

Developer Express - ExpressWeb Framework - Introduction

Contents

- Purpose of this Article
- Introduction
- EWF Components
 - The Menu Control
 - The Web Style Controller
 - The Web Check Box
 - The Web Popup Menu and Web Label
 - The Web DB DataSource
 - The Web DB Data Navigator and Web DB Grid
- Client Side HTML
 - The TcxHTMLLabelElement and TcxHTMLButtonElement
- HTML Tables
- Server Side Scripting
- Using existing HTML as a Template
- Appendix
 - Using the Web App Debugger
 - Deployment

Purpose of this Article

This white paper is part of a set providing detailed information about the ExpressWeb Framework technology from Developer Express. It provides an introduction to the technology by means of a simple tutorial demonstrating how to create a web application.

In order to do the tutorial yourself, you need to have a basic familiarity with Delphi and the following products installed:

- Delphi 6 OR Delphi 7 (Professional or Enterprise)
- ExpressWeb Framework V1.6 or greater

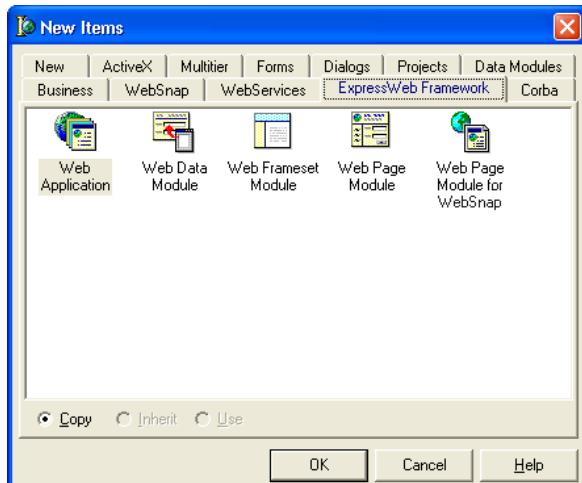
Introduction

Developer Express's ExpressWeb Framework (EWF) enables the creation of web applications. A web application is one that runs within a web browser such as Internet Explorer, Netscape Navigator and other similar platforms. This tutorial covers:

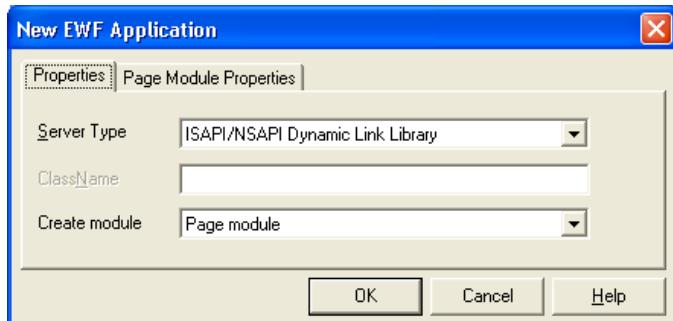
- Creating an EWF application
- Running via Web App Debugger (WAD)
- Comparison of WAD and ISAPI applications
- Form/HTML Result/Preview modes
- EWF Components
- Creating and using a WebMenu
- Adding web pages to the application
- Styles via the WebStyleController
- Popup Menus
- Database access
- Data Navigator and Grid controls
- Other data controls
- EWF HTML Elements
- HTML editing
- HTML Tag display in Object Inspector
- HTML table editing
- Client and server side scripting
- Using existing HTML as a Template

Note that each of the above items is built on the preceding ones and it is recommended that you read everything first.

First we need to create an EWF application. Start Delphi and select **File | New | Other**. Choose the ExpressWeb Framework page:



Select 'Web Application' (the default) and click [OK]. You'll see the 'New EWF Application' modal dialog box.



For the 'Server Type', select 'Web App Debugger executable'. Web App Debugger executables are designed to make debugging easier by creating executables with the server built-in. For 'ClassName' enter 'EWFIntroApp'. The ClassName field is used to identify the Web App Debugger executable so that it can be distinguished from other Web App Debugger executables. Set 'Create module' entry to 'Page Module' (the default). Then click [OK] to generate the application.

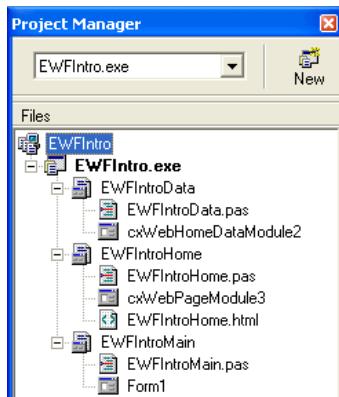
Three Units will be created:

- **Unit 1** is the main form for the Web App Debugger (WAD) executable. This form has no real purpose in a delivered web application. It is there only as a part of the WAD.
- **Unit 2** is the Home Data Module for the application and acts like the "brain". This module is used to set application-wide settings. You would use this in the same way as you might a Data Module in a standard Delphi application. You can also use the DefaultPage property to declare the home page for the application.
- **Unit 3** is the web applications main form, the initial home page. This form has an HTML document, a standard Delphi PAS file, and a standard Delphi DFM file.

Rename project parts as follows:

- **Project Group** : EWFIntro
- **Project1** : EWFIntro
- **Unit1** : EWFIntroMain
- **Unit2** : EWFIntroData
- **Unit3** : EWFIntroHome

When you have done this, your Project Manager (View | Project Manager or Ctrl+Alt+F11) should look something like this:

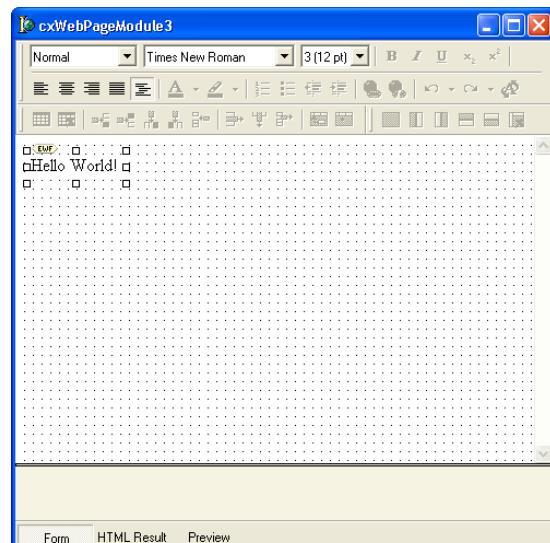


Select the EWFIntroHome unit. In Delphi's VCL, select the [EWF Controls] tab, then place a cxWebLabel on the form in the usual way. Using the object inspector, set the 'Caption' property to 'Hello World!'

Press F9 to run the application in your default browser. You have completed your first web application using EWF. (Note: If this is the first time that you have run an application under the WAD, you may find that your app errors on start. If it does, just shut down the Web App Debugger and re-start your application).

Notice that when your application runs two items are shown: The default browser and the main window form for your application. The latter is used to terminate your web application but is otherwise of little use. Therefore, at design time make the form as small as possible and place it out of the way.

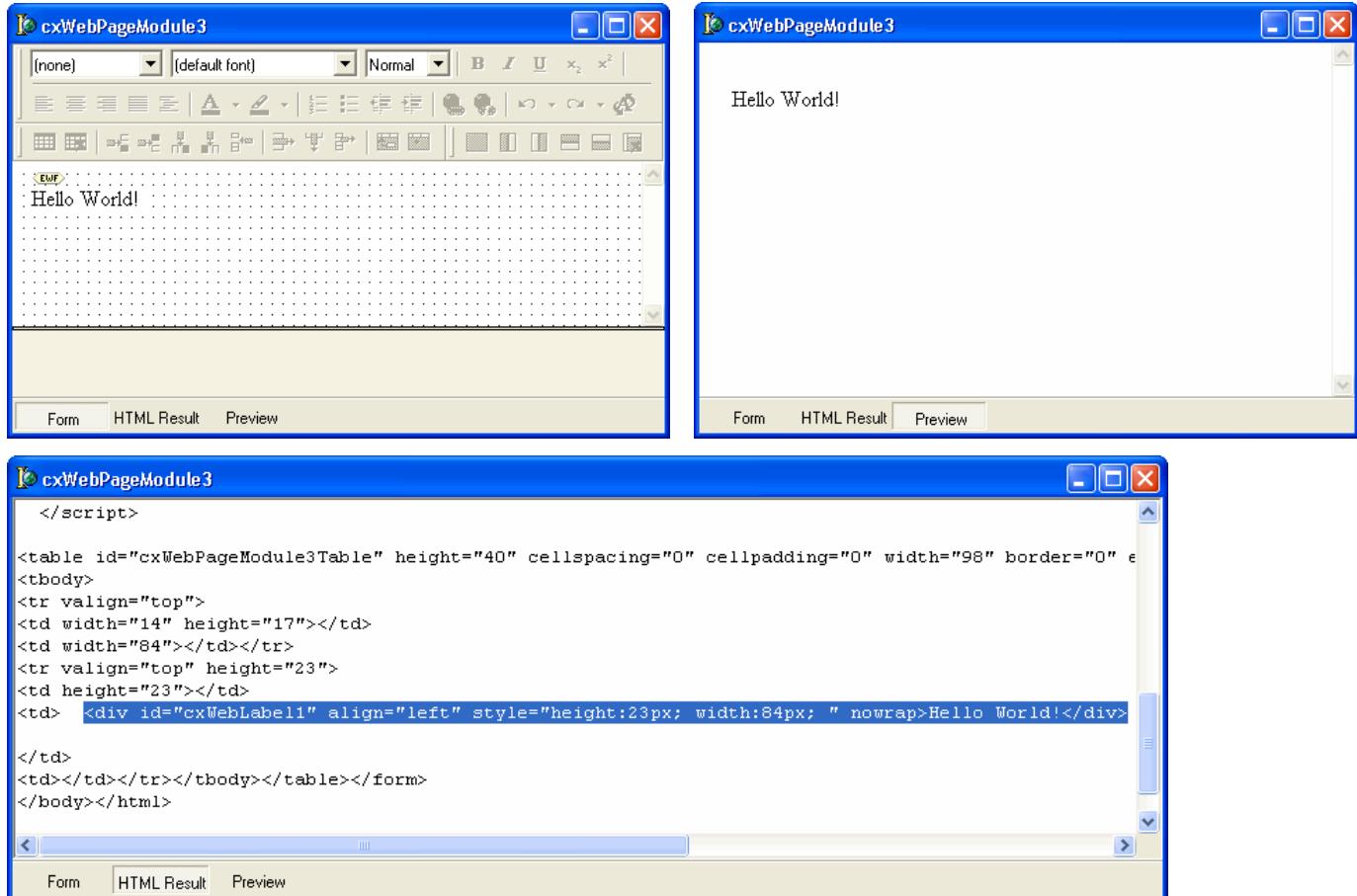
Review the EWFIntroHome form. It differs from the standard Delphi design time form in a few ways. One way is the buttons along the base labeled 'Form', 'HTML Result' and 'Preview'.



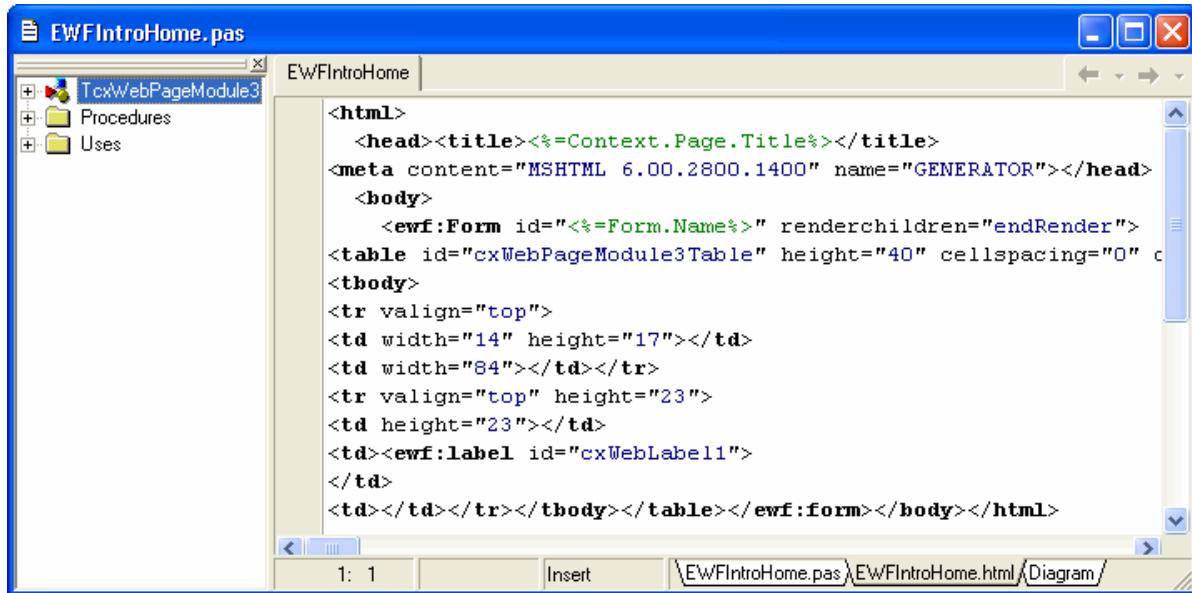
The 'Form' button (selected by default) enables you to place components on the form and manipulate them like any other Delphi component. The data from this view is written to a file called EWFIntroHome.html. This file contains some HTML code with EWF meta tags.

The 'HTML Result' button will show you the HTML code produced as a result of your design. To do this, it expands the ewf tags (special HTML tags, prefixed with "ewf:") held in the EWFIntroHome.html file.

The 'Preview' button will show an HTML preview of your design. Server side code (discussed shortly), is not executed. The three views are shown below.



Examine the Project Manager. Under EWFIntroHome, it shows 3 units: EWFIntroHome.pas (a standard pascal unit), cxWebPageModule3 (a standard Delphi form unit) and EWFIntroHome.html, the EWF HTML template unit. When opened in Delphi, the file looks like the following figure:



Compare this with the 'HTML Result' view to see how EWF meta tags are expanded. Note that the 'HTML Result' view is read only. To actually edit the HTML, you would need to edit the EWFIntroHome.html file.

Another way that the EWFIntroHome form differs from the standard Delphi design time form, is the WYSIWYG editing toolbar. You use this in conjunction with components that you place on the form.

EWF Components

Four pages of components are supplied by default:



EWF Controls: standard Delphi non-data components (server side) that provide normal Delphi event handlers (Object Pascal)



EWF Data Controls: standard Delphi data-aware components



EWF Components: non-visual controls and data sources

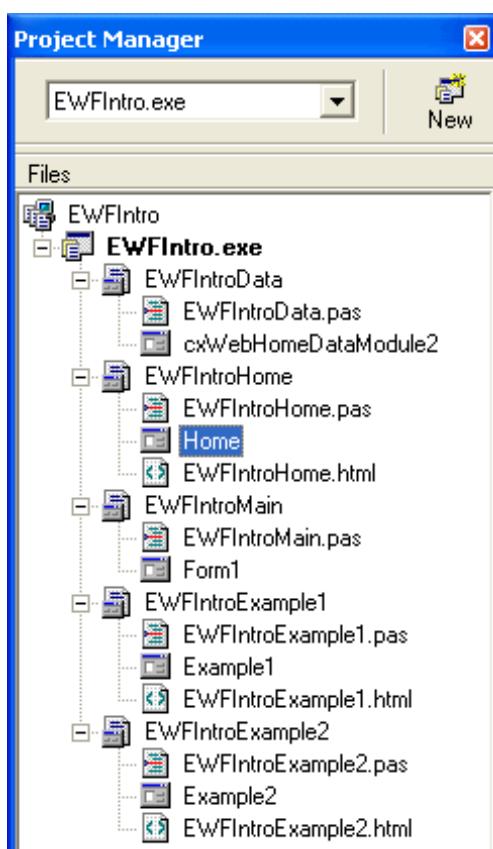


EWF HTML Elements: for adding client side HTML code to the html file.

To continue with the tutorial, select the 'Hello World' label and delete it.

Add two new pages to the EWF application by clicking on **File | New | Other**. Select the ExpressWeb Framework page and select 'Web Page Module'. Do this twice and name the units **EWFIintroExample1** and **EWFIintroExample2** respectively.

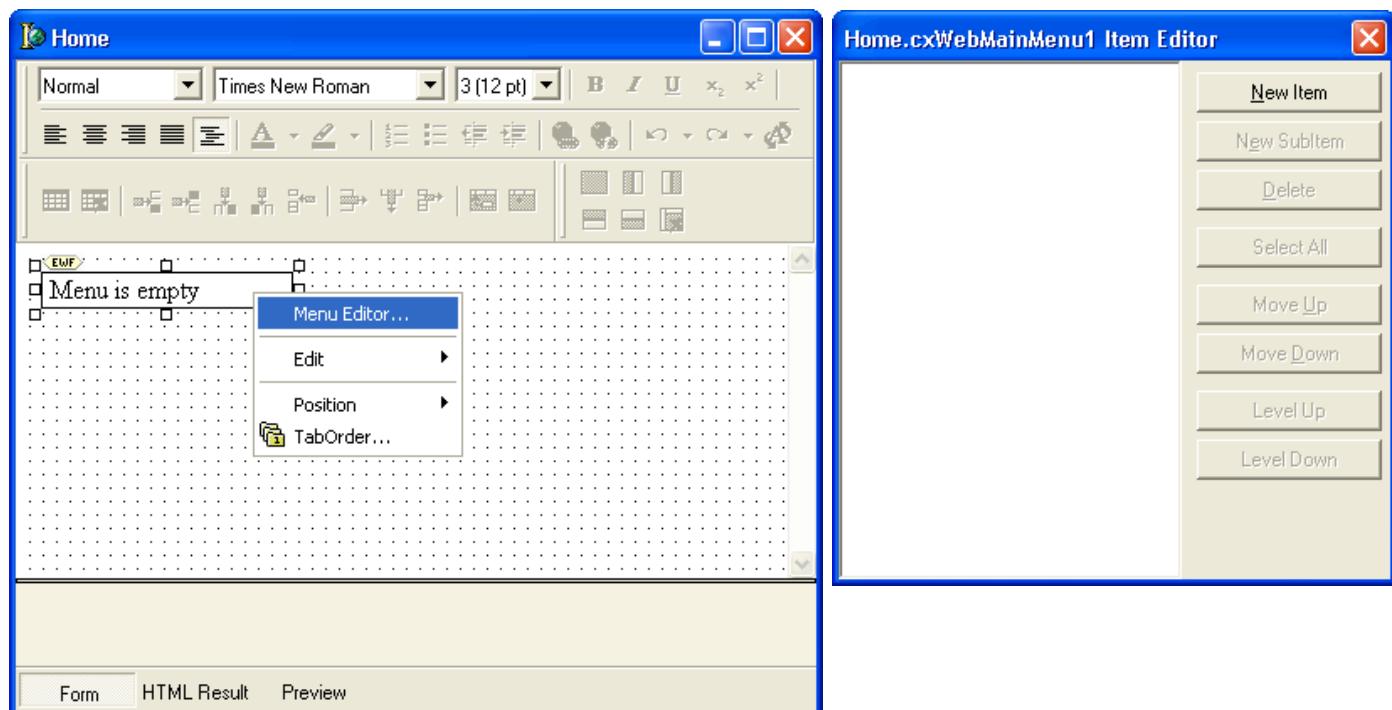
Also, change the form names for EWFIntroHome, EWFIntroExample1 and EWFIntroExample2 to Home, Example1 and Example2 respectively. This is important as we'll be using these names to access those forms from the menu.



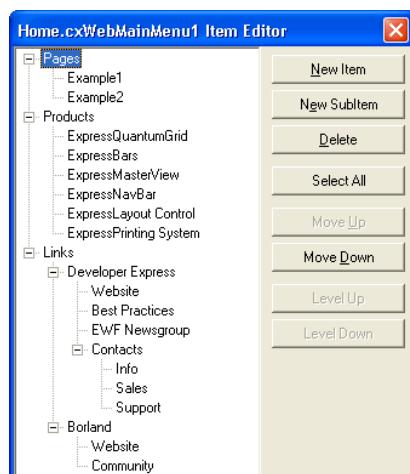


The Menu Control

Select the 'EWF Controls' page on the Component Palette and place a TcxWebMainMenu on to the Home form. Note that the menu component has an EWF flag attached to it. This indicates that it is an EWF component rather than an HTML element. Right click on the menu and select 'Menu Editor'.



Enter some items and sub-items to create a menu system. To follow the tutorial, match the entries in the screenshot below. To change the menu's caption, enter the value into the Caption property in the Object Inspector. To change the width, enter a value for that property also. Sub-items inherit their width from the item above them.



To make the menu system useful, we need to enter an URL for each menu option. This URL can be in standard HTML format (ie: <http://www.devexpress.com>), or you can simply enter the page name for a page in your EWF application (ie: Example2).

Revisit the menu items and populate the URL -> Href property as follows:

Menu Item	URL->HREF Value
Example 1	Example1
Example 2	Example2
ExpressQuantumGrid	http://www.devexpress.com/?section=/Products/VCL/ExQuantumGrid
ExpressBars	http://www.devexpress.com/?section=/Products/VCL/ExBars
ExpressMasterView	http://www.devexpress.com/?section=/Products/VCL/ExMasterView
ExpressNavBar	http://www.devexpress.com/?section=/Products/VCL/ExNavBar
ExpressLayout Control	http://www.devexpress.com/?section=/Products/VCL/ExLayoutControl
ExpressPrinting System	http://www.devexpress.com/?section=/Products/VCL/ExPrintingSystem
Website	http://www.devexpress.com
Best Practices	http://www.devexpress.com/?section=/BestPractices
EWF Newsgroup	news://news.devexpress.com/devexpress.public.vcl.expresswebframework
Info	mailto://info@devexpress.com
Sales	mailto://clientservices@devexpress.com
Support	mailto://support@devexpress.com
Website	http://www.borland.com
Community	http://community.borland.com

Note that all that is needed to refer to a page within the application is the name of the form for that page. (Tip: Under the WAD, the address to these pages must appear in the search path for the WAD so that it can find and load them correctly)



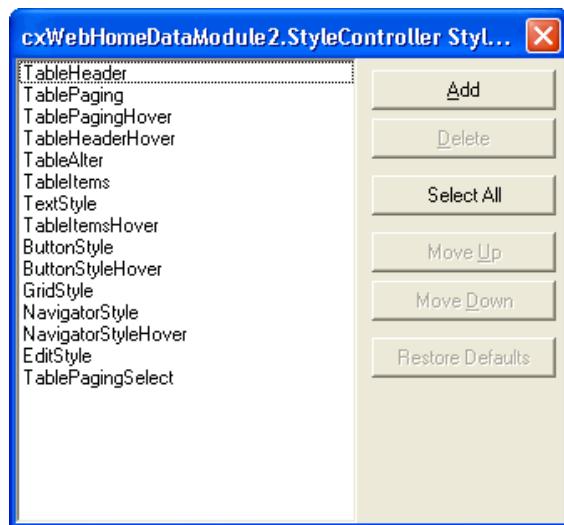
The Web Style Controller

Web developers will be aware of Cascading Style Sheets (CSS). CSS enables a unified look and feel to be applied to a site. Developer Express users will be aware of Style Controllers. The Web Style Controllers allow a unified look and feel to be applied across EWF Controls used in an ExpressWeb application.

The TcxWebStyleController component (found on the EWF tab in the Component Palette) is a non-visual component. It can be placed in one of three locations:

1. On the form (in the special non-visual area just above the bottom status bar)
2. On the EWFIntroData WebHomeDataModule
3. On a new WebDataModule

The third option is recommended for the re-use of data between different applications (i.e. by adding the module to the Repository). For this tutorial, use the second option and add the component to the EWFIntroData unit, to allow the styles to be available to all the pages in the application:



To make it available to all the pages in the application, switch to each page and use Delphi's menu: [File | Use Unit](#).

To add styles to the cxWebStyleController, double click on it or right click and select 'Styles Editor'. Alternatively, rather than enter a lot of items, you may want to copy a style controller from one of the EWF demos. This tutorial uses the style controller found in the ExpressSupportForum demo, the HomeDM unit.

Next, select the Home page for the application and the menu on it. Set the Styles->Default and Styles->Hover properties as shown.





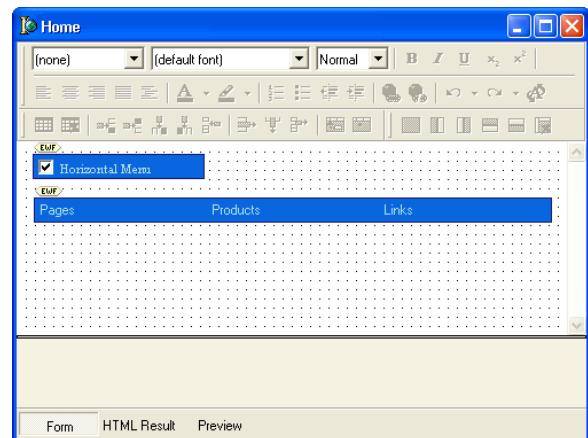
The Web Check Box

- Drop a TcxWebCheckBox instance on to the Home form (EWF Controls Page)
- Set cxWebCheckBox1.Caption to 'Horizontal Menu'
- Set cxWebCheckBox1.Checked to True
- Set cxWebCheckBox1.Styles.Default to cxWebHomeDataModule2.ButtonStyle
- Set cxWebCheckBox1.Styles.Hover to cxWebHomeDataModule2.ButtonStyleHover
- Double-click on cxWebCheckBox1 and enter the following code:

```
cxWebMainMenu1.Horizontal := cxWebCheckBox1.Checked;
```

Running the app via F9 predictably shows the effect of clicking on the check box control.

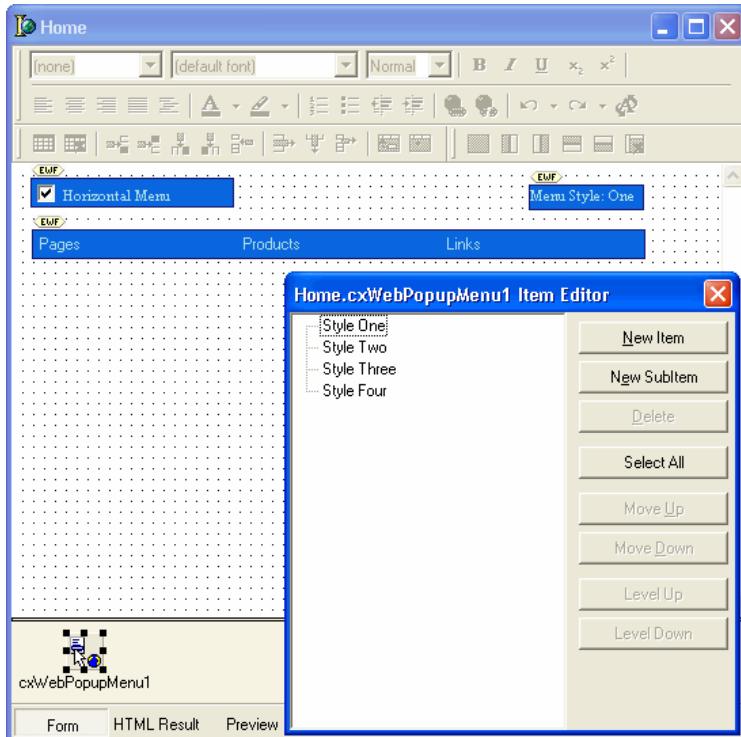
Use the menu system to go to either the Example1 or Example2 page. Hit the browser's return button and you'll see the check box is still in its checked or unchecked state as you last left it. This is an example of maintaining state. This is handled for you automatically by EWF.



The Web Popup Menu and TcxWebLabel

Drop a TcxWebLabel (EWF Controls page) and a TcxWebPopupMenu (EWF Components page) onto the Home page of the EWF application. Set the following properties:

- cxWebLabel1.Caption to 'Menu Style: One'
- cxWebLabel1.Styles.Default to cxWebHomeDataModule2.ButtonStyle
- cxWebLabel1.Styles.Hover to cxWebHomeDataModule2.ButtonStyleHover
- cxWebLabel1.PopupMenu to cxWebPopupMenu1
- cxWebPopupMenu1.Styles.Default to cxWebHomeDataModule2.ButtonStyle
- cxWebPopupMenu1.Styles.Hover to cxWebHomeDataModule2.ButtonStyleHover



Double-click cxWebPopupMenu1 to invoke the Item Editor and add four items as shown above.

In order to simplify the generic event handler for changing the menu style, set the Tag properties of the four cxWebPopupMenuItems to 1, 2, 3 and 4 respectively.

Hook the four items to a common event handler (multi-select the items in the Item Editor then double click on the OnClick event in the Object Inspector). Rename the event handler to WebPopupMenu1ItemClick. Code the event as follows and then run the application to check the effect.

```

procedure THome.WebPopupMenu1ItemClick(Sender: TObject);
begin
  case (Sender as TcxWebMenuItem).Tag of
    1: begin
      cxWebLabel1.Caption      := 'Menu Style : One';
      cxWebMainMenu1.Styles.Default := cxWebHomeDataModule2.ButtonStyle;
      cxWebMainMenu1.Styles.Hover  := cxWebHomeDataModule2.ButtonStyleHover;
    end;
    2: begin
      cxWebLabel1.Caption      := 'Menu Style : Two';
      cxWebMainMenu1.Styles.Default := cxWebHomeDataModule2.TablePaging;
      cxWebMainMenu1.Styles.Hover  := cxWebHomeDataModule2.TablePagingHover;
    end;
    3: begin
      cxWebLabel1.Caption      := 'Menu Style : Three';
      cxWebMainMenu1.Styles.Default := cxWebHomeDataModule2.TableItems;
      cxWebMainMenu1.Styles.Hover  := cxWebHomeDataModule2.TableItemsHover;
    end;
    4: begin
      cxWebLabel1.Caption      := 'Menu Style : Four';
      cxWebMainMenu1.Styles.Default := cxWebHomeDataModule2.TableHeader;
      cxWebMainMenu1.Styles.Hover  := cxWebHomeDataModule2.TableHeaderHover;
    end;
  end;
end;

cxWebLabel1.Styles.Default      := cxWebMainMenu1.Styles.Default;
cxWebLabel1.Styles.Hover        := cxWebMainMenu1.Styles.Hover;
cxWebPopupMenu1.Styles.Default := cxWebMainMenu1.Styles.Default;
cxWebPopupMenu1.Styles.Hover   := cxWebMainMenu1.Styles.Hover;
end;

```

At run-time, the menu will popup whenever the mouse is over the label. This is because cxWebLabel1.PopupShowingType is set as the default action for stMouseOver. You can change this to stClick if desired.

Note: The menu dropdown is a client event, while clicking on one of the menu options generates a server refresh, the latter requiring a round-trip to the server to re-create the page.



The Web DB DataSource

Using the application page Example1, drop a standard Delphi TTable (from the BDE page) and an EWF TcxWebDBDataSource onto the form. (Note that the BDE is not thread-safe without a TSession component.)

Connect the TTable to the DBDEMONS alias and select the table biolife.db. Make the TTable active. Add persistent fields to the TTable and set the cxWebDBDataSource.DataSet property to Table1. It is the cxWebDBDataSource that makes EWF data aware

in this instance. There are two other datasources that could have been used: cxStdWebDataSource and cxWebDataSource and there is often confusion over which should be used under any given circumstance. The guidelines are to start with the 'thinnest' component that provides the least functionality and then use a progressively more functional component as required. In order:

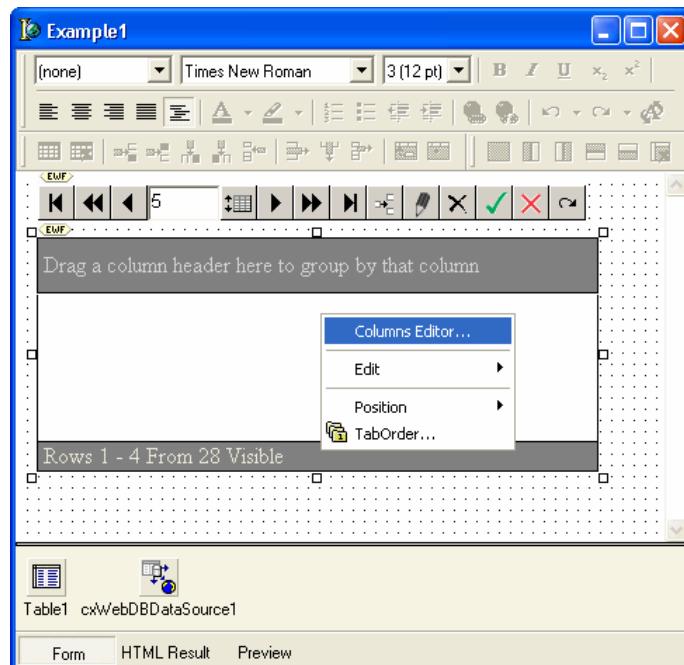
- TcxWebDataSource (non data-aware. Use with in-memory data)
- If you require a connection to a data-aware component, then use TcxStdWebDataSource (data aware component. Use with TDataSet derivatives)
- If you intend to group or sort dataset data, then use TcxWebDBDataSource (data-aware with extended functions)



The Web DB Data Navigator and Web DB Grid

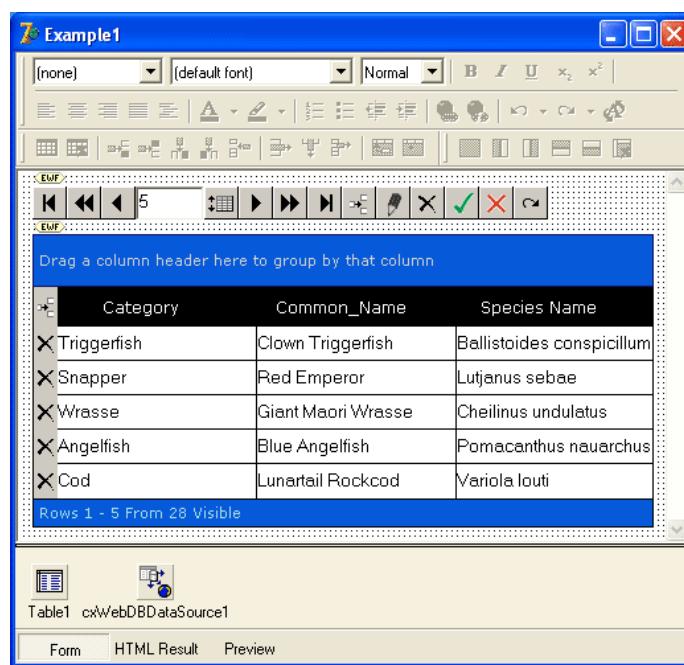
Add a cxWebDBNavigator and cxWebDBGrid (both on the 'EWF Data Controls' page in the VCL) to your Example1 page. Connect the navigator and grid to the datasource using the DataBinding>DataSource property.

Right click on the grid and select 'Columns Editor' from the popup menu. Select the fields Category, Common Name and Species Name.



Run the application and see the effect. Note that you can sort, group and edit data using these components.

Adjust some of the style properties. Recall that this page can utilize the StyleController we introduced earlier on in the exercise. The following screenshot gives an example of what can be done. Also, try using some of the other data-aware controls provided with EWF. They work in the same way as a standard Delphi application.



Client Side HTML

Select the Example2 page and, using the Object Inspector, set the PositioningType property to cxptAbsolute.



The TcxHTMLLabelElement and TcxHTMLButtonElement

Add a TcxHTMLLabelElement and a TcxHTMLButtonElement (both available on the EWF HTML Elements page) to the form.

Select the HTML Label and it will appear like this:

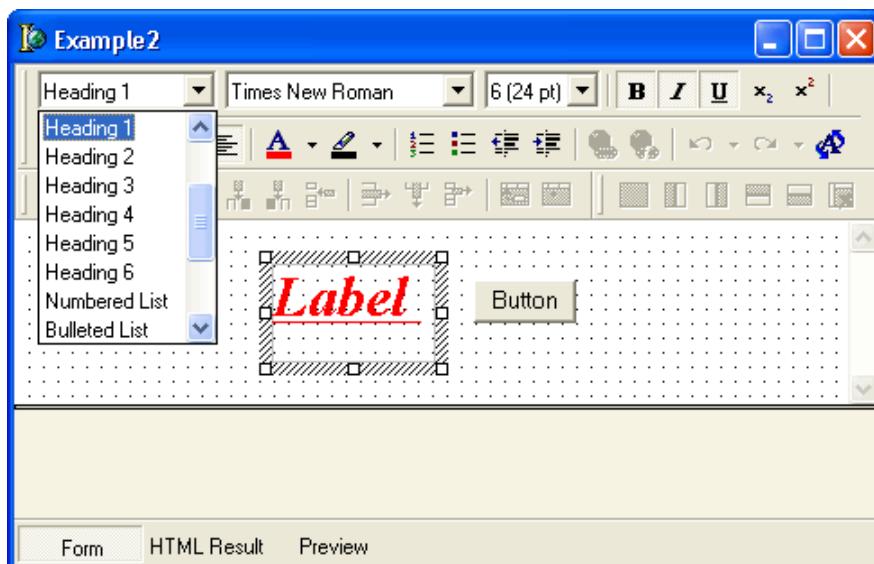


This mode of selection allows you to move and resize the component. Select the HTML label again to enable editing by using the WYSIWYG toolbars.

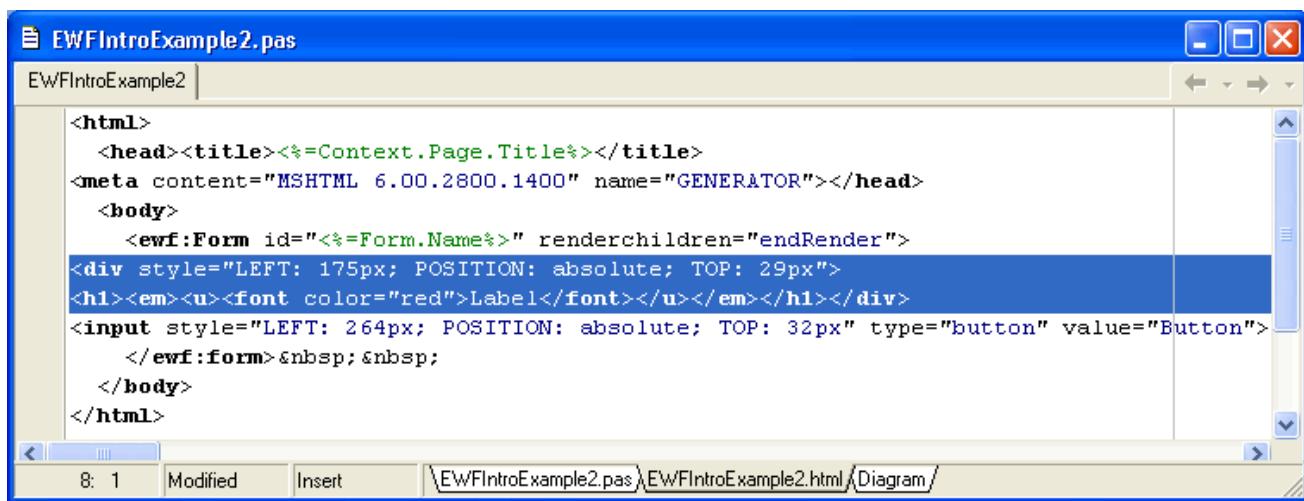
Select the HTML Label again and it will now look like this:



At this stage, use the WYSIWYG toolbar and set the text to 'Heading 1'. Select Italic, Bold, and, Underline and change the font's color to red.



When done, examine the file EWFIntroExample2.html. The line referring to the label just amended is highlighted in the following screenshot:



Try changing one of the values in this code. For example, change the color "red" to "blue". Note that the label on the form does not change straight away. However, click on the label and the new color will appear. Note also that there are no EWF meta-tags in use. This is because we are using HTML Elements, which are distinct from EWF Elements (Components, Controls and Data Controls). This fact is further indicated by the lack of the ewf flag on the component. For more information about HTML elements, visit the World Wide Web Consortium (W3C) at <http://www.w3.org/MarkUp/>.

With the HTML Label selected, examine the Object Inspector. You will immediately notice that the properties are all lowercase - whereas Delphi component properties are usually ProperCase. This is another indication that we are dealing with HTML elements.

HTML Elements and their properties will be easily understood by those who have coded with (D)HTML before. This tutorial does not cover such topics but there are many good books available on it.

The Style property expands to a very large range of sub-options. Many of those options can also be set using the WYSIWYG toolbar.

Coding events using EWF HTML Elements is done by entering Javascript directly into the code. Select the code page EWFIntroExample2.html. In there, you'll find a line similar to this:

```
<input style="LEFT: 264px; POSITION: absolute; TOP: 32px" type="button" value="Button">
```

(Note: Some parts are EWF meta tags that will be expanded at run-time but the remainder is pure HTML. Note also that this line was wrapped for the purposes of this tutorial and will probably appear as a single line in your code)

Change the line so that it reads like this (by adding the highlighted part):

```
<input style="LEFT: 264px; POSITION: absolute; TOP: 32px" type="button" value="Button"
      onclick="alert('oops')">
```

What we've done is to add some JavaScript to the HTML. Run the application. When you click the button on example page 2, you should see a modal dialog pop up with the phrase 'oops' in it.

JavaScript is not covered in this tutorial but to get the best out of your EWF application you will require some knowledge of it. An excellent reference book is Danny Goodman's "JavaScript Bible" (ISBN: 0-7645-3342-8)

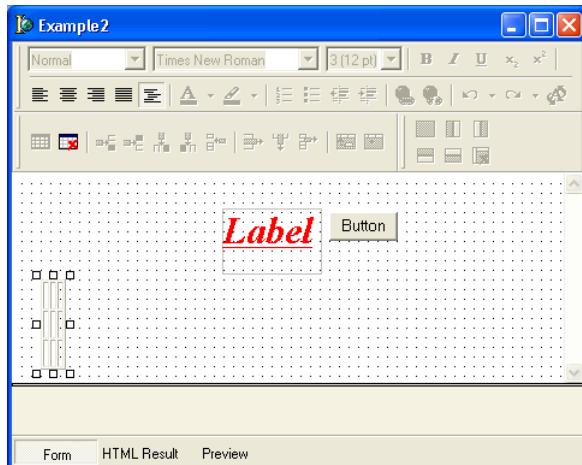
When using JavaScript, you'll find the above example very limiting and you'll want the ability to be able to call JavaScript functions. The following code snippet shows exactly how to do that. Although it performs the same action, it is easy to see how this could be extended. The code that has been added to the original is highlighted:

```
<html>
  <head><title><%=Context.Page.Title%></title>
  <meta content="MSHTML 6.00.2800.1400" name="GENERATOR"></head>
  <script language="JavaScript">
    <!-- Script Begin
      function MyButtonClick(event)
      {
        alert("oops");
      }
    // Script End --></script>
  <body>
    <ewf:Form id="<%=Form.Name%>" renderchildren="endRender">
      <div style="LEFT: 175px; POSITION: absolute; TOP: 29px">
        <h1><em><u><font color="red">Label</font></u></em></h1>
      </div>
      <input style="LEFT: 264px; POSITION: absolute; TOP: 32px" type="button" value="Button"
            onclick="MyButtonClick(event)">
    </ewf:Form>&nbsp;&nbsp;
  </body>
</html>
```

HTML Tables

At this stage, HTML tables have been used for positioning controls. This is done automatically as the default value for the form's PositioningType is cxptGrid. What this means is that the components you place on a form have a table underlying them to ensure (as far as possible) that the components remain in the same place when the form is reproduced at run-time (note that, because of the disparity in available browsers, differences in presentation are to be expected).

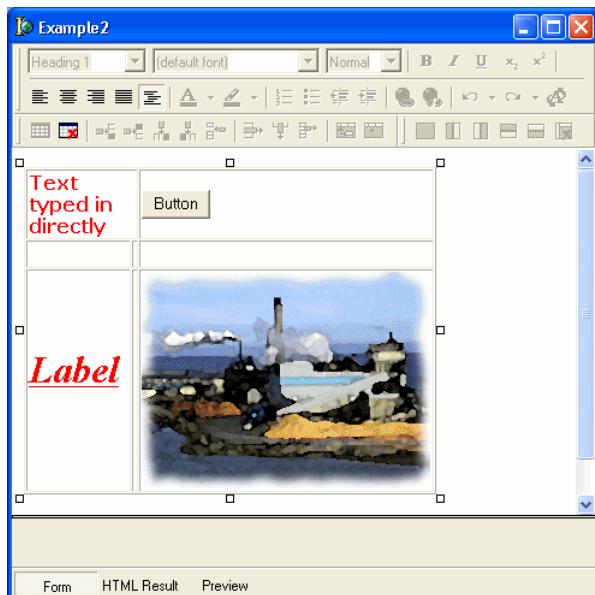
HTML tables may also be added to the form just like any other component. Select the cxHTMLTableElement  from the EWF HTML Elements page of the VCL and place it on the Example2 page.



Stretch the table over the width of the page. Place the cursor in the top left cell of the table. While holding down the Shift key, press the right arrow key once. This selects the adjacent cell as well (note that there is no visual feedback given on the form) and the table cell merge button on the Example2 form  becomes available. Click this button to merge the two cells. With the cursor positioned in the merged cells, type in some text. Use the text formatting toolbar on the Example2 form to format the text you just entered.

It would be convenient to be able to place components directly into the table's cells, but if you try it, you'll see that the controls seem to float above the grid and will render themselves strangely at run-time dependent upon your browser. To gain the required functionality, set the page's PositioningType to cxptFlow. Now, select the button on the page and drag and drop it into the top right-hand cell of the table. Repeat the exercise with the label on the form and drop it in the lower left table cell (**Note:** The only way to select the text control is to select all of the text on it first and then drag it). Check the result using HTML Preview and then run the application to see how your browser renders the result.

Next, select a cxHTMLImageElement control  from the EWF HTML Elements page. Attempt to place it in the lower right cell of the table. Note that the control will move and appear after the table. You will need to use drag and drop to position it correctly. Do this and then set the src property of the cxHTMLImageElement to point to a graphic. This property's default value is set to the location of your EWF application. For this tutorial, copy a graphic into that location and append the graphics name to the property. The tutorial uses a standard, Borland supplied, splash graphic.



Note that the HTML table will resize itself, its rows and columns, as items are manipulated within it. When you have finished your design work, use the table controls to set the table as you wish it to appear at run-time. Note, however, that the only way to change a table's row and column sizes is by editing the HTML.

Finally, return the form's PositioningType property to cxptAbsolute. Note that the formatting just set for the table is retained.

Server Side Scripting

Throughout the tutorial so far, mention has been made of EWF meta tags and scripting. We have also covered EWF components and HTML components. It is also possible to produce pages by writing all the script and meta tags yourself. The ability of EWF to access Delphi objects and properties via scripts at run-time is hugely useful in this regard.

Add another WebPageModule to the application and call it EWFIntroExample3 (change the form's name to Example3). Add a TTable component from the BDE page of the VCL and set the properties to hook this table to the DBDEMONS BDE alias that is supplied by default with Delphi installs (select the events.db table from there).

Open the source code file EWFIntroExample3.html and rewrite it to match the following:

```
<html>
  <head>
    <title><%=Context.Page.Title%></title>
    <meta content="MSHTML 6.00.2800.1400" name="GENERATOR">
  </head>
  <body>
    <ewf:Form id="<%=Form.Name%>" renderchildren="endRender">
      <%while (!Table1.EOF) { %>
      <TABLE cellSpacing=0 cellPadding=2 border=1>
        <TR>
          <TD width="250" colSpan="2"><%=Table1.FindField("event_name").DisplayText %></TD>
          <TD width="400" rowspan="3"><%=Table1.FindField("event_description").AsString %></TD>
          <TD width="200" rowspan="3"><Image src=<%=Form.CurrentImageSource%>></TD>
        </TR>
        <TR>
          <TD width=100><%= Table1.FindField("event_date").DisplayText %></TD>
          <TD width=150><%= Table1.FindField("event_time").DisplayText %></TD>
        </TR>
        <TR>
          <TD width="205" colSpan="2"><%= Table1.FindField("ticket_price").DisplayText %></TD>
        </TR>
      </TABLE>
      <% Table1.Next; %>
    </ewf:form>&nbsp;
  </body>
</html>
```

The code should be fairly simple to follow but the thing to note is that we're utilizing the properties and methods of Table1 (the BDE TTable component) in our HTML code. Before being able to use this functionality, you must add the unit cxScriptDBImpl to the implementation uses clause of the EWFIntroExample3.pas source code file. By default, all published component properties are available via script, however it is necessary to add the cxScriptDBImpl unit to the uses clause to make public properties and methods available via script. While in this unit, code to open and close the TTable component in the forms OnActivate and OnDeactivate events respectively. Finally, write a function to return an image as a URL for use on the form. The relevant parts for the pas unit follow:

```
unit EWFIntroExample3;
---
type
  TExample3 = class(TcxWebPageModule)
    Table1: TTable;
    procedure cxWebPageModuleActivate(Sender: TObject);
    procedure cxWebPageModuleDeactivate(Sender: TObject);
  private
    function GetCurrentImageSource: string;
  published
    property CurrentImageSource: string read GetCurrentImageSource;
  end;
---
implementation
{$R *.DFM} {*.html}

uses
  WebReq, WebCntxt, cxWebModFact, cxWebScript, Variants, cxWebDataUtils, Graphics, cxScriptDBImpl;

function Example3: TExample3;
begin
  Result := TExample3(WebContext.FindModuleClass(TExample3));
end;
```

```

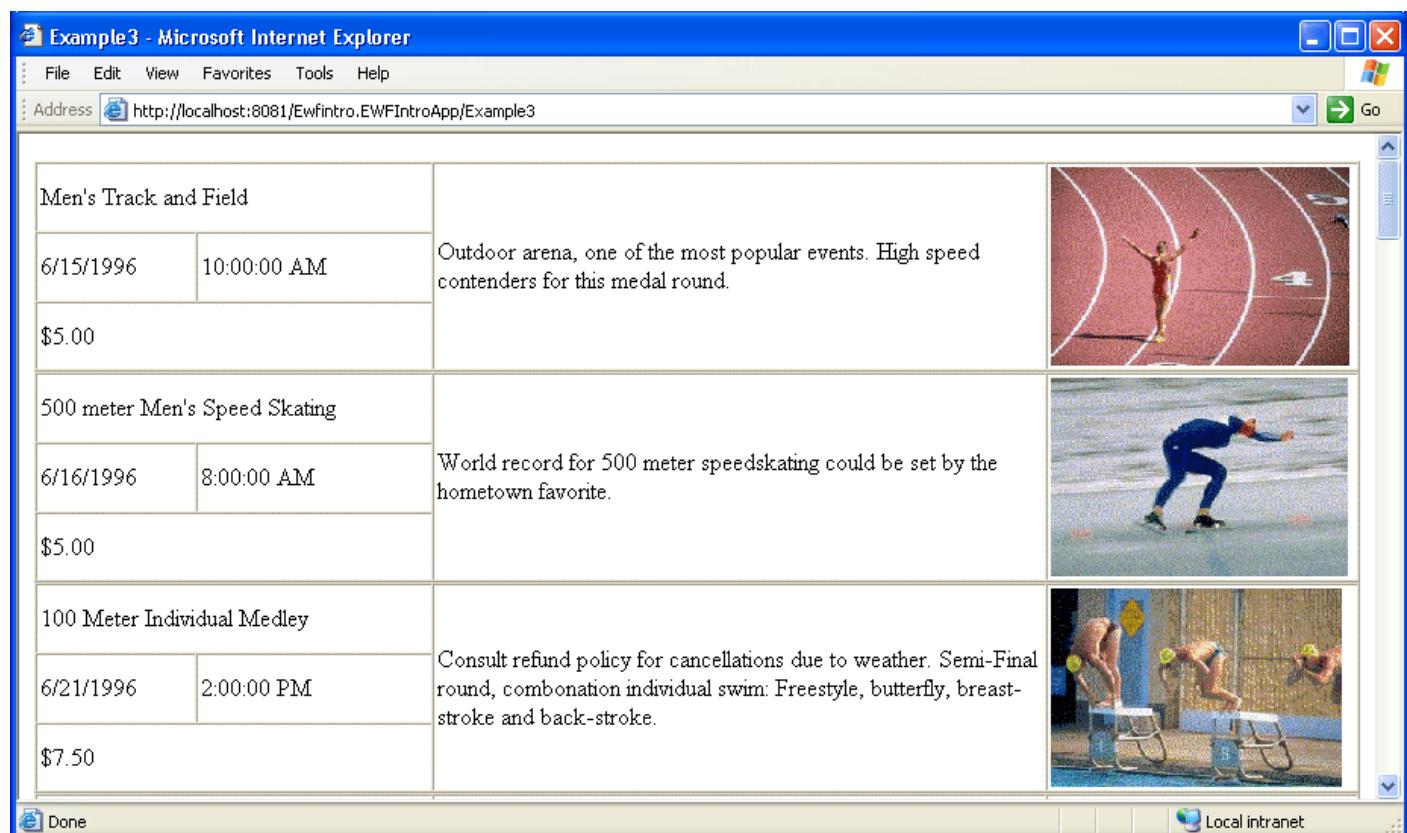
procedure TExample3.cxWebPageModuleActivate(Sender: TObject);
begin
  Table1.Open;
end;

procedure TExample3.cxWebPageModuleDeactivate(Sender: TObject);
begin
  Table1.Close;
end;

function TExample3.GetCurrentImageSource: string;
var
  AWidth, AHeight: integer;
begin
  LoadImage( Table1.FindField('Event_Photo').Value, TBitmap, Result, AWidth, AHeight );
end;

```

Coded successfully, the final page should be rendered to the browser as follows:



The screenshot shows a Microsoft Internet Explorer window titled "Example3 - Microsoft Internet Explorer". The address bar contains the URL "http://localhost:8081/Ewfintro.EWFIntroApp/Example3". The main content is a table with three rows, each representing an event with a photo and a description.

Men's Track and Field		
6/15/1996	10:00:00 AM	Outdoor arena, one of the most popular events. High speed contenders for this medal round. 
\$5.00		
500 meter Men's Speed Skating		
6/16/1996	8:00:00 AM	World record for 500 meter speedskating could be set by the hometown favorite. 
\$5.00		
100 Meter Individual Medley		
6/21/1996	2:00:00 PM	Consult refund policy for cancellations due to weather. Semi-Final round, combination individual swim: Freestyle, butterfly, breast-stroke and back-stroke. 
\$7.50		

To access your newly added page, either enter the URL directly at run-time when testing your app, or add a link to it in the menu system as discussed earlier.

Using existing HTML as a Template

For a tool like EWF to be truly useful, there has to be a simple way of importing an existing web page to use as an html template and that is what we are going to show you next.

The page chosen as an example is the EWF controls page on our website: <http://www.devexpress.com/?section=/products/vcl/ewf/Components>, which is sufficiently complex to prove that EWF can handle real pages and not just simple examples.

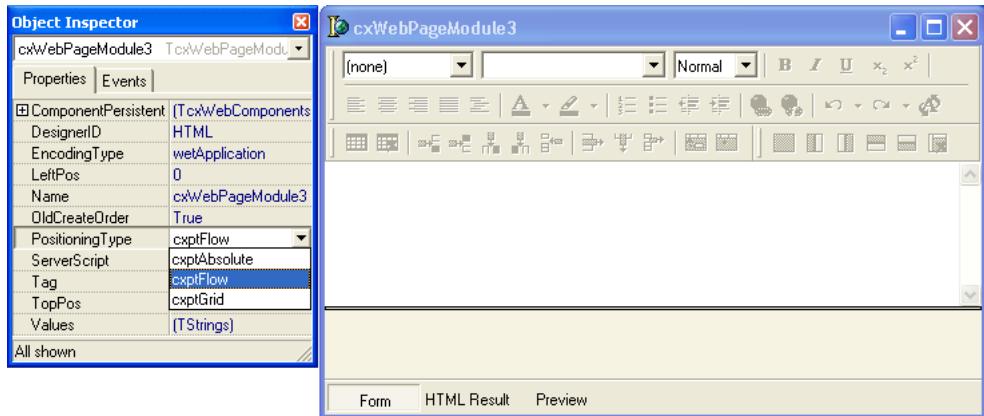


To import this page into your Delphi project (it is assumed that a new project named EWFImport has already been created) you should first save the page to the project's folder. Note that when saving complete web pages, IE5 saves all the associated images and stylesheets in a sub-directory:

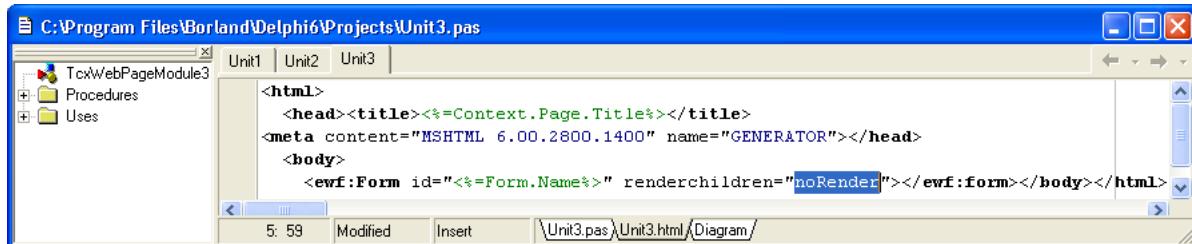


Having saved the web page, we can now turn our attention to the home page of our newly created EWF application (EWFImport).

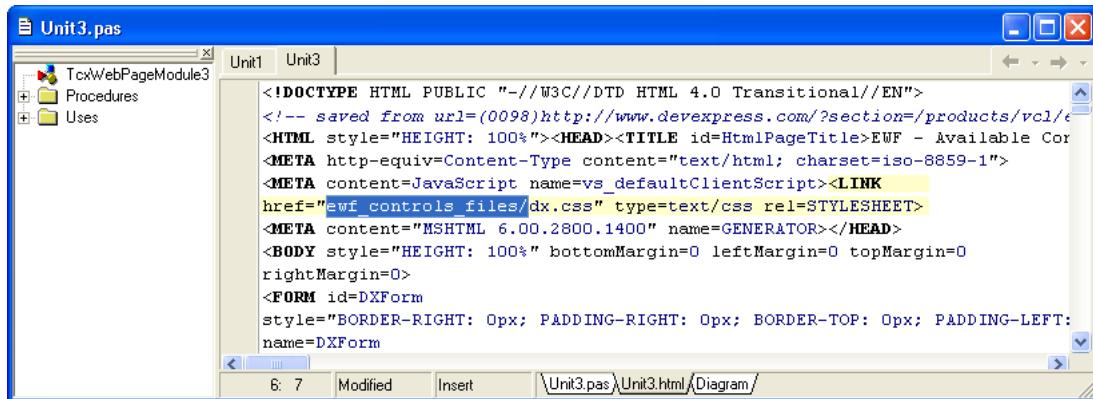
The home page's settings as represented by the cxWebPageModule3 form should look like the screenshots below:



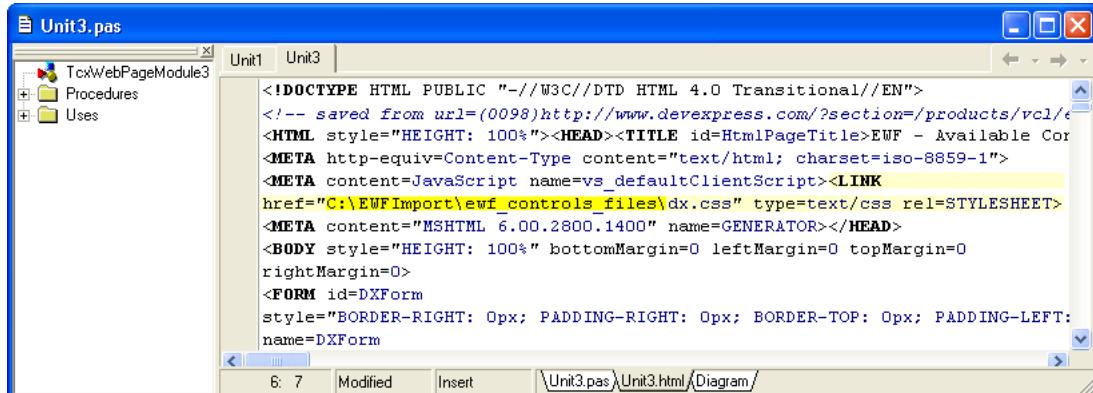
At this point, Unit3.html looks like this:



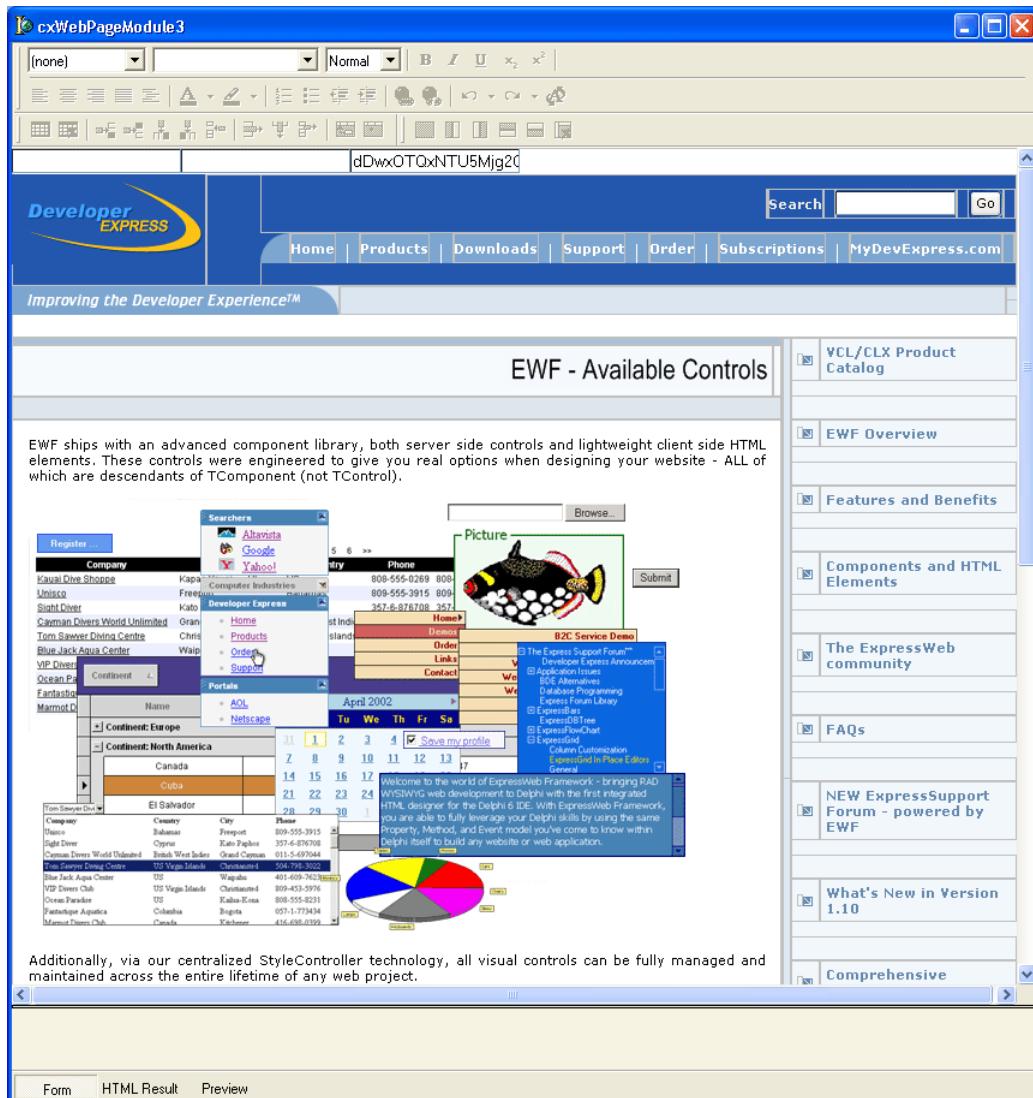
Now comes the interesting bit. We replace the contents of unit3.html by pasting in the contents of the product_catalog.htm file. An alternative is to use an external file manager and copy soapbox.html to unit3.html.



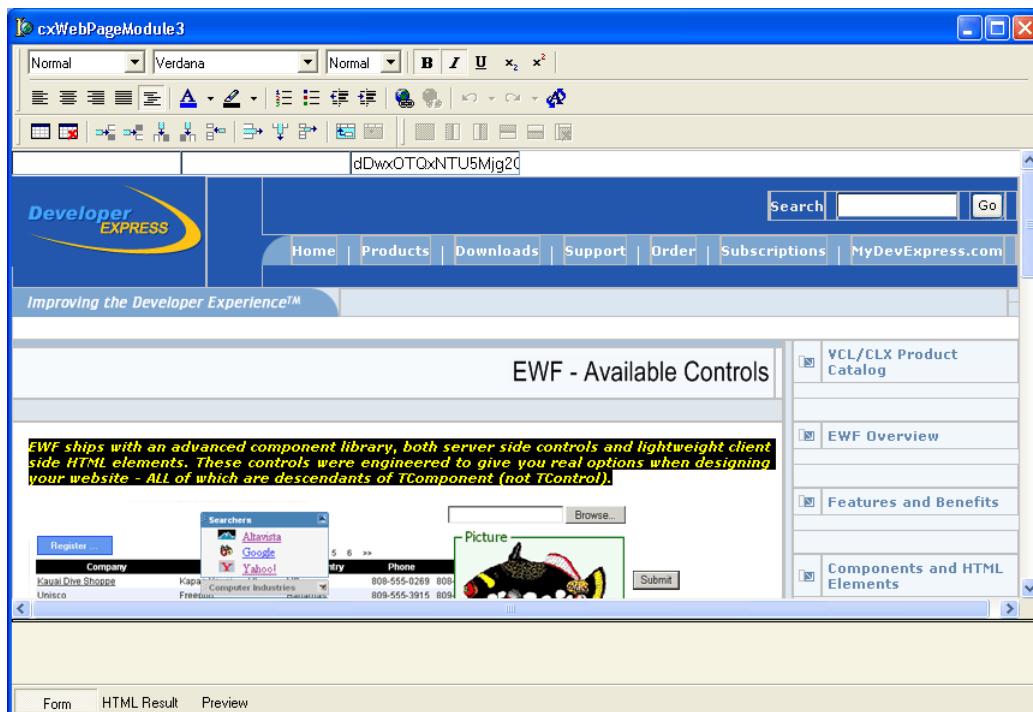
Next, we examined the contents of the unit3.htm file and the only problem relates to the highlighted lines below: The highlighted line is currently referencing an external style sheet by a relative address. Just for the purpose of design time style visualization, we merely change the root so that it represents the absolute path to the style sheet file:



Having saved the above changes, we can now turn our attention to the form designer which should now look like the image below:

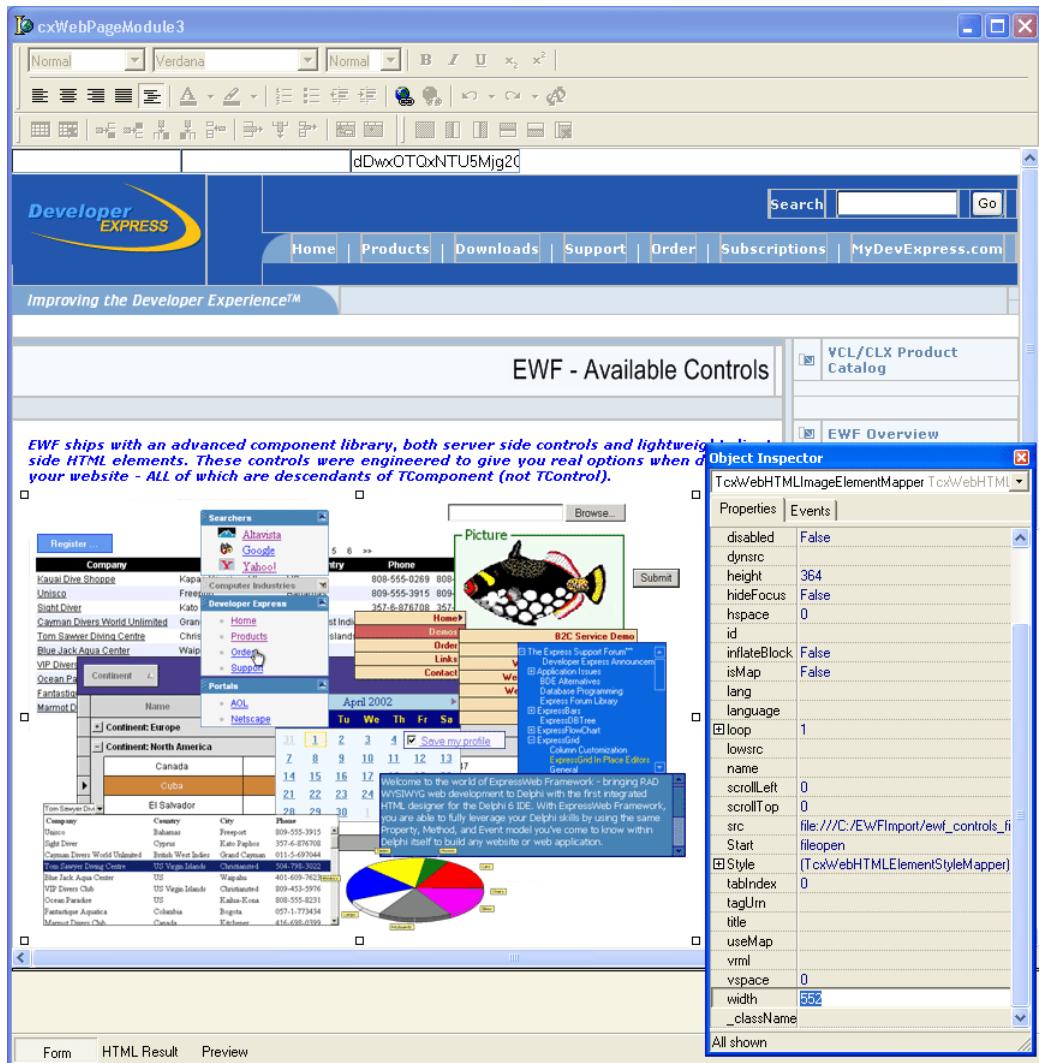


You can now modify and format the web page contents using the form designer's toolbars. The image below demonstrates how you can format the first paragraph of text.



Still, this is not all the form designer offers you. In exactly the same way for the imported page as when you create your own page using EWF HTML Elements, you can select elements, rearrange them using drag and drop and modify their attributes via the Object Inspector.

The following screenshot shows how to access the attributes of the image element.



There are five 'RAD' ways of modifying a page:

- dropping an EWF component on it (i.e. basically server side)
- dropping an EWF HTML Element on it (adds an html tag to the template)
- move text or controls (component, element or existing html tag) by drag/drop
- modify selected text by overtype or by means of the editing toolbars
- modify controls (previously dropped or template html tag) via the Object Inspector

In addition to the above, you also have complete freedom to modify the html template manually. You can create or copy controls there and the changes you make will be seen as soon as you switch back to Form mode. We mentioned earlier that one of our targets was to let you work as you wish. The synchronization we provide allows you to pass the html file to a graphic designer before, in the middle and after other work is done on the page.

In the Designer paper (which is available on our web site), a similar example to the one above is taken a bit further by showing how we modified the page to add another item to the left hand side menu followed by the creation of another page with a similar style.

Well, that is all for this introductory tutorial. As you can imagine there is much more to show you than we can manage here. Have a look at our other papers for more details. In particular, look out for the examples which use all the various components and HTML Elements we supply.

Appendix

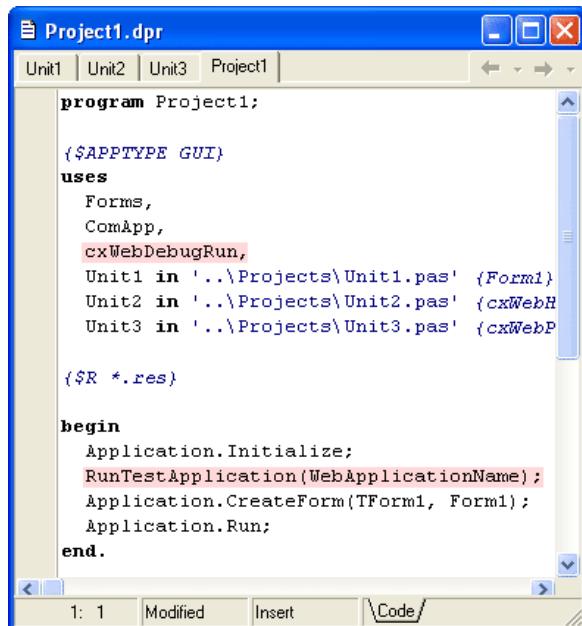
This appendix will consider the points related to debugging and deploying web applications created using EWF. Note that deploying an application (transferring the created and tested modules to a real web application) is only needed when the application type doesn't support debugging. This is so for all application types but ISAPI applications. When developing an ISAPI application you can make use of the ExpressWeb Debugger. Read the 'Debugging ISAPI Applications with WWSDebug' document for details on setting up the tool. This document can be found in the WWSDebug folder of your EWF installation.

When developing applications other than ISAPI, you will have to use the Web App Debugger tool integrated into the Delphi environment. To use it, you need to create the 'Web App Debugger Executable' project. The first section of the appendix describes how to work with such projects. After the application is built and tested, you need to create a real web application and transfer all the created modules to it. This process is described in the second section of the appendix.

Using the Web App Debugger

EWF has encapsulated the steps required to run via the Web App Debugger (WAD) so that you merely have to press **F9**, or select **Run | Run** from the IDE menu just the same as if you are running a standard Win32 application.

Take a look at the source of an EWF application running under WAD:



```
program Project1;

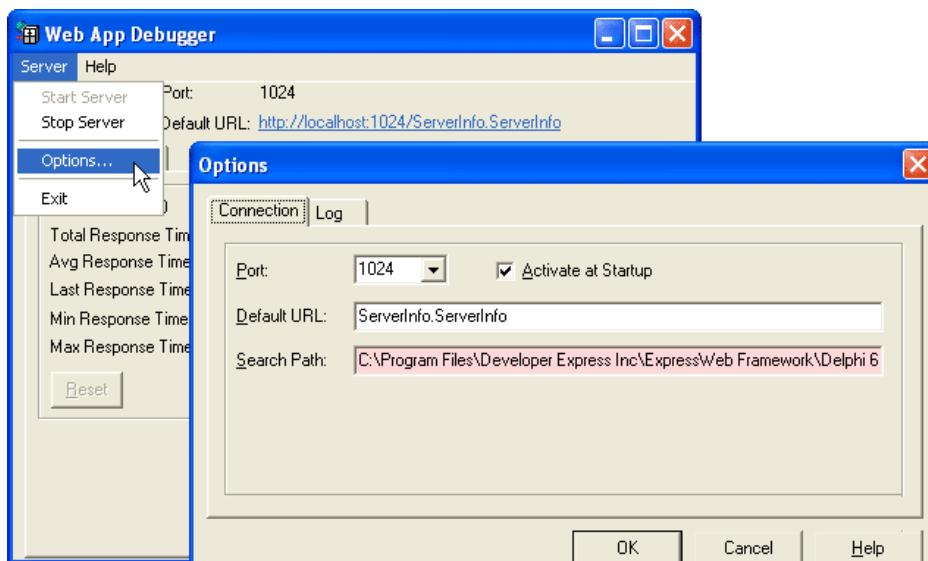
{$APPTYPE GUI}
uses
  Forms,
  ComApp,
  cxWebDebugRun,
  Unit1 in '..\Projects\Unit1.pas' {Form1},
  Unit2 in '..\Projects\Unit2.pas' {cxWebH},
  Unit3 in '..\Projects\Unit3.pas' {cxWebP};

{$R *.res}

begin
  Application.Initialize;
  RunTestApplication(WebApplicationName);
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

If we remove the two highlighted lines, this functionality is removed and we revert to the classical multi-step WAD invocation. But as can be seen there are no reasons for wanting to do that.

However, there are a couple of things worth mentioning here. First, for scripts and images to be available, their paths do have to be available to WAD. This is achieved by switching to the WAD application (its main window is opened automatically when you run the EWF application or can be invoked at design-time using the **Tools | Web App Debugger** menu item):

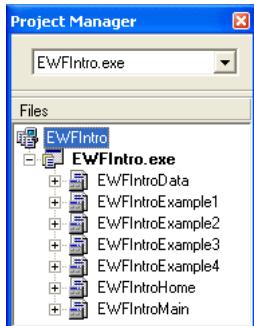


Deployment

After you have created and tested your web application, you will obviously want to deploy it to a real web server. To do this simply create a new application of the appropriate type (Apache, CGI, etc) and add all the units that you have worked on to the new project. Compile, and deploy appropriately for your target web server.

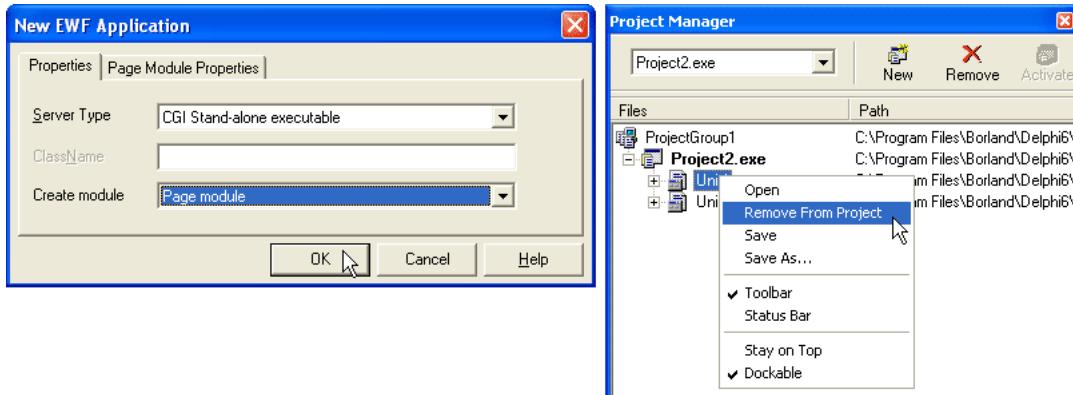
It is a good idea to create a project group that contains a different project for each type of deployment that you may make with your application: ISAPI, CGI, Apache, etc.

Below is a brief example of how to do this. Consider the files used by the tutorial:

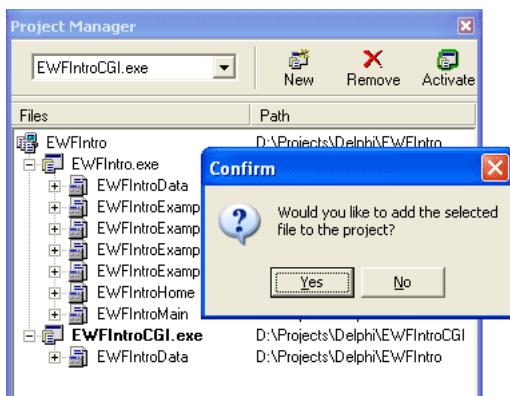


The EWFIntroMain form shown above is specifically there to support the Web App Debugger, but all the others are needed by the new version of our application.

Create a new EWF application via [File | New | Other | EWF](#) page and choose the 'CGI Stand-alone executable' project type. When the application is created, remove all automatically created modules from it:



Rename the project to EWFIntroCGI and save it. After that, reopen the EWFIntro project group and add the EWFIntroCGI project to it. Then, simply move all the files needed using drag and drop. In the screenshot below, EWFIntroData has already been copied and the drag and drop of EWFIntroHome is almost complete.



Note: only one copy of each file actually exists. Further development can take place via EWFIntro and then a quick compile of EWFIntroCGI is all that is necessary when you are ready to deploy.

Once you have compiled the CGI application, you can upload it to your local web server. For details on how to do this, refer to the Deployment section of the EWF help file.