works on the DataSet property as well as the HTMLDoc property. The TDataSet which is connected to the DataSet property is analysed, and an HTML template that lists the fieldnames inside the TDataSet is dynamically produced inside the HTMLDoc property of the DataSetPageProducer. This is a great way to prepare

the DataSetPageProducer and make it a powerful generic component that can be connected to any dataset and generate HTML that lists the current record of any dataset.

The PrepareDataSetPageProducer procedure is implemented as follows:

```
procedure PrepareDataSetPageProducer
  (var DSPP: TDataSetPageProducer);
var
  i: Integer;
begin
  with DSPP do if Assigned(DataSet) then
    begin
      HTMLDoc.Clear;
      HTMLDoc.Add('<h1>' + DataSet.Name
        + '</h1><hr>');
      if not DataSet.Active then
        DataSet.Open;
      for i:=0 to
        Pred(DataSet.FieldCount) do
        HTMLDoc.Add('<br><b>' +
          DataSet.Fields[i].FieldName
          + '</b>: <#>' +
          DataSet.Fields[i].FieldName
          + '>');
    end
  end;
end;
```

Note that there are some precautions to ensure that we don't do anything if the DataSet property of the DataSetPageProducer is not assigned and, if it is, we also check to make sure it's opened. It will be opened in a moment anyway, to resolve the special #-tag fieldname values that we're generating.

Of course, inside the our web action item, we must make sure that procedure PrepareDataSetPageProducer is indeed called at the right moment (i.e. before we can call the DataSetPageProducer.Content property, which invokes the #-tag for fieldvalues replacement process). This is handled inside the OnAction event handler for any web action item as follows:

```
procedure TWebModule1.WebModule1
  WebActionItem0Action(
  Sender: TObject;
  Request: TWebRequest;
  Response: TWebResponse;
  var Handled: Boolean);
begin
  PrepareDataSetPageProducer
    (DataSetPageProducer1);
  {DataSetPageProducer1.DataSet :=
    nil;}
  Response.Content :=
    DataSetPageProducer1.Content
end;
```

The line where I set the DataSet property to nil can be used as a debug statement. The result would be that the generated content of the HTMLDoc property would not be processed any further (there is no dataset to obtain field values from) so the result of calling DataSetPageProducer1.Content is just the content of the HTMLDoc property - an easy way to check if PrepareDataSetPageProducer is working correctly.

## Generating Templates

A final, and perhaps even more flexible solution can be obtained if we take the code from the PrepareDataSetPageProducer and turn it into a GenerateDatasetTemplateHTML: using a DataSet as an argument, returning a string as result (which can be written to an external template file).

```
function GenerateDatasetTemplateHTML
  (var DataSet: TDataSet):
    String;
var
  i: Integer;
begin
  Result := '<h1>' + DataSet.Name
    + '</h1><hr>';
  if not DataSet.Active then
    DataSet.Open;
  for i:=0 to
    Pred(DataSet.FieldCount) do
    Result := Result + '<br><b>' +
      DataSet.Fields[i].FieldName
    + '</b>: <#' +
      +
      DataSet.Fields[i].FieldName
    + '>'
end;
```

This can be especially useful in a WebSnap environment, where HTML templates are pre-generated when you create a new Page Module, but can easily be extended with a little custom section (especially useful in case you're using a DataSetPageProducer inside the Page Module).

You can also use the above routine to generate a HTML string that can be written to an external file (for use in the HTMLFile property that I mentioned in the beginning of the column).

## Unit Prescribe

The total source code inside unit Prescribe contains the two routines PrepareDataSetPageProducer and GenerateDatasetTemplateHTML.

```
unit Prescribe;
interface
uses
  DB, DSProd;

procedure PrepareDataSetPageProducer
  (var DSPP:
    TDataSetPageProducer);

function GenerateDatasetTemplateHTML
  (var DataSet: TDataSet): String;

implementation

procedure PrepareDataSetPageProducer
  (var DSPP:
    TDataSetPageProducer);
```

```
var
  i: Integer;
begin
  with DSPP do if Assigned(DataSet)
    then
      begin
        HTMLDoc.Clear;
        HTMLDoc.Add('<h1>' + DataSet.Name
          + '</h1><hr>');
        if not DataSet.Active then
          DataSet.Open;
        for i:=0 to
          Pred(DataSet.FieldCount)
          do
            HTMLDoc.Add('<br><b>' +
              DataSet.Fields[i].FieldName
            + '</b>: <#' +
              DataSet.Fields[i].FieldName
            + '>')
          end
        end;

function GenerateDatasetTemplateHTML
  (var DataSet: TDataSet): String;
var
  i: Integer;
begin
  Result := '<h1>' + DataSet.Name
    + '</h1><hr>';
  if not DataSet.Active then
    DataSet.Open;
  for i:=0 to
    Pred(DataSet.FieldCount) do
    Result := Result + '<br><b>' +
      DataSet.Fields[i].FieldName
    + '</b>: <#' +
      DataSet.Fields[i].
        FieldName
    + '>'
  end;
end.
```

## Delphi, Kylix and C++Builder

The Code works with Delphi and Kylix, and even with C++Builder (if you use the final unit at the end of this article, which can be compiled with C++Builder without problems). And since Delphi, Kylix and C++Builder all support WebBroker (at this time) and WebSnap (at this time in Delphi 6, and shortly in the forthcoming Kylix 2 and C++Builder 6), the techniques are in fact general and useful at the same time!

**Bob Swart** (aka Dr.Bob - [www.drbob42.com](http://www.drbob42.com)) is an independent author, trainer and webmaster using Delphi, C++Builder and Kylix.



He lives in The Netherlands, but loves to shop in London. He's a firm believer in (and addicted to) e-business and spends lots of money on Amazon.com and other e-commerce sites.