Le nouveau Morfik est arrivé
(The new Morfik has arrived)

Michaël Van Canneyt

April 7, 2008

Abstract

Soon, it will be 2 years ago since the first review of Morfik in this magazine. When the current issue of Toolbox will be available in the newsstands, Morfik will have unveiled version 2 of their amazing webapplication development product. A preview, based on the first beta of version 2.

1 Introduction

Almost 2 years ago, a review of a Morfik almost-version-1 was presented in Toolbox. While the first release product was definitely a great step in Ajax and Web 2.0 development, the morfik development team hasn't been sitting idle since the initial release of their webapplication development toolchain, and is about to release version 2 of their IDE and toolchain. Its release is expected at the web 2.0 event at the end of April in San Francisco. In this article, the new features of the second version will be examined

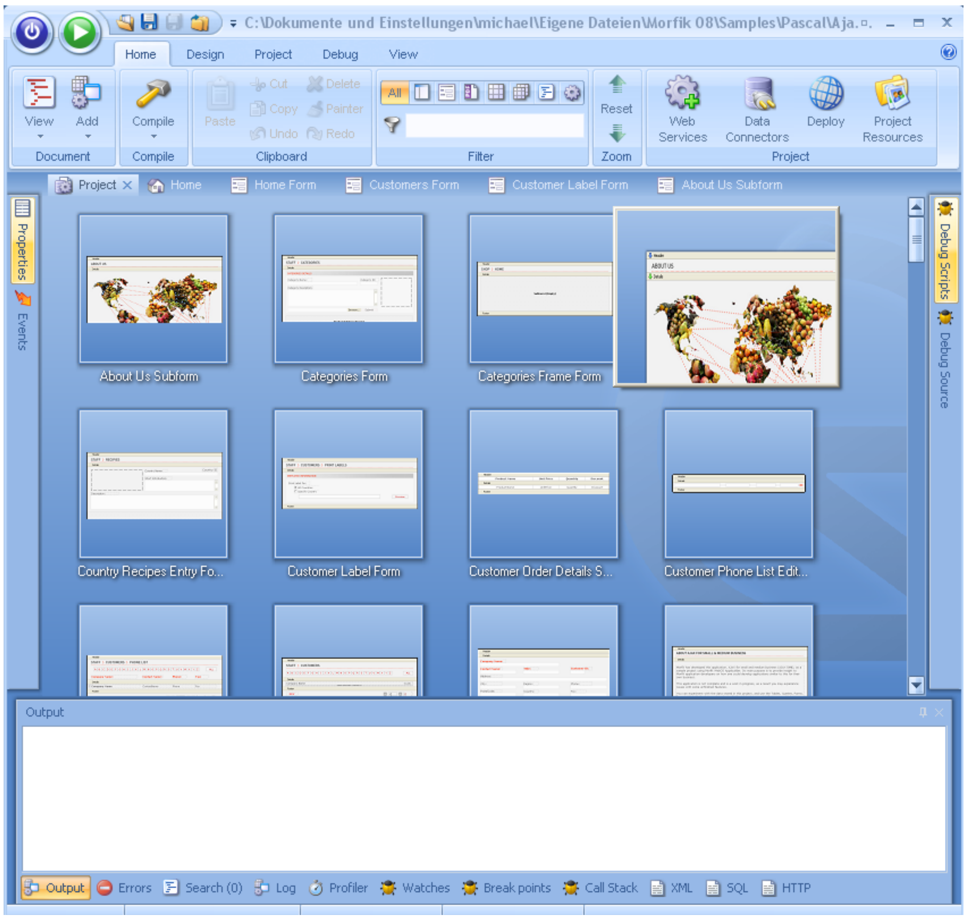
When starting the new IDE, the first change to Morfik will hit the user at once: the IDE has been completely reworked. It has been given a Vista look, showing that a development tool and a slick user interface are not necessarily contradictory. The color scheme of the IDE can be set in the preferences. The so-called ribbon has replaced the classical toolbar and menu, and a quick-access bar can be configured with some commands not available by default, such as 'Close all' to close all tabs. Nevertheless most common fnunctions are available directly from the ribbon.

When a new project is started, or an existing project is opened, the project view is shown (see figure 1 on page 2) and here the new looks are also accompagnied with a more visual way of presenting the project: a thumbnail is presented for all objects in the project. The thumbnail shows a miniature of the object it represents: It is possible to select the objects based on their visual appearance alone.

For projects with a lot of forms, the thumbnails can be resized using the zoom controls, and the number of thumbnails can be reduced with the filtering functionality, prominently placed in the middle of the ribbon. In figure 1 on page 2, a new and nifty feature of the IDE is also visible: when the mouse hovers over a tab that would open a form/report/table when clicked, the IDE will present a preview of what is under the tab - again allowing to visually select an already opened object. This focus on visualizing things will come back in various other places.

Obviously, an upgraded IDE is not all that was done. Many things have been added to the Morfik framework as well. One of these is (again) a very visual one: the concept of a Page. A page can be thought of as a form (in reality, it is not: it is a totally separe class), but with some automatisms built in:

Figure 1: The project view



- Automatic layouting. Several pre-defined layouts are available. Only subforms can be placed on a page; controls cannot be dropped directly on a layout page.
- Accessible through a simple URL: this allows to make easy to remember URLS for special parts of the website, for instance:

  ```
  http://www.mycompany.com/#downloads/
  ```
- Forms that placed on it will actually show their content in the IDE - again the focus on the visual designing.
- to place a new form in a partition of a Page, one can double-click on the partition, and the new form wizard will start. Again visual designing.

A page can have a theme assigned to it. There are currently some pre-defined themes, and these control things such as colors, background images and fonts for the forms and subforms.

Additionally, the theme can define a set of styles for each of the controls: When adding a control to a form, a style of the control can be chosen. These styles should be thought of as functional styles: emphasized/normal/important textlabels etc.

In the beta that was made available for testing, it was not yet possible to define themes in the IDE.

Traditionally, a morfik application consists of 2 parts: the browser part, and the server part. The server part can be a stand-alone webserver, or a module for a webserver such as ISS or Apache. However, for simple websites that do not need a database, it is not necessary to create a full-blown webserver: the Javascript that makes the browser part tick can be in files on the server just as well: this could be deployed to any existing webserver, without any need for server logic.

There are even more cases when no server part is necessary: If the browser application just needs to connect to existing webservices, it doesn't need a server part of its own. If the existing webservice conforms to a certain API, it even be used as a datasource to create data bound browser-only applications.

In fact, this feature is used by Morfik to create a browser-only application which uses XML files on the server as read-only tables - the file is transferred to the browser and treated as a local read-only datasource. This can for instance be used to browse through a catalog, and then a webservice can be used to place an order. Or one can imagine even using the XML output of the svn log command as a table, to create an interface to a Subversion repository. A picture browsing gallery could also be constructed, with the XML file containing the list of files and possible annotations.

Creating these tables in the IDE, would be no different than creating them with Firebird as a backend: the only difference is that it's not possible to create queries on the XML files.

Realizing that there are plenty of cases where a serverless application would be very useful, the Morfik team has made it a prominent feature of M2: There is a special button to start a new browser-only application. Creating such an application is not different from a normal Morfik webapplication, with the exception that there is no support for queries, reports and server side code.

When the browser-only application is compiled (obviously, the browser code still must be compiled), all files needed to deploy it are placed in a special directory below the project. It has the name of the project, followed by 'Pub'. The contents of this directory can be copied to the webserver and is ready for use.

The 'Webservice-based datasources' was mentioned above already in the context of browser-only applications. Basically, it means that the browser side of a Morfik application can treat a webservice as a source of data - and hence, bind a form to it. Obviously, not any webservice can be used as a source of data, it has to correspond to a certain API (the specifications of this API are not yet available).

One such possible source of a webservice based data source is - obviously - another Morfik server application: Every query and table of a Morfik application can be accessed as a webservice based datasource. To do this, the 'Data connectors' button can be used to connect to such a datasource: In the first page of the 'Data connectors' wizard, the 'Web Service-based Data Sources (*.wsdl)' option must be chosen. In the second page the filename of the WSDL file should be entered: the Morfik IDE will analyse the file and present a list of data sources that can be used in the new application. After the objects were chosen, the datasources appear in the list of tables as if it was a normal table.

If every table or query in a Morfik application was available for read/write, this would not be very safe. Luckily, it is possible to control whether a table (or query) is exported from a Morfik application in its WSDL file: in the table design view, the ribbon displays 2 buttons for Read/Write WSDL publishing of a table. Pressing the buttons will publish the table for read and/or write operations.

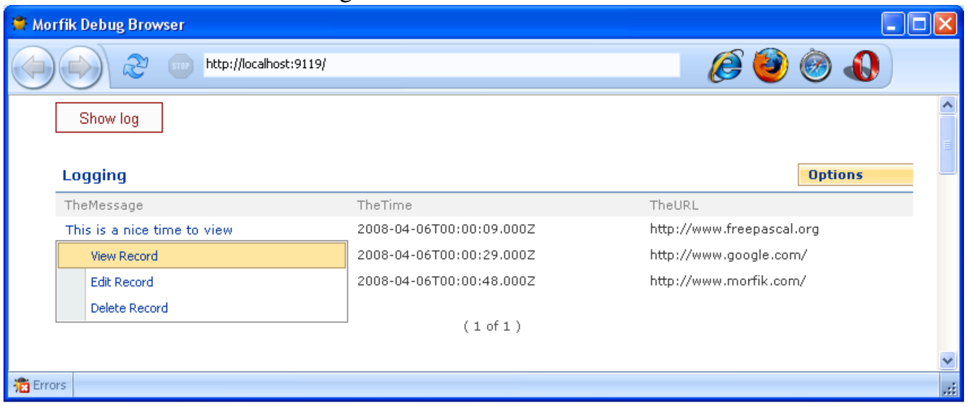
Javascript has some security restrictions: it can only access webpages (or webservices) that are in the same domain as the current page. This means that WSDL datasources should also be in the same domain as the current page, to be accessible. To circumvent this restriction, the Morfik server application can act as a transparant proxy for webservices: it can be told to execute a webservice's method at any location, and return the result to the browser. This is not only true for web service-based datasources, but for any webservice (webmethod). This is of course a very powerful mechanism, allowing to access just about any webservice from within the browser side of the application. The security implications of this probably need some investigation, and some mechanism for restricting this mechanism may need to be put in place.

A new visual feature is the data form. The data form can be used to display data from any data source in a tabular format, much like a grid. To do this is actually quite simple. Supposing a form has a subform container called 'SubForm1'. Then the subform can be filled with the data from the datasource 'Logging' as follows:

```
XApp.OpenDataForm('MyForm:Subform1','Logging');
```

The result of this can be shown in figure 2 on page 4. The data form has a lot of features built-in: the grid can be sorted on any column, a filter condition can be added for each column, and the records can be edited, with the help of a small local menu. Combined with

Figure 2: A data form in action



Logging

TheMessage	TheTime	TheURL
This is a nice time to view	2008-04-06T00:00:09.000Z	http://www.freepascal.org
	2008-04-06T00:00:29.000Z	http://www.google.com/
	2008-04-06T00:00:48.000Z	http://www.morfik.com/

(1 of 1)

the Web-service based data sources, this is a very powerful mechanism, allowing a Morfik webapplication to act as a central data access point for a whole enterprise - data mining made easy.

RSS feeds are a popular feature of many websites these days. Well, the Morfik IDE makes creating a RSS deceptively simple: Each query datasource in the application can be used as the source of an RSS feed. A click on the 'RSS feeds' button in the ribbon will start the RSS feed wizard. All that needs to be done is to indicate the query, and which fields of the query should go in the various fields of the RSS feed, and it's all done. Optionally, a link-back page can be specified, which will be opened in a browser when a feed item is clicked.

The deployment wizard is a new feature. It is supposed to work in an environment where there is a webserver running a special Morfik XApp. When the deployment button is clicked, the Xapp will be compiled (as a module, most likely) and uploaded to the server. There is a dialog which asks where the module should be uploaded to (essentially a server name, username and password), and off it goes. As close to one-click deployment as one can get.

The first intention is that Morfik sets up servers, to which you can upload your Morfik-created website. Secondly, the Morfik service that runs on the server will also be distributed at a later stage, so it will be possibly to set up such a deployment server all by yourself.

Most features till now have been rather visual, little had to do with the actual code. Here also, the IDE has had some features added. The first of these is again visual: Web Actions. Web actions are simply a way of generating code for you for some common tasks. Often, a button will be used to close a form, post data, open a new form, navigate: a whole lot of things that are done often. The IDE now offers a wizard that will allow you to choose one of a set of predefined actions, much like Delphi has a set of standard, pre-defined actions: it is started from the events property editor (the button with the ellipsis, visible in figure 3 on page 6), and will guide you through the steps to create the action. After it is done, the code for the action is generated and put in the event handler. The IDE also recognizes this code as a web action, and will display it in the event inspector as such: the parameters can even be set (see again figure 3 on page 6).

A not-so-visual change is the concept of helper methods. It is based on the point of view that everything is a class. This comes natural to a Java programmer, who is used to it. For pascal or visual basic people, there is a fundamental distinction between classes and basic types. Well, helper methods can be used to make basic types look like classes. One can define for instance the following:

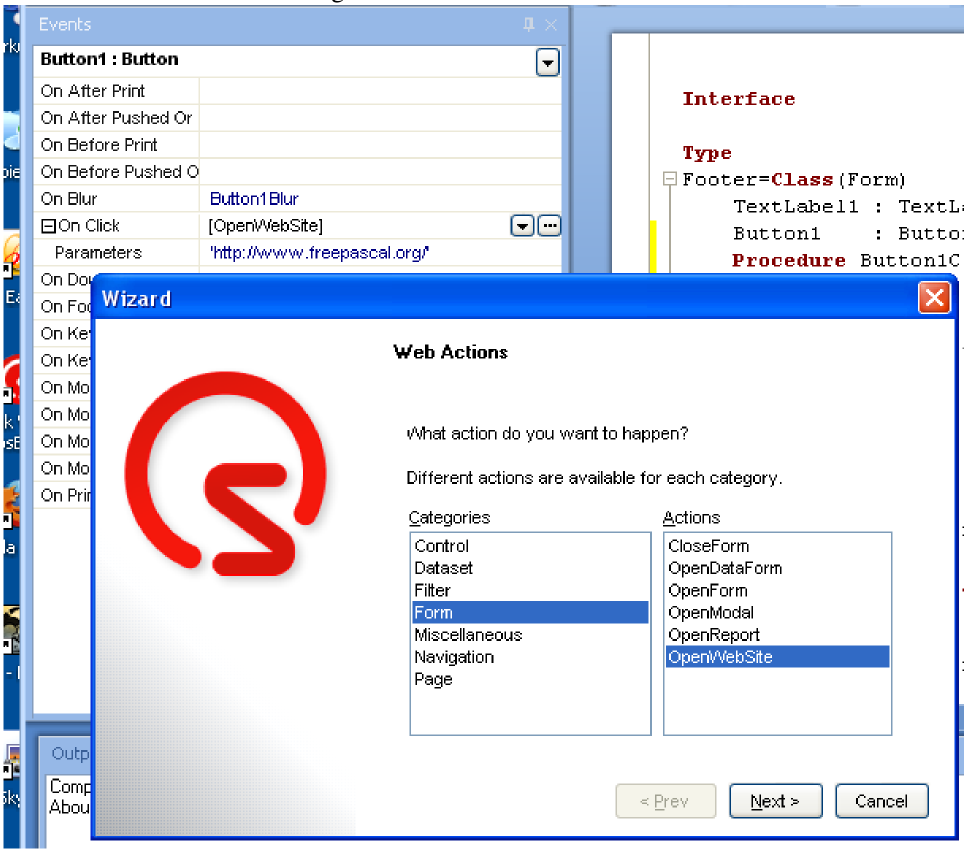
```
Function Integer.AsMyString : String;
begin
 Result:=IntToStr(Self);
 While Length(Result)<10 do
   Result:='0'+Result;
end.
```

The `Self` refers to the value that the helper method will be applied to - which is an integer in the example above. The code above transforms an integer to a string, padded with zeroes so it has at least length 10. It can be used as follows:

```
Var
  I : Integer;
  S : String;

begin
  S:=I.AsMyString;
end.
```

Figure 3: Web Actions in Action



Button1 : Button

On After Print	
On After Pushed Or	
On Before Print	
On Before Pushed O	
On Blur	Button1Blur
On Click	[OpenWebSite]
Parameters	'http://www.freepascal.org/'

```
Interface

Type
Footer=Class(Form)
     TextLabel1 : TextL
     Button1    : Butto
     Procedure Button1C
```

Web Actions

What action do you want to happen?

Different actions are available for each category.

Categories	Actions
Control	CloseForm
Dataset	OpenDataForm
Filter	OpenForm
Form	OpenModal
Miscellaneous	OpenReport
Navigation	OpenWebSite
Page	

Basically, the idea is not so different from C macros, with the difference that the code is type checked. Internally, that is also what happens: the helper method implementation is transformed to a regular function with some special name, and the call of this helper method is transformed to a normal function call. This transformed code is then fed to the pascal compiler - making sure that type safety is preserved. Helper methods can also be applied to classes: Obviously, because of the way this is processed internally, they fall outside the normal class hierarchy, so private and protected fields are not available.

One can see that the Morfik team likes this feature: They have added a lot of such helper functions for the basic types such as integer, string and some of the classes: the `SystemHelperMethods` module contains these helpers.

In Morfik 1, handling a webservice method in the browser was rather cumbersome: It was necessary to specify a callback function to handle the response of the method call. This has now been alleviated: the callback is no longer necessary (but can still be used, obviously).

Suppose there is a webservice with the following interface:

```
Procedure MyLogin (AUser, APassword : String; Out UserID : Integer);
```

Then it can now be called as:

```
 MyLogin.AUser:=Edit1.Text;
 MyLogin.APassword:=Edit2.Text;
 MyLogin.Execute; // Go to server
 If (MyLogin.AUserID<>-1) then
   begin
   // Do Something.
   end;
```

The call looks synchronous: when the 'Execute' method returns, the results from the webservice have been saved to the various fields and can be used: The program logic will become a lot easier with this particular feature.

2 Conclusion

With this second major release, the Morfik team has shown the direction which they plan to follow: they have resolutely pulled the card of Visual development and easy deployment. While the beta was clearly a beta (there were some performance issues), the overall feel is consistent and very true to this vision of visual development.

Complete Nirwana has not yet been reached, however: For instance, the set of controls is essentially the same as in version 1, which is - to the taste of the author - a particularly weak spot. At the time of writing, there was no clear indication what will happen on the controls front, although it is likely that some controls will make it to the final version 2 release. Despite this, the ease with which one can create web applications has once more been considerably improved with this second version of Morfik, making an upgrade more than worth it.