Mixing ExtJS webapplications and Pascal: ExtPascal

Michaël Van Canneyt

September 17, 2009

Abstract

There are many Javascript libraries to create webapplications. ExtJS is such a library. A new and interesting project now allows to create ExtJS webapplications by transforming Pascal code written in Delphi or Lazarus to javascript code: An introduction to Extpascal.

1 Introduction

Writing a webapplication isn't terribly hard these days. All that is needed is a webserver, a bunch of HTML and javascript files, and some server mechanism to answer AJAX calls. There are meanwhile plenty of Javascript libraries that contain a full featured set of widgets (controls) and a mechanism to code AJAX calls: an expensive term for what is basically simply an RPC call using HTTP as a protocol and SOAP or JSON as the encoding of the call. Qooxdoo, Dojo, zapatec, jQuery are all examples of such a library. ExtJS is also one. It can be obtained from

`http://www.extjs.com/`

ExtJS is dual-licensed: There is an open-source license for open source applications, but there is a commercial license for commercial applications (on a per-developer basis). It is very full-featured, has an active community, and great ambitions with it's recent release of version 3.0.

The server mechanism - to produce Javascript webpages or answer Ajax calls, cannot be written in Javascript. Thus, it must be programmed using a traditional computer language, meaning that each webapplication is usually written using 2 languages. (rare exceptions include Morfik and Google's GWT).

For Pascal programmers, the idea of having to write Javascript isn't a particular appealing one. Fortunately, this may be a thing of the past: Extpascal takes care of producing Javascript from pascal code: complete windows can be constructed in 100% object pascal code, Ajax calls are also coded in pascal code, and there is limited functionality for creating Javascript event handlers also from pascal code. ExtPascal works both in Delphi and Free Pascal (or lazarus) and can be downloaded from

`http://code.google.com/p/extpascal/`

2 Installation of ExtPascal

ExtPascal is a recent development, and installation is still a bit rough: it's more than extracting a simple zip file. To install ExtPascal, one must first install ExtJS. It suffices to download the open source version from the ExtJS website, and to extract the zip file somewhere.

After this, ExtPascal can be downloaded from it's website: the current version is 0.9.6. For some cutting-edge developments, the subversion repository can also be reached: instructions are available on the ExtPascal website.

After extracting the ExtPascal zip file, some further installation steps are necessary. The ExtPascal classes are auto-generated from the ExtJS documentation: the ExtJS documentation is generated from the ExtJS sources by some automated tool, which creates structured text which can be easily parsed by a pascal program.

The distribution comes with a ExtToPascal.dpr project file which contains the conversion program. The project should be compiled and then run on the command-line, with as the sole argument the name of the directory where the ExtJS documentation is located. By default, the documentation is in the

```
ext-3.0.0/docs/output
```

Directory.

If the ExtPascal code was extracted in a directory extpascal next to the ext-3.0.0 directory, this means that the command

```
ExtToPascal ../ext-3.0.0/docs/output
```

will produce the Extpascal classes. (on Windows, the slashes must be replaced with backslashes). The output of this program starts like this:

```
ExtToPascal ../ext-3.0.0/docs/output
ExtToPascal - ExtJS docs to Pascal units wrapper, version 0.9.8
(c) 2008-2009 by Wanderlan Santos dos Anjos, BSD license
http://extpascal.googlecode.com/

Reading ExtJS HTML files...
  > ../ext-3.0.0/docs/output//Ext.ux.form.DateTime.html
  > ../ext-3.0.0/docs/output//Ext.ux.layout.RowLayout.html
```

The output finishes with the following messages:

```
  > ExtUxGrid.pas
  > ExtAppUser.pas
284 ExtJS classes wrapped.
21 unit files generated.
39.450 seconds elapsed.
Done! Press Enter.
```

Pressing the enter key will finish the program. The above output shows that 21 units and 284 ExtJS class wrapper have been generated.

After this, one can in theory start programming ExtPascal programs: one just needs to add the installation directory to the compilers unit search path for the project. As this is rather cumbersome, the author has created a Lazarus package (lazextpascal) that can be

compiled. The benefit is that it is then sufficient to add a dependency on the lazextpascal package to the project: The Lazarus IDE will then automatically add the correct directory to the compiler unit search path.

With this, installation is almost finished. Any ExtPascal application will produce HTML code that refers to the extjs javascript files. It does this by emitting `script` tags of the form:

```
<script src="/ext/adapter/ext/ext-base.js"></script>
<script src="/ext/ext-all.js"></script>
```

So the webserver should be able to serve the extjs source files from the /ext location. In Apache, the easiest way to achieve this is to add an `Alias` directive:

```
 Alias /ext/ "/data/source/ext-3.0.0/"
```

Obviously, the path should be changed to the actual directory where extJS was extracted. As an alternative, the extjs files can be copied to the documentroot directory of the website. It is also possible to set the paths for the script tags in the extpascal code - but more about this later.

3 Hello World in extpascal

All applications in ExtPascal currently start with a class that must descend from `TExtThread`. It must have at least 1 published method called 'Home'. This method must produce the web page that will be shown by default when the user accesses the webapplication. The following unit contains such a class:

```
unit ethelloworld;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, ExtPascal, ExtPascalUtils, Ext, ExtGlobal,
  ExtData, ExtForm, ExtGrid, ExtUtil, ExtAir, ExtDd, ExtLayout,
  ExtMenu, ExtState, ExtTree;

type
  THelloWorld = class(TExtThread)
  published
    procedure Home; override;
  end;

implementation

procedure THelloWorld.Home;

Var
  Form : TExtWindow;

begin
```

```
  Form:=TExtWindow.Create;
  With Form do
    try
      Title:='Hello, World !';
      Width:=380;
      Height:= 140;
      Closable := True;
      Html:='<center><b>Hello, World!</b></center> ';
      Show;
    Finally
      Free;
    end;
end;

end.
```

This unit is quite simple. The uses clause contains most of the ExtPascal units, so a all needed classes are available without having to look for the correct unit to add.

The `THelloWorld` class contains just one method, `Home`. The implementation of this method does not look very surprising: It creates a window, sets some properties, and then calls the `Show` method: Code that could be found in any desktop application. The only strange thing is the `Free` method at the end: in a desktop application, this would cause the window to dissapear at once. Not so in ExtPascal, and this will be explained later on.

The window class is `TExtWindow`. It corresponds to `TForm` in a Delphi or Lazarus application. It stems from the Javascript class `Ext.Window` : The naming of the pascal classes is easily derived from the Javascript class name: strip the namespace dot, and prepend with a T. Thus the `Ext.Panel` class would become `TExtPanel`.

The properties are mostly self-explaining, only `Closable` might need some explanation: if the 'Closable' property is true, then ExtJS will generate and display a 'Close' icon on the window border, which can be used to close the window.

The `HTML` property can contain arbitrary HTMl that will be shown in the window. It is not mandatory to specify this property, but for the "Hello, world" example, this is appropriate.

Obviously, there are much more properties in TExtWindow than explained here: since the documentation of ExtJs is used to generate the classes, all available properties are well-documented.

4 Using FastCGI

When this class is coded, the rest of the application can be coded as well. By default, ExtPascal applications are FastCGI applications. ExtPascal comes with a class that takes care of all the details of a FastCGI application. Basically, this application will listen on a TCP/IP socket for incoming requests, and send a response back over the socket. It is the task of the webserver to redirect the browser request to the FastCGI application.

To create a fastCGI application, a console application must be created and filled with code. The following program code is sufficient:

```
program helloworld;

{$APPTYPE CONSOLE}
```

```
uses
  ExtPascal, ExtPascalUtils, SysUtils, Math,FCGIApp, ethelloworld;

{$IFDEF WINDOWS}{$R helloworld.rc}{$ENDIF}

begin
  Application:=TFCGIApplication.Create('Hello World in ExtPascal',
                                       THelloWorld,2015);
  Application.Run;
end.
```

FastCGI applications are console applications, hence the `{$APPTYPE}` directive. The FCGIApp unit - which comes with Extpascal - contains a `TFCGIApplication` class, which handles all communication with the webserver. The `Application` variable is also defined in this unit.

The first line of the application creates a `TFCGIApplication` instance, and passes it the program title, the descendent of the `TExtThread` class to use when handling requests (THelloWorld in the case of the demo program), and finally the TCP/IP port on which to listen for requests from the webserver (2015 in the above example). After the instance is created, the run method is invoked, which will start listening for requests.

After compiling this, the application can be run. When it is run, apparently nothing will happen - which is correct because it runs in the background.

Once the fastcgi application is up and running, the webserver must be told where it is. This can be done in Apache with the `FastCgiExternalServer` directive:

```
FastCgiExternalServer /var/www/HelloWorld -host 127.0.0.1:2015
```

Assuming that the DocumentRoot of the webserver is /var/www, this directive tells the Apache server that all requests that start with /HelloWorld should be forwarded to the FastCGI server listening at port 2015 on the local machine.

After restarting the apache server, the ExtPascal application should be ready to run. Testing this can be done by entering the following URL in a web-browser:

```
http://localhost/HelloWorld
```

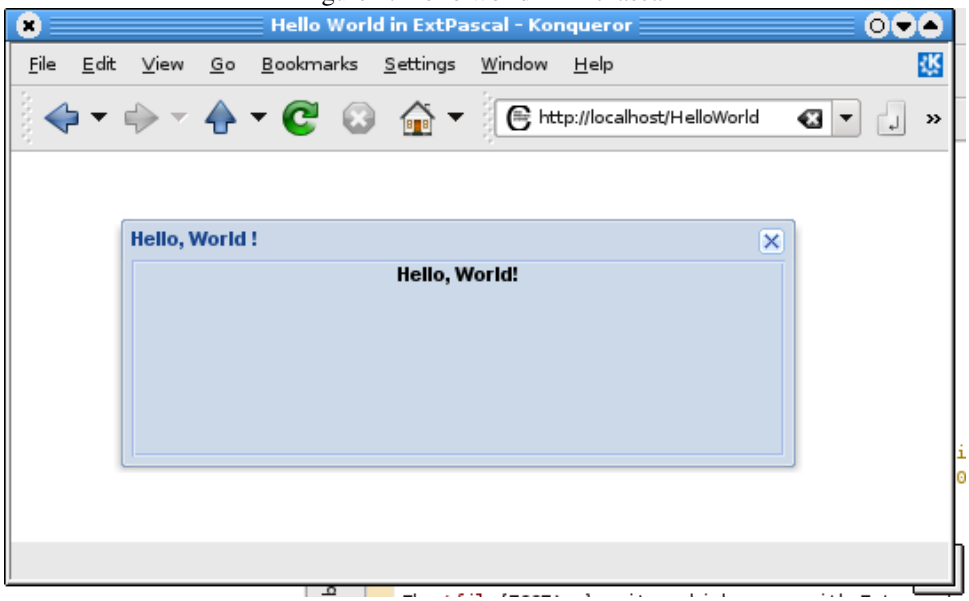
If all went well, the browser should display a window like in figure 1 on page 6.

5 Architecture: Self-Translation

How did all this work ? When the fastCGI application gets a request from the web server, it checks the path. If it finds no path information after the `HelloWorld`, it will invoke the default method of the `THelloWorld` class: `Home`. The Home method then produces the JavaScript which will be sent to the browser - it is wrapped with some standard HTML, which is normally sufficient for most applications. The javascript is produced with a process named 'Self-Translation' by the ExtPascal author: As each method of a ExtPascal class is invoked, or a property is set, Javascript is generated and appended to the response that will be sent to the browser: if no methods are invoked or properties are set, no Javascrip is generated. To show this, it is instructive to examine the JavaScript that was produced by the above code (it can be made visible using the 'Show Document Source' context menu in each browser window):

```
O0=new Ext.Window({ title:"Hello, World !",
```

Figure 1: Hello world in ExtPascal




```
  width:380,
  height:140,
  closable:true,
  html:"<center><b>Hello, World!</b></center> "});
O0.nm="O0";
O0.show("");}
```

The first line is generated by the `TExtWindow.Create` statement. All following statements are generated by the various assignments to properties, and finally the form.Show statement is translated to the final javascript statement. The name `O0` (capital O, number 0) is generated automatically by extpascal. These statements are generated as a response to the statements in pascal code: if there is no pascal code, no Javascript is generated. An easy way to check this is reverse two assignment statements in pascal; The resulting Javascript will also have the properties exchanged.

6 Simple event handlers

The above is of course nice to produce GUI elements: windows, buttons, panels, checkboxes and so on. With the above techniques, complex GUIs can be produced, without a single line of Javascript, all in Pascal. But it still does not allow for serious interaction or business logic. ExtPascal has a solution for this too.

It can produce Javascript code for event handlers in much the same way as it produces code to set properties. Consider the following example:

```
Resourcestring
  SButtonPressed = 'You pressed the button to show an alert';

procedure TMyButtonDemo.Home;
```

```
var
  ShowConfig : TExtShowConfig;
  F : TExtWindow;
  B : TExtButton;

begin
  F:=TExtWindow.Create;
  With F do
    Try
      Title:='Message Box Dialog';
      Width:=300;
      Height:=200;
      Plain:=True;
      Frame:=true;
      Layout:=lyAbsolute;
      Closable:=False;
      B:=TExtButton.AddTo(Items);
      with B do
        begin
        X:=100;
        y:=10;
        Text:='Show Alert';
        Handler:=ExtMessageBox.Alert('Confirmed',
                                     SButtonPressed,
                                     Nil);
        end;
      Show;
    Finally
      Free;
    end;
end;
```

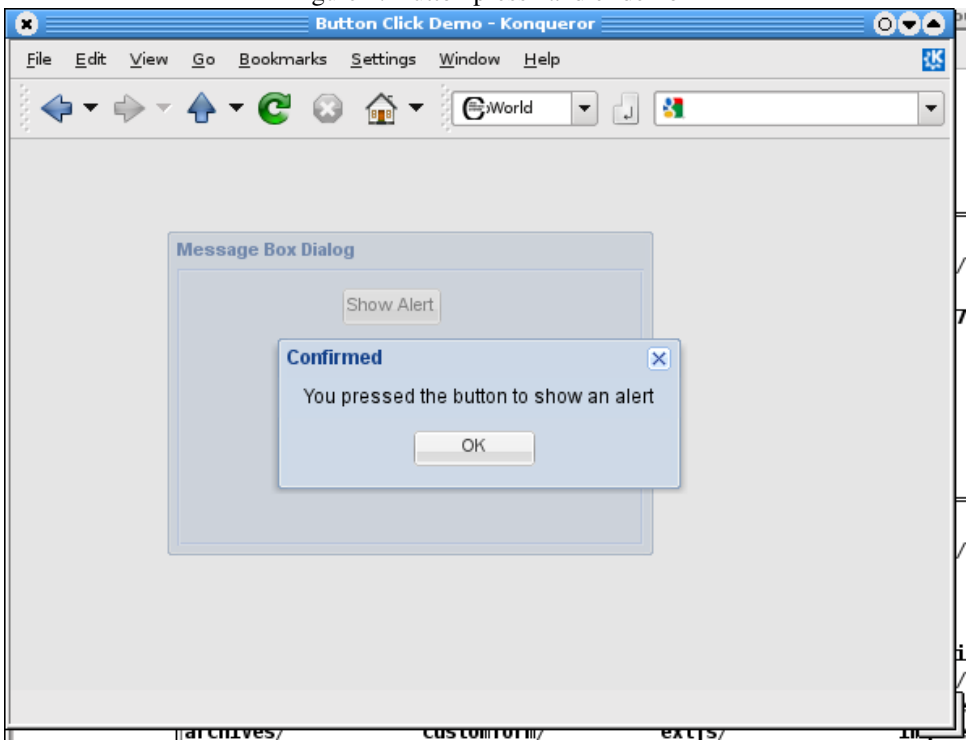
This looks much like the previous example, but now, instead of setting the `html` property, it adds a button to the form. The `TExtButton.AddTo(Items)` statement creates a TExtButton instance, and adds it to the form: the `Items` property is a property of the form: it is a list which contains all child widgets of the form. The AddTo constructor adds the new button instance to the children of the form. After setting the X,Y coordinates of the button and a caption, the 'Handler' property is set.

`Handler` is of type TExtFunction. Contrary to what may seem intuitive, this is an object, not a function. The `ExtMessageBox.Alert` call returns such an object. Assigning the object to the 'Handler' property, will not set any procedural property, but instead will convert the object to Javascript, which will be sent to the browser as the code to be executed when the button is pressed. The produced Javascript looks like this:

```
items:[new Ext.Button({x:100,
                       y:10,
                       text:"Show Alert",
                       handler:function(){
  return Ext.MessageBox.alert("Confirmed",
    "You pressed the button to show an alert");
})]
```

When compiled and executed and shown in a browser, the result will look something like in figure 2 on page 8 The `ExtMessageBox.Alert` call returns some pre-defined

Figure 2: Button press handler demo



javascript. It is also possible to create more complex javascript by specifying a method and turning all javascript it produces into an event handler. For example, the above code could as well have been coded as follows:

```
        Handler:=JSFunction(@ShowAlert);
        end;
      Show;
    Finally
      Free;
    end;
end;

Procedure TMyButtonDemo.ShowAlert;

begin
  ExtMessageBox.Alert('Confirmed',SButtonPressed,Nil);
end;
```

The `JSFunction` procedure takes any method, executes it, and converts all the JavaScript that was produced while the method executed into a `TExtFunction` object. This `TExtFunction` object can then be assigned to the `Handler` property of the button instance, and will result in an event handler.

Note that this method is executed on the server, while the code for the browser is generated, not when the button is actually pushed ! It is simply a way to produce Javascript event handlers, and can therefor e.g. not use any values that the user entered, because they are not yet available. The method can only produce simple Javascript that will be sent to the server, no complicated statements can be made: real business logic cannot be converted like this.

7 Using AJAX

The above method for creating browser-side javascript - while useful, as can be seen by studying the ExtPascal examples, is limited. It cannot contain any real business logic. But luckily, there is more. ExtPascal can convert any published method to a event handler for an AJAX request. Any JavaScript produced during the AJAX request, will be sent back to the browser and evaluated in the browser's Javascript engine. The Ajax request can contain as many parameters as needed, which can be collected from the various gui elements in the form.

This is easily demonstrated with the following example: a form with an entry box and a button on it, which, when pressed, will invoke an Ajax method on the server. The contents of the entry box will be sent to the server, and the server will display a message box with the entered value. The form is produced as follows:

```
 procedure TAjaxPrimer.Home;

Var
  Person : TExtFormTextField;
  Form   : TExtWindow;

begin
  Form:=CreateWindow;
  Try
    with TExtFormFormPanel.AddTo(Form.Items) do
      begin
      LabelWidth  := 70;
      Border      := false;
      XType       := xtForm;
      ButtonAlign := baRight;
      BodyStyle   := SetPaddings(5, 5);
      DefaultType := xtTextField;
      Defaults    := JSObject('width: 150');
      Person    := TExtFormTextField.Create;
      with Person.AddTo(Items) do
        begin
        Name       := 'person';
        FieldLabel := 'Your name';
        InputType  := itText;
        end;
      with TExtButton.AddTo(Buttons) do
        begin
        Text:='Say hello';
        Handler:=Ajax(@SayHello,['Person',Person.GetValue]);
        end;
      end;
    Form.Show;
  finally
    Form.Free;
  end;
end;
```

The `CreateWindow` call creates a window as in the previous examples. To this window, a `TExtFormFormPanel` instance is created and added. This is a panel, to be used specially when created fill-in forms. It will choose a nice layout for all elements dropped on it, and

contains a special area for buttons. (usually there is only 1 button in this area: the 'submit' button). The various properties shown in code determine the layout of this panel.

Of more interest is the `Person` instance, of class `TExtFormTextField`. This creates an edit field with a label in front of it, which will display whatever is entered in the `FieldLabel` property. The `itText` value for `InputType` tells it to act as a simple edit field. (A value of `itPassword` for instance would let it act as a password edit).

Finally, a button is added. A noteworthy item about the button is that it is added to the `buttons` property of the `TExtFormFormPanel` instance instead of being added to the `Items` property. The effect is that the panel will then display the button in the special button area.

The second item to note is the assignment to the handler property:

```
Handler := Ajax(@SayHello, ['Person', Person.GetValue]);
```

The Ajax call takes as the first argument a published method. This method will be invoked through an Ajax call when the user pushes the button. What follows is an array of name, value pairs: this can be static values, but also `TExtFunction` instances. The `Person.GetValue` call returns a `TExtFunction` object: The object will contain the necessary Javascript to retrieve the value of the Person edit field.

The name-value pairs will be sent as parameters to the AJAX call, which can then access them through the `Query` property of the `TExtThread` class, as in the `SayHello` method shown below:

```
procedure TAjaxPrimer.SayHello;

Const
  SCaption = 'Say Hello demo';
  SGreeting = 'Hello, %s!'#13#10'How are you ?';

begin
  ExtMessageBox.Alert(SCaption,
                      Format(SGreeting,[Query['Person']]));
end;
```

From the code it can be seen that the only thing done by the ajax handler, is to produce the necessary Javascript code to show a nice message. When executed, the result will look something like figure 3 on page 11.

8 Conclusion

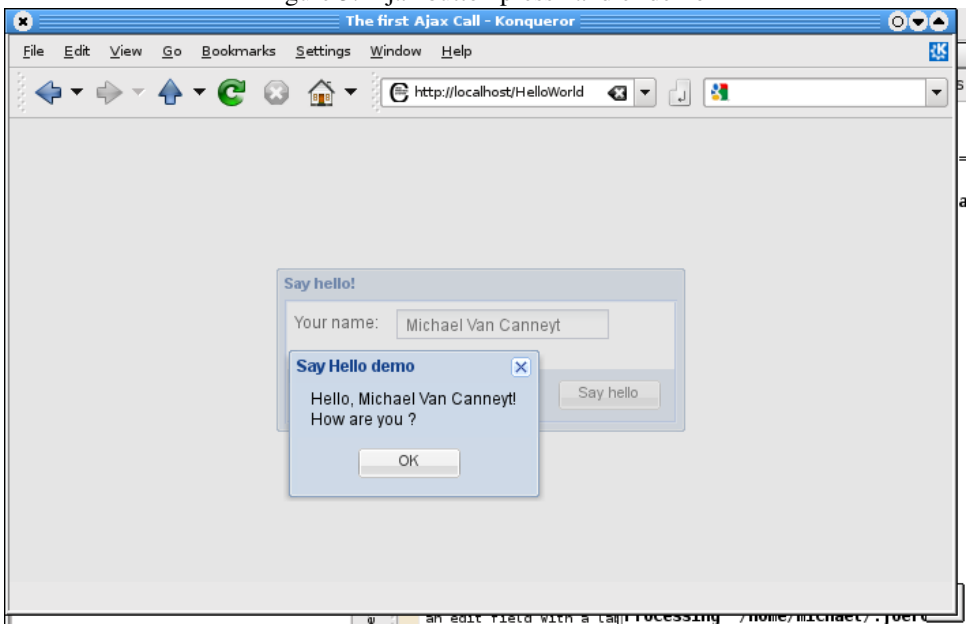
ExtPascal looks like a promising technology if one wishes to create good looking web-applications. The techniques used in this framework - self-translation - are not limited to ExtJS: one could attempt another set of Javascript controls.

Extpascal is a young project, and this shows in some of it's deficiencies:

1. Currently, only FastCGI is supported. It should be said that there is experimental support for a self-contained Indy HTTP Server, and that a CGI-to-FastGI bridge exists.

2. It is not possible to design the web-page visually yet: everything must be created in code, which is not very productive. There are some obstacles currently prohibiting

Figure 3: Ajax button press handler demo



this: the classes do not descend from TComponent, properties are not published. There is a sub-project which aims to transform delphi .dfm files or Lazarus .lfm files to Pascal code that recreates the form in Javascript, but this is a very cumbersome approach, and since the various properties of the controls in the VCL/LCL differ wildly from the ExtJS properties, this is in the opinion of the author not a very viable approach.

3. Lastly - but connected to the first item - the communication layer (FastCGI) and the content-producing layer (TExtThread) are currently very much interwoven. This prohibits the use of Extpascal with other communication layers, such as websnap or fpWeb. Even creating an apache module is also not an option.

The author of the ExtPascal has recognised these problems and is working to solve them, so it's definitely worth keeping an eye on this new project.