Getting started with Morfik: Creating a GUI

Michaël Van Canneyt

June 6, 2007

Abstract

Morfik is an exciting new environment for creating webapplications. It is unique in many ways and in this article, it will be shown how Morfik can be used to create webapplications that function (almost) like a regular desktop application.

1 Introduction

Creating webpages is easy these days. Creating interactive webpages is also easy. Creating rich internet applications or webapplications is already more difficult: intimate knowledge of Javascript and AJAX is required in order to be able to create an application which looks and reacts like a regular desktop application. Morfik is a development environment which makes creating a web application as easy as creating a desktop application: Morfik and it's unique architecture was introduced in 2 previous articles. In a couple of articles Morfik's architecture will be explained in more detail. Various areas of Morfik will be explained:

- GUI design: how to code the visual part of the application. This will cover basic events, and how to open other windows in the browser.
- Database access: in this part, the possibilities for data access will be discussed, and how Morfik can tightly couple the GUI with a database. It also includes a banded reporting engine, which emits nice-looking PDF documents.
- Webservices: no web-application tool would be complete without support for web-services. Morfik has support for consuming third-party webservices, but can also be used to create webservices. Indeed, webservices are used to let the GUI (which runs in the browser) communicate with the Morfik server application.

Each of these areas will be covered in a separate article.

In this article the nuts and bolts of coding a GUI in Morfik will be explained. No database access will be performed yet, only GUI elements will be treated: the various possibilities, some methods that will be absolutely needed. The demo is coded as a single morfik application, with the different parts available as separate forms in the demo, accessible through the main menu.

2 The IDE

The various parts of the IDE have been discussed more extensively in a previous article. The main parts of the IDE are organized in a way which closely resembles MS-Access, and consist of the following elements:

1. The project manager, which displays the various objects in the application, either all in one overview, or organized per category.
2. The table design window, which can be used to design the database to be used by the Morfik application (all morfik applications have a database)
3. The query design window, which can be used to design various queries for use in the application.
4. The Form design window, used to design the GUI elements of the application.
5. The report design window, used to design reports which can be sent to the user.
6. The web methods design window, used to design webservices.
7. The modules window, used for coding common code.

For the GUI design part, the code editor and the form designer are the main instruments. The form designer is shown in figure 1 on page 3. The form designer has several parts:

- The design preview of the form itself, on which visual elements can be dropped and manipulated with the mouse.
- The object inspector, which allows to view and set the various properties of the currently selected element.
- The toolbar, with the toolbox, formatting and alignment toolbars: they can be floating toolwindows or docked on the toolbar (by default the toolbox is floating).
- The client (browser) code for the form. The little tab must be clicked to show the code.
- The server code for the form. The little tab must be clicked to show the code.

Each form is implemented as 2 classes: one instance which runs on the server, and one which runs on the client. The F12 key cycles the form editor through the preview window, the server code and the client code window. Both the server code and the browser code is written in one of 4 languages:

Object Pascal

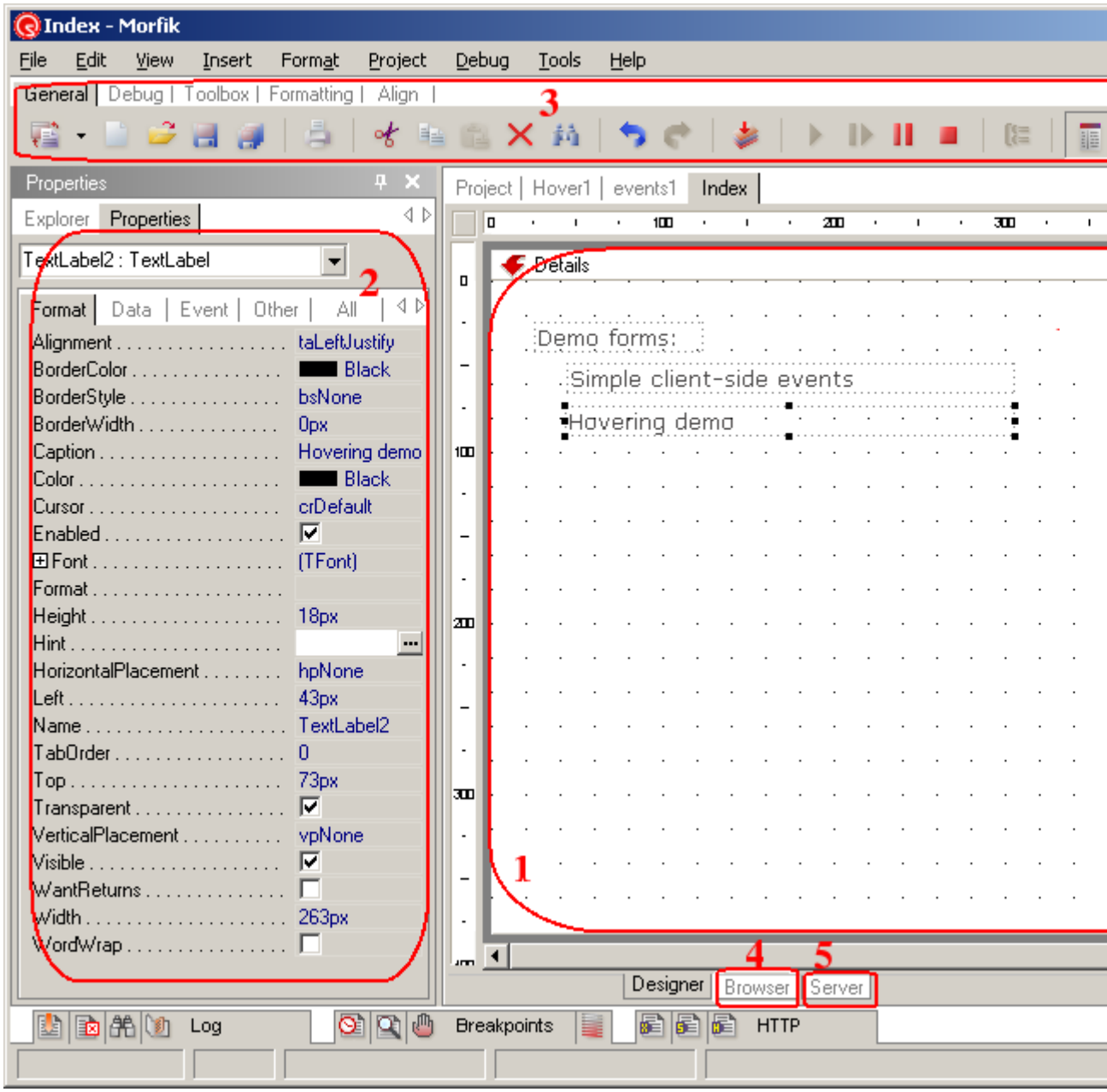
Basic

C#

MorfikJ which is comparable to Sun's Java.

In future, more languages may be added. Not all constructs available in the original 4 languages are supported by Morfik: The Morfik IDE translates the language to Object Pascal for code which must run on the server, while the code which runs in the browser is translated to Javascript. However, the 4 languages are supported equaly well. Further, it should be obvious that not all kind of code can run in the browser: the browser offers only a limited environment, which is basically limited to the window being shown, and HTTP communication with the server. That means that for serious tasks, the code must be run on the server.

Figure 1: The form designer window



3 Controls and their properties

Morfik uses a component-based approach to a web-page: a web page is a form on which various components (or controls) are dropped, just as one does in a RAD such as Delphi. The controls are then converted to a combination of HTML tags and style sheets, which are then rendered in the browser.

The appearance of the controls is governed by a lot of properties: the most basic ones are `top`, `left` for the position, and `width`, `height` for the size. There are additional properties such as `HorizontalPlacement` and `VerticalPlacement` which can be used to specify a certain resizing/repositioning behaviour in case the form is resized. `Color` and `Font` are also obvious properties, common to most components. The setting of controls is similar to other RAD environments such as MS-Access, Delphi, Visual Basic: they can be set in the object inspector, so this aspect will not be discussed here.

The Morfik IDE provides most common controls, many of them corresponding to HTML (form) elements:

TextLabel a simple label, which can be used to display a text.

TextEdit an edit control, useful for editing one or more lines of text.

Button a button control.

OptionButton a radio button. When used in group, only one of the group can be selected at a time.

CheckBox a checkbox control. Useful in case more than one item should be chosen.

ListBox a control for selecting one or more items from a list.

Combobox a control for selecting one item from a dropdown list.

Image a control to show an image.

FileUploader a control to upload a file to the server.

But there are also a series of non-HTML elements:

Rectangle to draw a bevel or filled rectangle in several styles.

Tabs a page-control (or tabbed control).

DateTimeEdit a special edit control, suitable for editing dates.

Container a container for other controls.

Subform much like a container, but is not visible untill it is filled with another form.

PaintBox A box in which can be painted. The painting happens on the server, not on the client.

Timer A timer control, which can be used for animation effects.

Each of these controls has additional properties, specific to the control; It would be beyond the scope of this article to describe them all.

Besides these controls, there are a number of controls which are Morfik wrappers around external controls or functionality:

Flash can be used to embed a Adobe Flash applet in the HTML page.

Zapatec there are several Zapatec controls: menu, calendar, tree, grid, which are actually controls implemented in Javascript and DHTML; they are accessible and manipulatable from within Morfik without any knowledge of Javascript.

Dojo Similar to the Zapatec controls, these controls provide a color palette and a date picker

Wysiwigeditor is a simple wordprocessor control which allows to edit rich text: the resulting text is a HTML formatted text.

Google maps Allows to use google maps in your application.

The number of controls is not very big, but they should be sufficient for basic applications, and more controls will be added in the future. There are various example applications on the morfik website which demonstrate how to write custom controls (a toolbar, tree, a menu and some others). These do require a knowledge of HTML and Javascript.

4 Basic client-side events

HTML provides a mechanism for attaching scripting functions to certain user events. The Morfik IDE wraps this feature in a mechanism which is similar to the ones used by Delphi and Visual Basic: Each event handler is a method of the form that contains the control.

The events are split up in 2 categories: server events and client events.

- Server events are triggered when the control is rendered, i.e. when the needed HTML for the control is sent to the web-browser.
- Browser events are events which are triggered on the client, which basically means that they are user-triggered events, such as a click, double click, keypress, mouse dragging etc.

Whenever an event handler is created in the Morfik IDE, the IDE creates the method in the right place: in the client code or in the server code.

A complete discussion of all possible events is beyond the scope of this article, but the names of the events are in general quite logical, a few exceptions aside.

To demonstrate the working of events, a small example form (called `Event1`) is made with 2 `TextLabel` controls on it (named `LClick` and `LClickToo`). The `OnClick` event of both labels is filled with code:

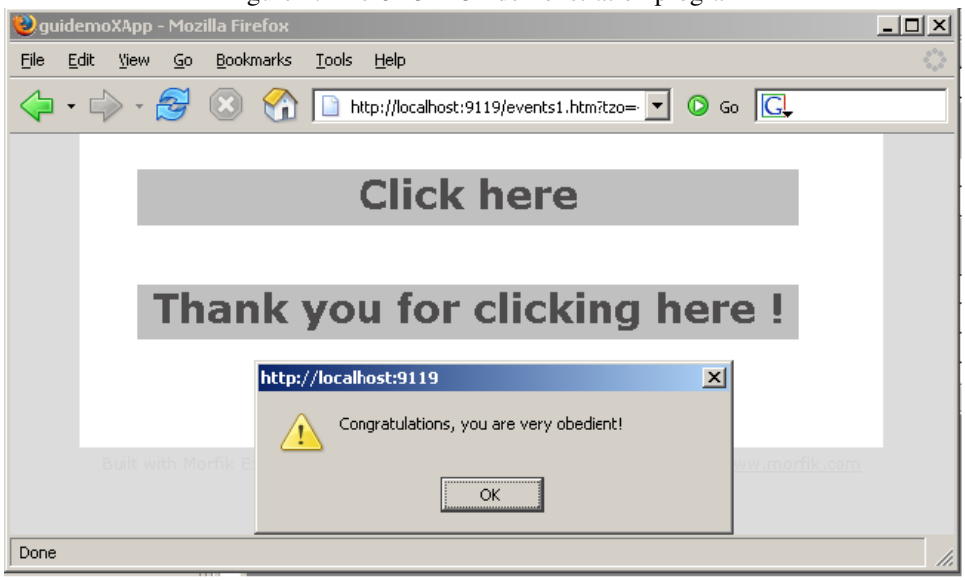
```
Procedure events1.LClickClick(Event: TDOMEvent);
Begin
  ShowMessage('Congratulations, you are very obedient!')
End;
```

The above code shows that it's easy to show a small message dialog to the user with the standard `ShowMessage` routine. It simply pops up a small dialog window displaying the text passed to it.

The second label has a slightly more elaborate `OnClick` handler:

```
Procedure events1.LClickTooClick(Event: TDOMEvent);
Begin
  If Not ClickTooClicked then
```

Figure 2: The `Onclick` demonstration program




```
    begin
    ClickTooClicked:=True;
    LClickToo.Caption:='Thank you for clicking here !';
    end;
End;
```

It shows 2 things:

1. The caption of the label can be set in an event handler: Generally, all properties available in the object inspector can be set at runtime, both in the client and on the server.
2. Variables can be declared as fields of the form class: the `ClickTooClicked` variable is a boolean variable, defined as a field of the form class.

Of course, both examples demonstrate what was said earlier: the above code is Object Pascal code, and runs on the browser. Obviously, the browser does not understand pascal, but the Morfik IDE translates this to Javascript code which runs in the browser (it does the same for the other supported languages). The result can be seen in figure 2 on page 6.

The above example shows that the properties of all controls can be manipulated in the event handlers. Sometimes, however, the names (or structure) of the properties change. A good example is the font property: in the Object Inspector, this is presented as a separate entity, with separate fields. In the client code, this is not so.

The following example (on a form called `Hover1`) shows what is meant. A label (named `LHover`) is dropped on the form, and the `OnMouseOver` and `OnMouseOut` events are used to create a kind of hovering effect for the label, by inverting the background and font colors:

```
Procedure Hover1.InvertColor;

Var
  C : TColor;
```

```
begin
  C:=LHover.Color;
  LHover.Color:=LHover.FontColor;
  LHover.FontColor:=C;
end;

Procedure Hover1.LHoverMouseOver(Event: TDOMEvent);
Begin
  InvertColor;
End;

Procedure Hover1.LHoverMouseOut(Event: TDOMEvent);
Begin
  InvertColor;
End;
```

In Visual Basic or Delphi, the font color would be specified as follows:

```
  LHover.Font.Color:=C;
```

Because the `Font` property is a class of it's own, with a `Color` property. This is not the case in the browser, there `FontColor` must be used.

Most of the events are demonstrated in the 'EventLogging' form, which is present in the demo application. All events are logged to a memo control, the maximum of information is displayed to the user.

It should be noted that many events receive a parameter which is defined more or less by the HTML specification: `TDOMEvent`. This is contrary to Delphi, where the first parameter is usually the control or component which triggered the event. This is not so in Morfik. The `TDOMEvent` is a class which exposes some properties that give information about the event. The same class is used for all events, but some fields will not give correct values when examined. Unfortunately, it's not documented which fields can be expected to have correct values. The 'EventLogging' can be extended to show all fields of the `TDOMEvent`, but this is left as an excercise for the reader.

5 Basic server-side events

The server side events are not meant to react on user actions. They are meant to change the appearance or the content of the form which will be sent to the user. There are not so much events. The form has only 2 events:

OnBeforeExecute Executed when the form is invoked from the browser. Further execution can be avoided by setting the `PContinue` parameter for this event to `False`.

OnAfterExecute Executed when all form data was sent to the browser.

Each control has 3 events:

OnBeforePrint This is executed before the HTML and CSS information for the control is generated. It has a boolean argument `Print`, which can be set to `False`, in which case the control will not be sent to the browser.

OnAfterPrint This is executed after the HTML and CSS information for the control is generated. Obviously, nothing can be changed to the layout of the control after it was sent to the client. It's suitable mainly for keeping track of state information.

OnPrintStyle This gives the programmer the opportunity to change the style elements for the control. The `Style` argument passed to the routine is the string containing the CSS style elements for the control: the programmer can change them in any way he sees fit.

Note that the order in which the controls are rendered is not guaranteed: this means that if the state of one control should be altered depending on some setting in another control, any code for this should go in the `OnBeforeExecute` handler of the form.

All this is demonstrated in the `Event2` form: A form is created with several textlabels on it. The server-side events of all controls are logged, and in the `OnBeforeExecute` event of the form, a global counter is updated, so the number of times the form is sent to the client can be shown:

```
var
  PrintCount : Integer;
  S : String;
  LastPrint : integer;

Procedure AddToS(ToAdd : String);

begin
  S:=S+LineEnding+ToAdd+' ('+IntToStr(printcount)+')';
end;

Procedure Events2.WebFormBeforeExecute(Sender: TWebDocument;
                                       Var PContinue: Boolean);
Begin
  Inc(PrintCount);
  LLog.Caption:=S;
  AddToS('Form: BeforeExecute');
End;
```

`PrintCount` is a global variable, which is increased with 1 with each invocation of the form. The string `S` is a global variable which contains a log of all events: the procedure `AddToS` simply adds a new line to this string, prepended with the value of the `PrintCount`. This procedure is used to log all events.

Note that the caption of the `LLog` textlabel is set with `S` in this routine. Since this happens at the start of the execution, only the log of the previous executions of the form will be shown (the log is not cleared between executions of the form). Setting the caption at the end of the execution would not have any effect: the client code for the `LLog` will already have been sent to the client.

Code like the above, using global variables on the server, is obviously not very safe: in a real-life situation, multiple threads will run concurrently on the server, and the access to `S` and `PrintCount` should be protected with a critical section.

Each form consists of a number of bands (more about this in the article about DB programming with Morfik), the `Header`, `Detail` and `Footer` band. These bands also have events. The `Detail` band has the following `OnBeforePrint` event:

```
Procedure Events2.DetailBeforePrint(Sender: TWebControl;
```

```
                                        Canvas: TWebCanvas;
                                        Var Print: Boolean);
Begin
  LCount.Caption:='This page has been printed '+
                  IntToStr(PrintCount)+' times';
  Print:=((PrintCount mod 5)<>0);
  AddToS('Detail beforePrint');
End;
```

In this code, the `LCount` caption is set to display the number of times the form has been displayed. But, every 5th call, the `Print` parameter is set to `False`, and the detail band will then not be printed. The last line logs the event.

The `OnBeforePrint` of the `LCount` textlabel is filled with the following code:

```
Procedure Events2.LCountBeforePrint(Sender: TWebControl;
                                        Canvas: TWebCanvas;
                                        Var Print: Boolean);
Begin
  Print:=((PrintCount mod 4)<>0);
  AddToS('LCount BeforePrint');
End;
```

Similar to the `Detail` band mechanism, the `LCount` textlabel will not be shown every fourth time the form is shown. The second line logs the event.

Lastly, the `OnBeforePrint` of the `LRandom` textlabel is filled with a simple event that sets the caption, which demonstrates that this is the last moment at which the caption can be set:

```
Procedure Events2.LRandomBeforePrint(Sender: TWebControl;
                                         Canvas: TWebCanvas;
                                         Var Print: Boolean);
Begin
  LRandom.Caption:='Random number in range [1..100]: '+
                   IntToStr(Random(100)+1);
  AddToS('LRandom BeforePrint');
End;
```

Each time the form is loaded, another random number will be shown.

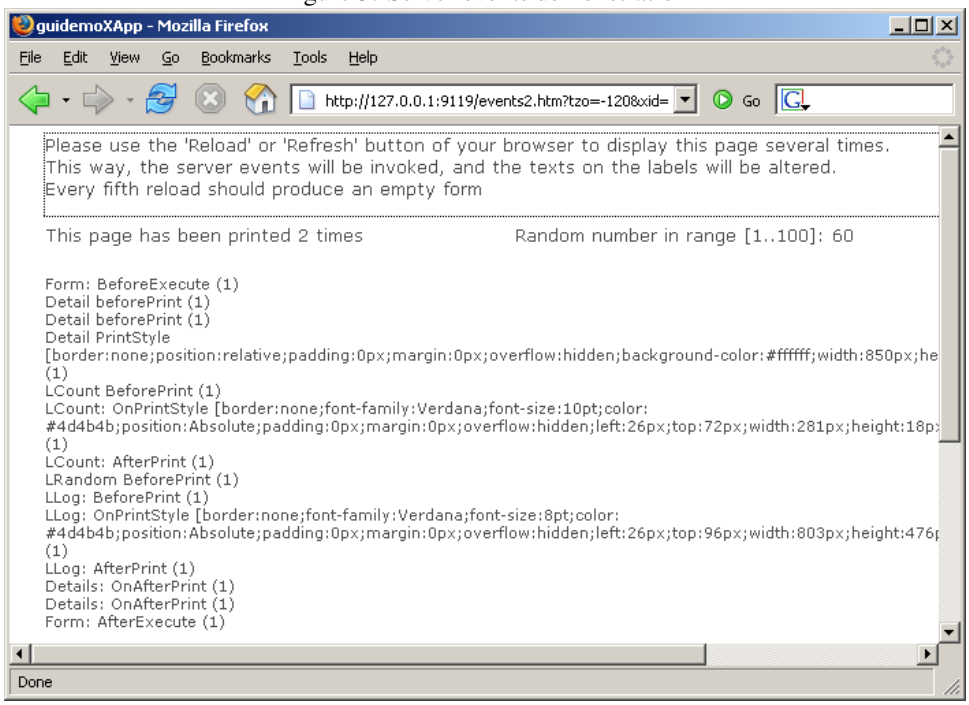
To see the effect of all this code, the form should be shown from the main menu, and then reloaded at least 5 times. Simply pushing the reload button of the browser will take care of this. The finished form is displayed in figure 3 on page 10.

6 Multiple forms

All code till now was limited to interaction with the user within a single form. No realistic application consists of a single form, so it's imperative that Morfik provides a mechanism to show multiple forms. It does this, and in many ways. The previous examples are all single-form examples and each form is shown from the main menu - below it will be explained how this is done.

When a Morfik application is invoked for the first time by a browser (a new session is started, so to speak), the main form of the application should be shown. In the apache

Figure 3: Server events demonstration



webserver the default page for a location is called 'index.html', similarly the main form of a Morfik application is called 'Index'. However, this is not mandatory: the name of the 'main' page can be set in the project options dialog.

To demonstrate the concepts in this section, a small form is created, called 'Dialog'. It has a distinctive color and border, so it will be easily recognizable when it's shown inside other forms or when it fills the complete browser window.

The first way to show a second form is to use a subform control and set it's `Form` property to the name of the form which should be dislayed in the subform control. This is exactly what happens in the `SubFormDemo` form: there is a `SFDemo` control, with it's `Form` property set to `Dialog`. At design time, the `Dialog` window is not shown in the subform control (only its name is shown) but at runtime, when the form is shown in the browser, the location of the `SFDemo` control is filled with the `Dialog` form.

While this use of a statically bound subform has it's uses – for instance to display a menu in the header or footer bands of all windows, or for a sidebar which is shown on all windows – most of the time, the content of the subform control will be set at runtime. This is done with what is probably one of the most important functions in the Morfik framework, the `OpenForm` procedure.

The `OpenForm` method of the `Form` class is defined as follows:

```
Function OpenForm(FormRef,Target,OpenFormParameters : String);
```

It has 3 arguments:

FormRef This argument specifies which form should be opened. It can also be an arbitrary URL: this can be used to show the contents of an arbitrary URL.

Target This argument specifies where the form should be shown. In general this argument is of the form `FormName:SubFormName`, meaning that the form will be

opened and shown inside the form `FormName`, in the SubForm control with name `SubFormName`. There are several special names possible, they will be shown below.

OpenFormParameters This argument allows to specify parameters which should be passed to the form, or parameters which determine how the newly opened form behaves.

The `SubFormDemo2` form is equal to the `SubFormDemo` demo, except that it shows the `Dialog` form in the subform control only when a button is clicked. The `OnClick` handler of the button has the following code:

```
Procedure SubFormDemo2.BShowClick(Event: TDOMEvent);

Begin
  Openform('Dialog','Self:SFDemo','');
End;
```

This code above illustrates a simple form of the `OpenForm` call: the `Dialog` form is opened and shown in the current window (designated by the`Self` name), in the subform control named `SFDemo`.

The `FormRef` argument of the `OpenForm` call can be the name of a form in the application, but it can also be an arbitrary URL. To demonstrate this, a `OpenURL` demo form is made, in which an edit control allows the user to enter an URL. A click on the 'Go' button will then load the URL in the subform container below the edit control. The `OnClick` handler of the button contains the following code:

```
Procedure OpenURL.BGoClick(Event: TDOMEvent);
Begin
  OpenForm(EURL.Text,'Self:URLWindow','');
End;
```

The result can be seen in figure 4 on page 12.

The `Target` argument to the `OpenForm` procedure is generally of the following form:

```
FormName:SubFormName
```

in which `FormName` can also have one of the following special names:

self The subform control is searched in the currently open window.

parent The subform control is searched in the parent window of the current window. This is useful if the current window itself is located in a subform control.

top The subform control is searched in the topmost window.

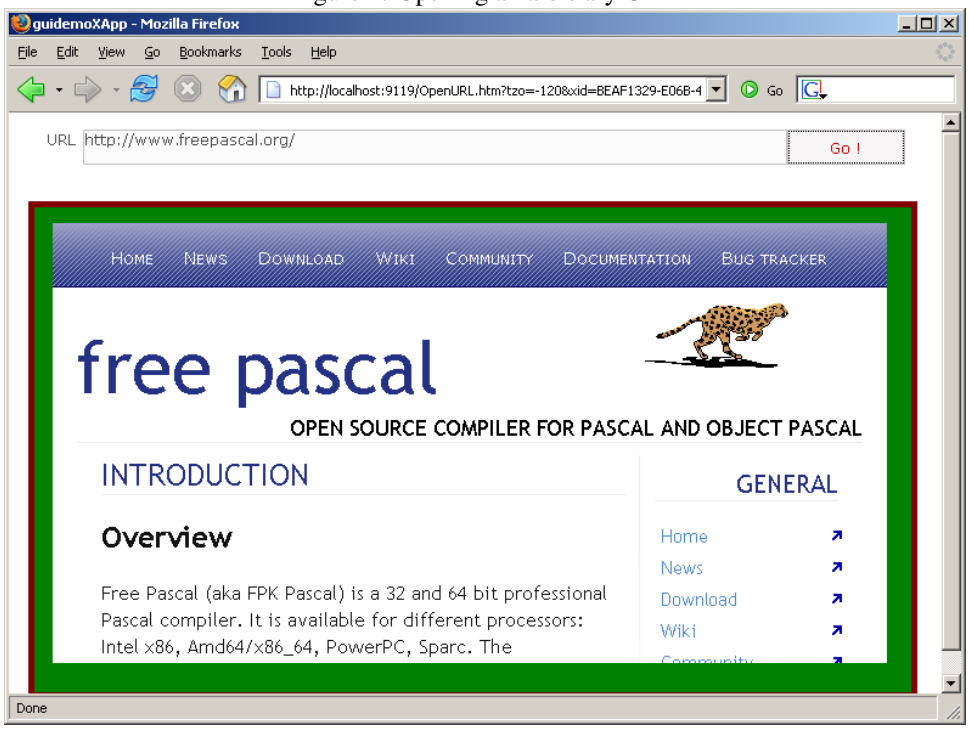
float No subform control is searched. Instead, a floating window is created, and the form is shown in that window. This can be used to create modal or floating dialogs.

blank The form is opened in a new browser window.

same The form is opened in the same browser window as the current form.

In the last 2 cases, a new session is started. All other forms of this call keep the current session. The `:SubFormName` part of the `Target` argument is actually optional. This can be seen in the main form of the demo application. In the main form, the menu is displayed, and the selected form is opened with the following call:

Figure 4: Opening an arbitrary URL




```
Procedure Index.ShowForm(FormName : String);

begin
  OpenForm(FormName,'top','');
end;
```

Since the menu window is the top-most window, the effect of this call is that the top-most window is replaced with the desired window.

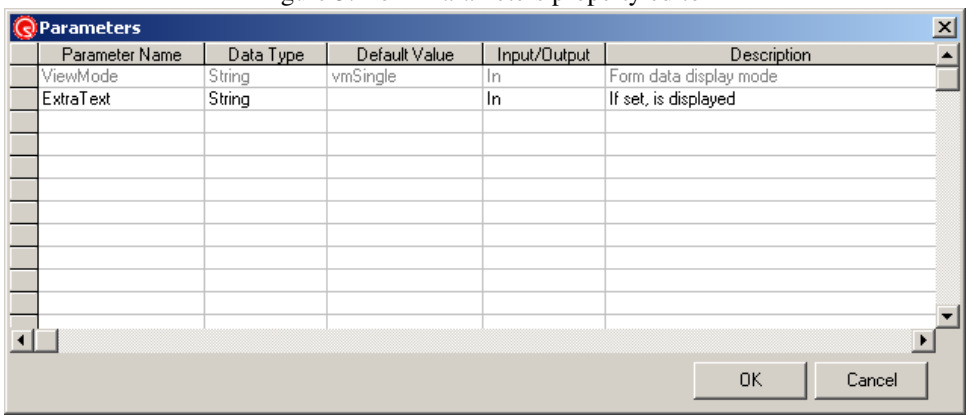
7 Form parameters

The last argument of the `ShowForm` call is used to pass parameters to the opened form. This argument serves many purposes. The primary purpose is to pass form-specific parameters: each form in a Morfik application can have a number of parameters associated with it, which must be defined in the `Parameters` property. When the `Parameters` property of the form is clicked, the parameter editor dialog is opened (see figure 5 on page 13). For each parameter that the form should be able to receive, a name, data type, initial value, type and comment can be entered in this dialog.

The parameters that are defined in this way, are accessible in the code as fields of the form class (although oddly enough they are not defined as such in the form class definition). This can be illustrated with the `Dialog` form which is used throughout the application. In this form a string parameter with name `ExtraText` is defined (see figure 5 on page 13). An extra textlabel is dropped on the `Dialog` form (named `LExtraText`), with it's `Visible` property set to `False`, and the following code is entered in the `OnShow` event of the form:

```
Procedure Dialog.WebFormShow(Var Show: Boolean);
```

Figure 5: Form Parameters property editor




```
Var
  B : Boolean;

Begin
  B:=(ExtraText<>'');
   If B then
     begin
     LExtraText.Caption:=ExtraText;
     LExtraText.Visible:=True;
     end;
   Show:=True;
End;
```

The first line checks of the `ExtraText` field is not empty. If it is not empty, then it is stored in the `Caption` property of the `LExtraText` textlabel, and the `Visible` property is set to `True`. The effect of this is that if an non-empty `ExtraText` parameter is passed to the form when it is shown, then the `LExtraText` textlabel will become visible, and will display the value of the parameter. Otherwise the textlabel remains invisible.

To demonstrate this, a `ParamsDemo` form is coded, which allows to enter a text in a textedit. As soon as the user hits a button, the dialog form is shown, and the text that the user entered in the edit control is passed as the value for the `ExtraText` parameter in the `OpenForm` call. This is achieved with the following code:

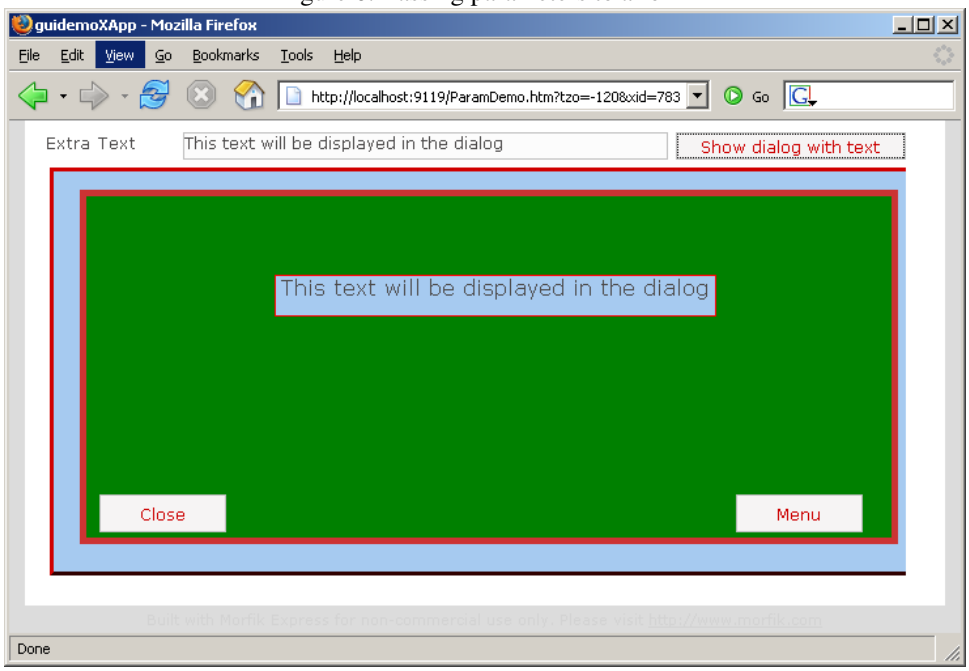
```
Procedure ParamDemo.BShowClick(Event: TDOMEvent);

Var
  S : String;

Begin
  S:='';
  If (EParam.Text<>'') then
    S:='"ExtraText='+Eparam.Text+'"';
  OpenForm('Dialog','self:sfparam',S);
End;
```

As can be seen, parameters to the form must be passed as a comma-separated list of

Figure 6: Passing parameters to a form



`"Name=Value"`

pairs. The result of this code is visible in figure 6 on page 14

The concept of form parameters is very important in Morfik, and will be discussed intensively when database access is discussed. However, form parameters are also used to modify the behaviour of a form when it is opened as a floating window. A form can be opened as a floating window when the `Target` parameter for the `OpenForm` call is set to `Float`. In that case, there are a number of predefined parameters which affect how the floating form behaves when it is shown. The following parameters are possible:

modal is the popup form modal or not. If it is modal, all other forms are disabled (grayed out) while the popup form is shown. This is a boolean value.

draggable can the popup form be dragged around the screen or not.

closable if `true`, then there is a small button on the border with which the floating window can be closed.

border style of the border of the popup form. This can be empty or `rounded`.

transition Transition effect when the popup form. This can be set to `overlayfadein`.

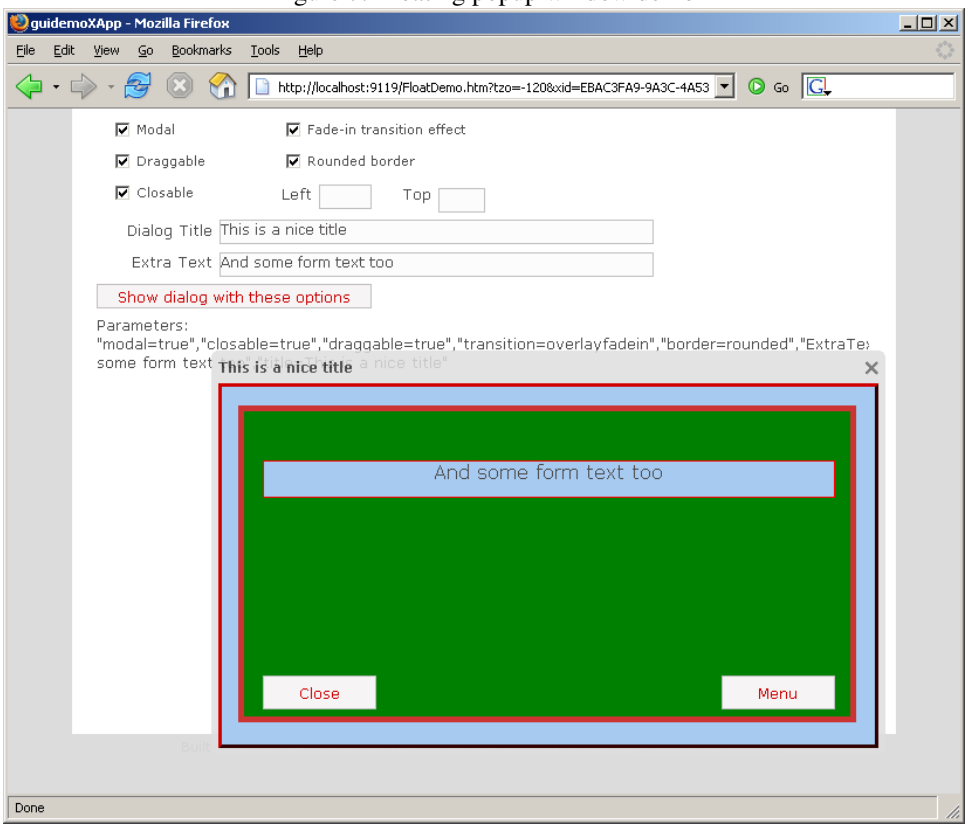
title The title which will be shown in the border of the popup form.

top A top coordinate (in pixels) measured from the top-left corner of the browser window.

left A left coordinate (in pixels) measured from the top-left corner of the browser window.

The `FloatDemo` window allows to set these options through some controls (checkboxes). There is a button which, when clicked, collects all parameters by examining the various settings and then opens the `FixedSizeDialog` window as a floating window. The `FixedSizeDialog` form is a copy of the `Dialog` form, but instead of a varying width

Figure 7: Floating popup window demo



and height – to accomodate for the size of the subform control in which it can be shown – it has a fixed width and height. This is necessary, beacuse otherwise it will be shown with a default width and height, which gives some rather strange effects.

The collected parameters are shown in a text label, so they can be studied. The result is shown in figure 7 on page 15.

A special parameter name is also `OpenMode`: if it is set to `DONOTACTIVATE`, then the form will be downloaded in the browser, but will not be shown. The second time the same form will be opened, it will no longer need to be downloaded: it'll be in the cache of the subform control in the browser.

Since the 'OpenForm' call is asynchronous - like all AJAX calls it is executed in the background - this can be used to prefetch forms: while the user is busy doing things, forms can already be downloaded in the background. When the user needs the form, they will be shown instantly. Obviously, this will hugely improve the responsiveness of the application.

8 Conclusion

In this article, the basic workings of a Morfik application have been examined: it was shown that a Morfik application can be programmed in any of the supported Morfik languages, and that the event model used is similar to the ones used in Visual Basic, Delphi or MS-Access. The working of server and client events was demonstrated, as well as the use of parameters in forms. All this can now be put to use in a real-life application with a database. But this is the subject of another contribution.