Embedding Gecko in a Delphi application

Michaël Van Canneyt

August 4, 2008

Abstract

Delphi comes standard with a TWebBrowser component that uses Internet Explorer to show webpages. For those that prefer Free software or simply prefer Mozilla and it's derivatives, it is also possible to embed Gecko (the rendering engine of Mozilla) in a Delphi application. This article shows how.

1 Introduction

Most programs these days have some kind of web component - an update page to which one can browse, help systems, whatever. Access to the web is easily done in Delphi using the `TWebBrowser` component that is distributed by default with Delphi. That component uses the services provided by Microsoft's Internet explorer to load and show HTML pages. However, this is not the only possibility.

Firefox - or any other derivative of the Mozilla browser suite - is a powerful web browser. It is based on a free set of components or API, called Gecko: Gecko is bundled in a DLL that is deployed with FireFox (or any Mozilla application): this DLL can be used to embed Gecko in a Delphi application.

Gecko is built using the XPCom technology: a cross-platform COM technology, which is compatible to Microsoft's COM implementation on Windows. Practically, this means that Gecko defines a series of interfaces which can be used to create a Gecko application. The interfaces map exactly to Delphi's interfaces, and this makes it relatively easy to create a Gecko application.

There are 2 ways to embed Gecko in a Delphi application. One is using an ActiveX wrapper around the Gecko SDK, the other is to use the Gecko SDK directly. The second option is the way to go, as it gives access to all of the functionality of the Gecko SDK and should be cross-platform, whereas the ActiveX component exposes the functionality only partially, and requires additional installation steps.

2 Gecko SDK

All interfaces of Gecko and XPMCom have been translated to Delphi (seemingly by a Japanese programmer, whose name is hitherto unavailable to the author) and have been declared in a series of units that are available from

`http://nesitive.net/nesitive/all/b02b7441f5ef25a07110a55959e89575.html`

As an alternative, there is a download on

`http://ftp.newbielabs.com/Delphi Gecko SDK/Readme.htm`

Which also has some sample applications. Finally the sources are in Subversion on sourceforge

```
http://sourceforge.net/projects/d-gecko
```

The sources on this location are changed so they work with Gecko version 1.8 and 1.9, but the interface (in particular, the startup and history mechanisms) are slightly different. The example here uses the version found on

```
http://ftp.newbielabs.com/Delphi Gecko SDK/Readme.htm
```

Using the version found on Sourceforge should take only minor modifications to the code.

The archives contain 2 delphi packages:

geckosdk This is a run-time package (.dpk) which contains the raw XPCom and Gecko API definitions. This package should be simply compiled, and the resulting .bpl should be in the PATH. This will be automatically the case for a standard Delphi installation.

geckocomponents This run and design-time package contains 2 wrapper components: a visual components that can be used to ember a web-browser window on a form, and a profile manager component. This package uses the geckosdk package for the low-level interface. The package should be compiled and installed in the IDE.

3 First steps

When the packages have been installed, a new tab called 'Gecko'should appear on the component palette. The `TGeckoBrowser` component is the browser component. To demonstrate it's use, a new application can be created witha single form (`TMainForm`). TGeckoBrowser component is dropped on it (name it `GBDemo`) together with an Edit Control (name it `EURL`) and a button (name it `BGo`). Some visual properties such as anchors can be set so the form behaves nice when rescaled, and then the real work can start:

In the `OnClick` event handler of the `BGo` button, the following code can be inserted:

```
TMainForm.BGoClick(Sender : TObject);

begin
  GBDemo.LoadURI(EURL.Text);
end;
```

That's it. The form is now a fully functional browser with an address bar. When a URL is entered and the button is clicked, the browser component will load the URL. Once it is loaded, the links can be clicked, and the browser component will follow all the links - no additional programming is required.

When running this program, some problems may pop up: the Gecko Runtime Environment (GRE) should be installed in the same directory as the demo application. If this is not the case, an error may occur. The download at the following location

```
http://ftp.newbielabs.com/Delphi Gecko SDK/Readme.htm
```

contains a complete GRE (in the file gre.zip), which can be unzipped in the directory of the application. Downloading and unzipping this file makes it possible to start the application, in which case the running application can look like figure figure 1 on page 3.

Figure 1: The running demo application

The `TGeckoBrowser` component has some event handlers. One of these handlers is the `OnTitleChange` event, which is called when the page title of the currently displayed HTML page changes. This can be used for instance to set the form caption to the HTML page title, as in the following example:

```
procedure TMainForm.GBDemoTitleChange(Sender: TObject; const Text: WideString);
begin
  Caption:=Text;
end;
```

The `OnStatusTextChange` event can be used to display status messages; These messages are sent by the Gecko Engine when downloading pages, images and other web material. Additionally, when the mouse cursor hovers over a hyperlink in a webpage, it sends a status message with the link information. Using this event is again straightforward: a statusbar can be dropped on the form (call it `BDemo`), and the event handler can be coded as follows:

```
procedure TMainForm.GBDemoStatusTextChange(Sender: TObject; aMessage: WideString)
begin
  SBDemo.Panels[0].Text:=AMessage;
end;
```

4 A history mechanism

When the current page changes in the browser, the `OnLocationChange` event can be used to implement a history mechanism. To show this, a toolbar with 2 buttons is

dropped on the form. The first button (`TBBack`) gets its style set to 'tbsDropdown', and a popup menu `PMHistory` is attached to it. The popup menu is filled with items in the `OnLocationChange` event, as follows:

```
procedure TMainForm.GBDemoLocationChange(Sender: TObject; const uri: string);

Var
  Mi : TMenuItem;

begin
  Mi:=TMenuItem.Create(Self);
  Mi.Caption:=URI;
  MI.OnClick:=HistoryClick;
  PMhistory.Items.Insert(0,MI);
end;
```

When the menu item is clicked, the `HistoryClick` event handler is called, and it simply loads the requested page:

```
procedure TMainForm.HistoryClick(Sender : TOBject);

Var
  MI : TMenuItem;
begin
  MI:=Sender as TMenuItem;
  GBDemo.LoadURI(MI.Caption);
end;
```

Obviously, this mechanism can be enhanced to e.g. display a short version of the webpage location. The OnClick handler of the `TBBack` button itself contains the following code:

```
procedure TMainForm.TBBackClick(Sender: TObject);

Var
  MI : TMenuItem;

begin
  if PMHistory.Items.Count>0 then
    begin
    MI:=PMHistory.Items[0];
    GBDemo.loadURI(MI.Caption);
    end;
end;
```

Which means it jumps to the last item in the list.

The forward button is a bit more problematic, as it would require keeping the last used position in the history list. Luckily the TGeckoBrowser component has it's own history list mechanism, which takes care of all these details. The following properties maintain the history list:

HistoryEntry This is a zero-based indexed property with the list of visited pages; Each entry is a class of type `TGeckoBrowserHisoty` (a typo in the implementation), which has relevant 2 properties: `Title` and `URI`.

HistoryCount This is the number of entries in the history list.

HistoryPosition This is the position, in the history list, of the currently shown web page.

These 3 properties should be enough to populate a list (or menu) with recently visited pages, and to decide whether a forward or backward navigation is possible.

The following methods can be used to navigate in the history list:

GoBack Navigate back to the previous item in the list.

GoForward Navigate forward to the next item in the list.

GotoIndex Accepts a single parameter: the index of the item in the history list that should be jumped to.

All three combined can be used to create a history mechanism. To show this, a main menu is dropped on the form, with a 'History' menu item. The item contains 3 sub menuitems: 'Recent', 'Back' and forward'. The `OnClick` events of the latter two look like this:

```
procedure TMainForm.MIBackClick(Sender: TObject);
begin
  GBDemo.GoBack;
end;

procedure TMainForm.MIForwardClick(Sender: TObject);
begin
  GBDemo.GoForward;
end;
```

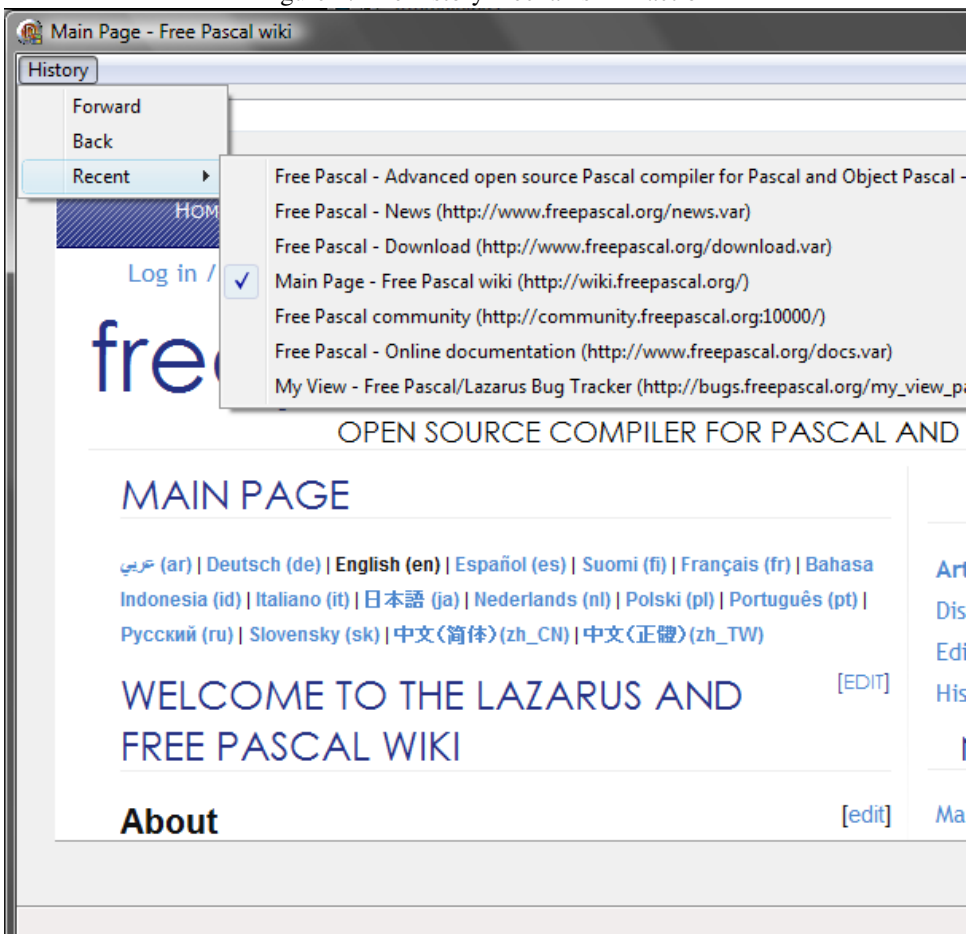
In the 'OnClick' event of the 'History' menu item, the 'Back' and 'Forward' menu items are disabled if their action cannot be executed, and the complete history is appended to the 'Recent' menu item:

```
Procedure TMainForm.MHistoryClick(Sender: TObject);

Var
  P,C,I : integer;
  H : TGeckoBrowserHisoty;
  MI : TMenuItem;

begin
  P:=GBDemo.HistoryPosition;
  C:=GBDemo.HistoryCount;
  MIForward.Enabled:=(P>=0) and (P<C-1);
  MIBack.Enabled:=P>0;
  MIRecent.Enabled:=C>0;
  MIRecent.Clear;
  for I:=0 to C-1 do
    begin
    H:=GBDemo.HistoryEntry[I];
    MI:=TMenuItem.Create(Self);
    MI.Caption:=H.Title+' ('+H.URI+')';
    MI.Checked:=(P=I);
    MI.OnClick:=DoRecent;
    MIRecent.Add(MI);
    end;
end;
```

Figure 2: The history mechanism in action



This routine is quite straightfoward: a menu item is created for each page in the history list, and the caption is created from the page title and URI. The menu item which represents the current page gets a check in front of it. The `DoRecent` event handler is called when the user clicks a menu item, and it executes the following code:

```
procedure TMainForm.DoRecent(Sender: TObject);

Var
  Mi : TMenuitem;
  I : Integer;

begin
  MI:=Sender as TMenuItem;
  I:=MIRecent.IndexOf(MI);
  if I>=0 then
    GBDemo.GotoIndex(I);
end;
```

It simply determines the position of the menu item in the history list, and jumps to this position. The result of all this coding can be seen in figure 2 on page 6

5 Deployment issues

As said earlier, the GRE must be installed in the application directory for the demo to work. This requirement is rather restrictive, but is an implementation choice of the `nsXPComglue` unit. The location of the `GRE` is stored in the `GRELocation` variable in the `nsXPCOMglue`. This variable is only in the implementation section, but can be moved to the interface section of the unit it is in.

The location of the GRE can then be set in the initialization section of the main form. For instance, if the GRE is in a subdirectory 'GRE' of the demo application, then the following code will load the GRE in that directory:

```
Initialization
  GRELocation:=ExtractFilePath(Paramstr(0)+'GRE');
end.
```

Because the GRE tries to load other libraries, it is necessary to add this directory to the `PATH` environment variable: then Windows will search for other libraries in the GRE installation directory.

6 Conclusion

As shown in the above code examples, embedding a Gecko-based browser in a Delphi application is suprisingly easy: The `TGeckoBrowser` component makes it particularly easy to use. The only caveat found currently is the deployment: the GRE should be installed correctly with the application.

What about Lazarus ? Since the code is fairly standard Object Pascal, it is to be expected that the code should compile and function under Lazarus, and normally it should function under all platforms, since XPCom is a cross-platform technology. Currently the code compiles, but does not function correctly: the startup for XPCom fails. The matter is currently under investigation by the FPC team, and success will surely be reported in this magazine.