Practical GUI construction in ExtJS

Michaël Van Canneyt

March 16, 2014

Abstract

In previous Toolbox articles, the ExtJS toolkit was introduced. This article continues the acquaintance with ExtJS and shows how to program simple GUI applications with practical examples.

1 Introduction

ExtJS was introduced in [**?**]. A project to generate ExtJS applications from a Pascal webserver module has also been presented in [**?**]. The library has just been updated and is currently at version 3.2 – high time to present it in a little more detail.

In this article, the GUI components of ExtJS will be explored in more detail: ExtJS offers a lot of GUI components which will render on the popular browsers: Internet Explorer,Firefox, Safari and Opera.

The GUI library is consistently structured, yet very versatile, offering an event mechanism where lots of hooks can be installed to provide customized behaviour when needed: the event mechanism was described in [**?**], as well as the class construction mechanism it uses.

In this article, the practical details of creating GUI webapplications in ExtJS will be explained in more detail, with sample code to explain the process. The sample files can be used without the need for any webserver software: a simple browser is enough.

2 Getting started

Before one can start programming ExtJS, first a copy of the extjs installation (a simple archive) must be downloaded, and unpacked in a location where the webserver (or the browser) can find it, for instance in:

```
/svr/www/htdocs/extjs-3.2.0
```

The archive contains both the JavaScript library sources and the documentation - in HTML.

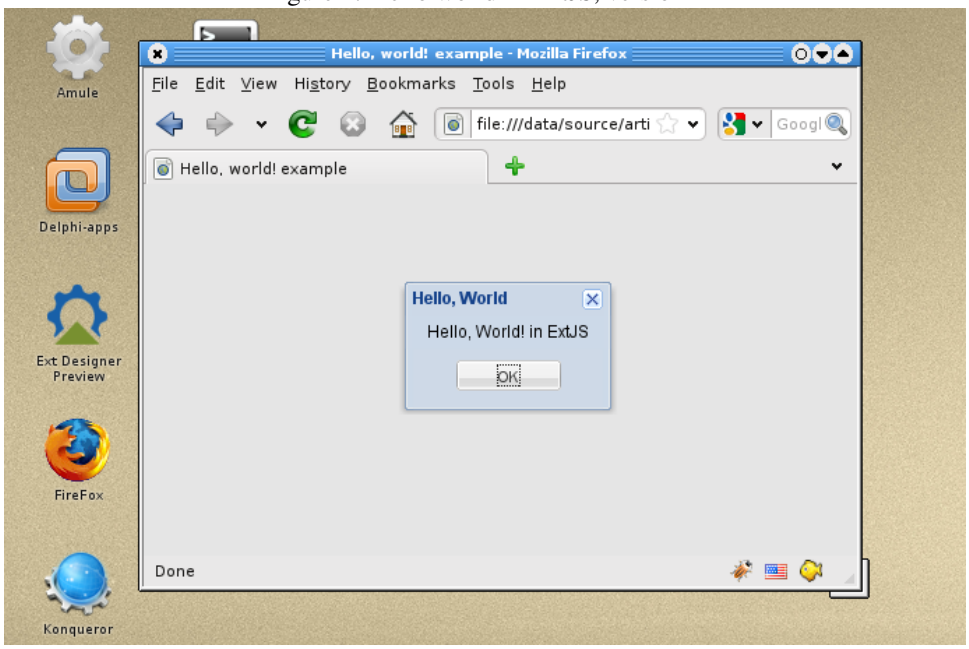
In Apache, it is advisable to provide an alias in the Apache configuration file:

```
Alias /ext/ "/svr/www/htdocs/extjs-3.2.0/"
```

Using an alias allows to install multiple versions at the same time and to change to a new version of extjs without having to change all references to the library location.

In this article, no webserver software is used, and the ExtJS library will be assumed to reside in the same directory as the sample applications.

Figure 1: Hello world in ExtJS, version 1



3 Hello world in ExtJS

Once ExtJS is installed, a simple test HTML file can be written to see whether the library was succesfully installed:

```
<html><head>
<title>Hello, world! example</title>
<script type="text/javascript"
        src="/ext/adapter/ext/ext-base.js"></script>
<script type="text/javascript"
        src="/ext/ext-all.js"></script>
<link rel="stylesheet" type="text/css"
     href="/ext/resources/css/ext-all.css" ></link>
</head><body>
<script type="text/javascript">
Ext.onReady(function(){
  Ext.MessageBox.show({
   msg: "Hello, World! in ExtJS",
   title: "Hello, World",
   buttons: Ext.MessageBox.OK
  });
});
</script>
</body></html>
```

The example demonstrates the use of the `onReady` function, as explained in [**?**], When loaded in the browser, something as in figure 1 on page 2 will be displayed.

Besides the mandatory HTML code of any web-page, several `script` tags and a `link` tag are needed to load the ExtJS library. ExtJS always needs at least 2 script files to be able to run:

- The first script tag loads the adaptor code: this is some JavaScript code that provides the glue between the HTML and server calls on the one hand, and the ExtJS library on the other hand. ExtJS provides a native adaptor, adapter/ext/ext-base.js. ExtJS also provides adaptors for other Javasript frameworks such as `jQuery` or `Prototype`, for people that already use these frameworks. They are located below the adapter directory.
- The second script tag loads the ExtJS library proper. This is a relatively large file (640K): for slow sites, it may be advisable to put it on a server with a fast internet connection (such a cachefly) and to refer to this site instead.
- The `link` tag loads the default styles for all ExtJS elements: The default stylesheet is located in resources/css/ext-all.css. It can be copied and changed at will to provide a different look and feel: indeed, several alternative versions of this file exists, each implementing a different theme.
- The last `script` tag contains the actual code. For the sake of convenience, it has been put directly in the HTML file, but it might just as well have been a reference to a separate javascript file.

The "Hello, World!"' script is quite simple but at the same time instructive:

```
Ext.onReady(function(){
  Ext.MessageBox.show({
   msg: "<b>Hello, World!</b> in ExtJS",
   title: "Hello, World",
   buttons: Ext.MessageBox.OK
  });
```

The first thing to note is that all of ExtJS is implemented in the `Ext` namespace: as explained in [**?**], all components reside in this namespace.

One of the objects in the `Ext` namespace is the `MessageBox` object, which can show various message dialogs on the screen. The `show` function is the most general variant of the dialogs.

Even this simple example demonstrates a second point made in [**?**], namely: like most (if not all) ExtJS functions, it accepts a configuration object as the sole argument of the function. The configuration object should contain values for various settings of the dialog.

There are far too many configuration values to be discussed here: the ExtJS documentation specifies for all configuration objects the allowed members. In case of the above `Ext.MessageBox.show` function, 3 members have been specified:

msg The message to display. Note that this can contain HTML formatting tags: most display texts are simply inserted in the HTML tree, and as such they can contain HTML tags which will be rendered correctly.

title the title of the dialog box.

buttons A button configuration object. There are many pre-defined values for this property: OK, OKCANCEL, YESNO, YESNOCANCEL.

Obviously, ExtJS code does not replace the regular HTML page. It can be integrated in existing pages. The following code puts the 'Hello, world' code behind the 'OnClick' handler of a button in a HTML page:

```
<body>
<h1>"Hello world!" behind a button</h1>
Clicking the following button will show the "hello, world!" dialog:<p>
<center><button id="mybutton">Do it</button></center>
<script type="text/javascript">
function myfunction (e) {
  Ext.MessageBox.show({
     msg: "<b>Hello, World!</b> in ExtJS",
     title: "Hello, World",
     buttons: Ext.MessageBox.OK
  });
}
Ext.onReady(function(){
  Ext.get('mybutton').on('click', myfunction);
});
</script>
</body>
```

For clarity, the 'OnClick' callback has been put in a separate function. This example demonstrates 2 important techniques in ExtJS:

1. The 'Ext.Get' function is the principle glue between the HTML page and the ExtJS code: it retrieves a DOM tag from the HTML page by name: 'mybutton' in the example. The function returns an `Ext.Element` object: this wraps a DOM element in a Javascript class, which allows to manipulate the DOM element.
2. The 'on' method of varExt.Element adds an event handler to the DOM element: in this case, the 'OnClick' handler of the button.

The result is shown in figure 2 on page 5

Unlike the claim in the introduction of [**?**], the use of the `Ext.onReady` function is not mandatory, although it is fair to say that in everyday ExtJS use, the function will be the starting point of the ExtJS code. A variant on the above method of adding event handler to HTML elements shows that ExtJS code can be called simply from the HTML tags:

```
<body>
<h1>"Hello world!" behind a button</h1>
<script type="text/javascript">
function myfunction (e) {
  Ext.MessageBox.show({
     msg: "<b>Hello, World!</b> in ExtJS",
     title: "Hello, World",
     buttons: Ext.MessageBox.OK
  });
}
</script>
Clicking the following button will show the "hello, world!" dialog:<p>
<center><button id="mybutton" onclick="myfunction();">Do it</button></center>
</body>
```

4 The Ext.Window class

The `Ext.Window` object defines a complete window class: the window acts as a window of a regular desktop application, except that it is rendered in the window of the browser: it

Figure 2: Hello world in ExtJS, version 2

is created completely in HTML, as explained in [**?**]. It is the equivalent to the `TForm` class in Delphi or the `Form` class in .NET's System.Windows.Forms library.

The following variant of the "hello, world" example creates a complete window by itself, instead of relying on the built-in MessageBox object. The ExtJS code to show a window looks as follows:

```
Ext.onReady(function(){
    var helloworld = new Ext.Window({
    title: "Hello, World",
    html: "<center><b>Hello, World! in ExtJS</b></center>",
    layout: 'fit',
    plain : true,
    closable: true,
    width: 200,
    height: 100
   });
   helloworld.show(this);
});
```

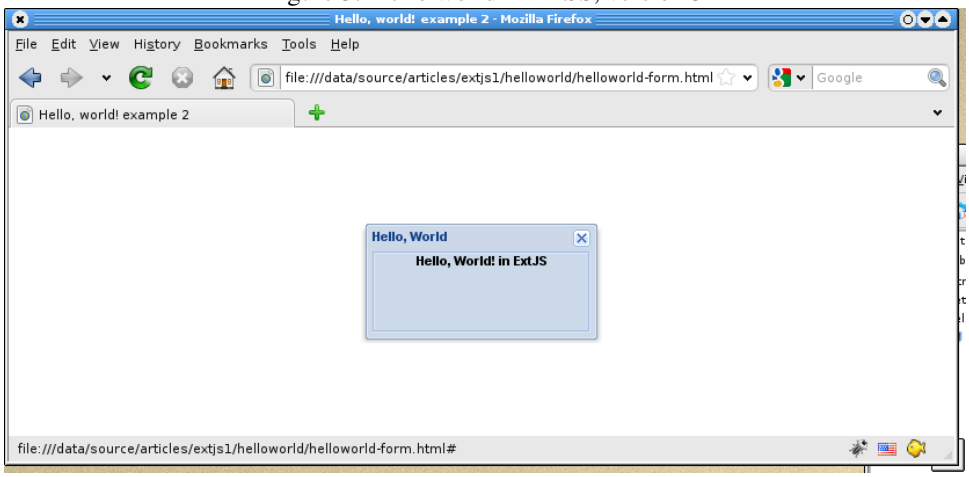
The result of this code is shown in figure 3 on page 6. Let us look at some of the configuration parameters of the newly created window:

title the text displayed in the title bar.

html the text displayed inside the window. This text can contain HTML markup, but can also be empty, if the window will contain other ExtJS GUI components as children.

layout The layout to use for widgets that will be displayed in the form.

Figure 3: Hello world in ExtJS, version 3



plain Determines the background for the window: when 'true', the window background will have the same color as the border. If false, then the background will be slightly lighter.

closable if 'true', then a close button will be shown in the window title bar, allowing to close the window.

width & height the dimensions of the window.

There are about 100 configuration options for `Ext.Window`: too many to be discussed in the context of an article, but the above properties will be found in most window constructors.

The 'show' method of `Ext.Window` does what it says: it shows the window in the browser: it will create the HTML DOM nodes to show the window in the browser (a process called 'rendering' by the ExtJS people). The generated DOM tree with the HTML for the window can be inspected with a tool such as firebug: firebug is an add-on to Firefox, and is an invaluable aid when debugging ExtJS applications (or indeed any Javascript application in a browser).

The screenshot in figure 4 on page 7 shows the generated DOM elements for the "Hello, world" window.

5 Control classes

Obviously, an empty window with some HTML in it is not sufficient to make a web-application. ExtJS has an extensive class tree, and provides a lot of simple controls (these classes are part of the global Ext object):

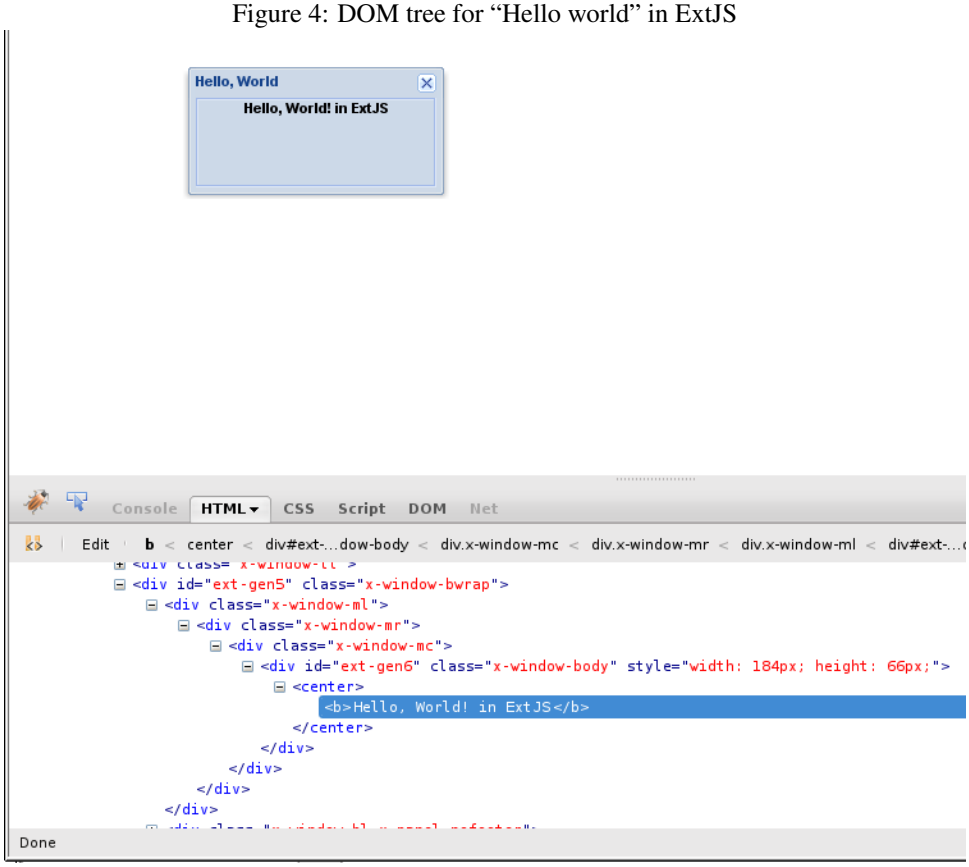
Button A simple button. It can contain text and an icon, and a menu can be appended to it.

CycleButton A button with a menu.

ColorPalette A color picker, displaying a panel to pick a color.

DatePicker A date picker component

Figure 4: DOM tree for “Hello world” in ExtJS



Editor A simple editor component.

Panel A panel which can contain other controls, and which has a title and footer section: these sections can contain buttons or other controls, allowing to construct for example a menu.

ProgressBar A progress bar. It can contain progress text, percentual display and can be rendered in various style.

Slider A slider component (a trackbar) which can be horizontal or vertical, uses a step value or not. It can be completely styled.

SplitButton This is a special variant of a button, it has a section which can be used to invoke a menu.

Toolbar The toolbar component, as it exists in many GUI widgetsets.

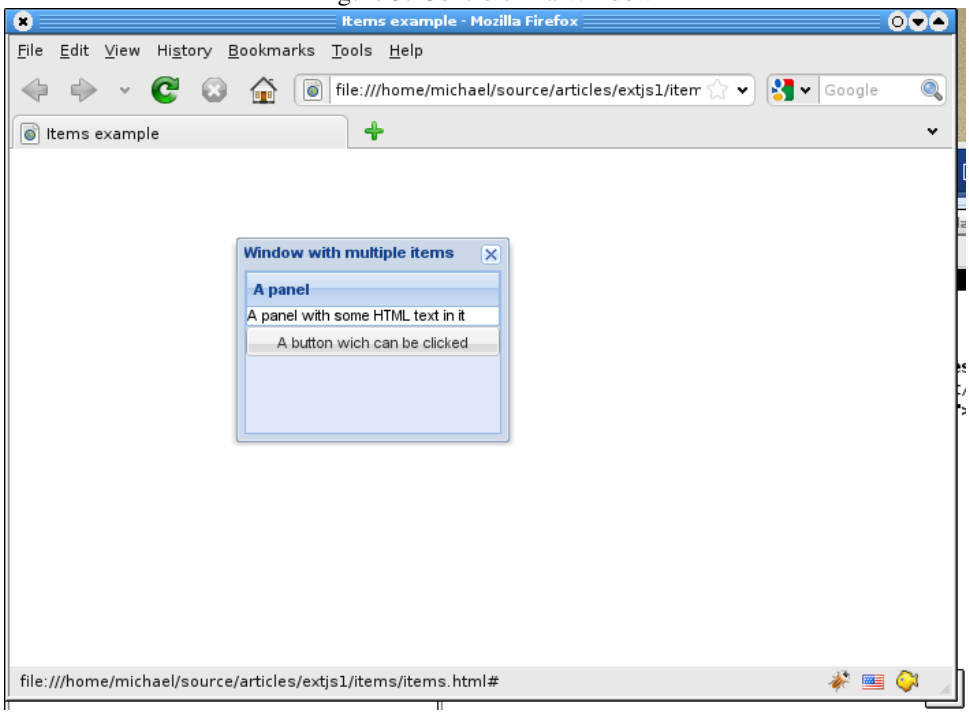
These controls can be created as children of a `Ext.Window` instance, but can just as well be rendered as part of the HTML page.

To put controls in a `Ext.Window` instance, they must be part of its `items` configuration option. The `items` option is very important throughout all of ExtJS GUI classes: all controls in the `items` property will have the window as their parent. The `items` configuration option is introduced in `Ext.Container`, which is the parent class for controls that can contain other controls: `Ext.Panel` or `Ext.Window` are controls which can contain other controls.

To show how to put controls in a window, the following example is instructive:

```
function RenderWindow() {
  var mypanel = new Ext.Panel({
    title: "A panel",
    html: "A panel with some HTML text in it"
  });
  var mybutton = new Ext.Button({
    text: "A button wich can be clicked",
    handler: function() {
       Ext.MessageBox.show({
         msg: "<b>You clicked the button!</b>",
         title: "Button click handler",
         buttons: Ext.MessageBox.OK});
    }
  });
  var mywindow = new Ext.Window({
     title: "Window with multiple items",
     layout: 'vbox',
     layoutConfig: { align: 'stretch'},
     plain : false,
     closable: true,
     width: 200,
     height: 150,
     items : [mypanel,mybutton]
    });
    mywindow.show(this);
}
Ext.onReady(RenderWindow);
```

Figure 5: Controls in a Window



Once more, the `onReady` event is handled by the `RenderWindow` function. This function creates a panel and a button; While neither does something interesting, the `handler` property in the button configuration object deserves mention: it can be used to specify a function which must be executed when the button is clicked: In the case of the example, it shows a simple message using `Ext.MessageBox.show`, which was introduced earlier.

The interesting part comes when the window is created. There are 3 properties which cooperate:

items this property should contain an array of ExtJS controls (if there is only 1 control, the control itself may be specified instead of an array). In this case the panel and button are put in an array.

layout This is a string which contains the layout to use, or it can be an object which descends from `Ext.layout.ContainerLayout`, and which will handle the layout of the controls in the container (in this case, `mywindow`). Some standard layouts are presented below.

layoutConfig in case the layout was specified as a string, this property can contain a layout configuration object, which will be used when the container creates the layout object.

The result of all this code is shown in figure 5 on page 9.

The following layouts exist by default in ExtJS:

absolute each control is positioned on the x,y coordinates specified in its configuration.

accordion should be used only when the controls are panels: only 1 panel is expanded at a time, all other panels are 'collapsed': only their title bar is visible. This is a layout as used in the outlook bar.

anchor all controls are rendered as if they were anchored to the sides of the container: this means that their measurements are specified as a percentage of the container's height or width.

border with border layout, the container is divided in 5 regions: north, south, east, west, and center. Each control in the container should specify one of these 5 regions in their `region` configuration property, and will be placed in the specified region of the control. The measurement of the control in the 'other' direction (for instance 'height' for a control in the 'north' region must be specified in the control's configuration.

card with card layout, only one of the containers is visible at a time. This is a layout used by the 'TabPanel' control.

column column layout will line the controls up in columns, one control per column.

fit Used for a single control, which will always get the size of the container.

form form layout is used for forms (discussed below).

hbox all controls are arranged horizontally in the container, in the order that they appear in the items list and using the specified width of the control.. This layout has some configuration properties to control the spacing and the vertical size of the controls.

table All controls are arranged in a table, once control per table cell. The table's column count must be specified, and the controls will be put in a cell, till the column count is reached, after which a new row is started. Each control can specify a 'rowspan' or 'colspan' configuration property, which will make the control span several rows or columns, as in a HTML table.

vbox all controls are arranged vertically in the container, in the order that they appear in the items list, using the specified height of the control. This layout has some configuration properties to control the spacing and the horizontal size of the controls.

When specifying one of these layouts, the appropriate descendent class of Ext.layout.ContainerLayout is created automatically to handle all layouting.

6 The use of Automatic component creation

The mechanism to create a layout based on a string description is used in another location as well: all controls descend from `Ext.Component`, and ExtJS maintains a component factory (`Ext.ComponentMgr`), That means that each known control class is registered with the component factory using a unique name, the so-called `xtype`, mentioned in [**?**]. This name can be used to instantiate an instance of the component, when a control must be added to a container's `items` list. For instance, the 'Ext.Button' class presented above is known as 'button'. The 'Ext.Panel' class is known as 'panel'. (a full list of names is presented in the ExtJS documentation of `Ext.Component`).

What does this mean ? Above, it was said that the 'items' property must be an array of instantiated controls. Instead of this, it is also possible to provide an array of object literals, which will be regarded as configuration objects for controls that must be created by 'Ext.Container'. Each object literal can have an 'xtype' property, which will be used to create the control from the component factory. If no 'xtype' property is specified, a default type will be used: which default this is, depends on the container class.

Let's clarify this by rewriting the previous example using the automatic component creation technique:

```
function RenderWindow() {
  var mywindow = new Ext.Window({
     title: "Window with multiple items",
     layout: 'vbox',
     layoutConfig: { align: 'stretch'},
     plain : false,
     closable: true,
     width: 200,
     height: 150,
     items : [{
        xtype: "panel",
        title: "A panel",
        html: "A panel with some HTML text in it"
      },{
        xtype: 'button',
        text: "A button wich can be clicked",
        handler: function() {
          Ext.MessageBox.show({
            msg: "<b>You clicked the button!</b>",
            title: "Button click handler",
            buttons: Ext.MessageBox.OK});
        }
      }]
    });
  mywindow.show(this);
}
Ext.onReady(RenderWindow);
```

The resulting HTML in the browser is identical. This technique is used extensively throughout the whole of ExtJS, and makes for very concise and compact code: the default `xtype` class per container is quite logically chosen, which makes it often possible to skip the `xtype` property alltogether. If it is necessary to refer to a control in code, then this technique cannot be used, in that case there are no variables (such as `mypanel` or `mybutton`) with the created items.

7 Form controls

The observant reader will have noticed that the list of controls presented earlier does not contain any controls which show information or data (or allow to edit them). This is because these controls are treated specially by ExtJS. They are defined in the `Ext.form` namespace: All controls that can show or contain information which can be sent to the web server, are in this namespace: the `Ext.form` namespace is used for everything related to HTML forms: the `Ext.form.BasicForm` class encapsulates the HTML `form` element, and is used in the visual `Ext.form.FormPanel` class (an `Ext.Panel` descendent).

Practically, this means that whenever data should be entered by the user, and sent to the browser, an `Ext.FormPanel` control should be used: it is a `Ext.Panel` descendent which has some special options for button bars at the top and bottom, and which layouts its controls in a special way.

The typical controls used in the `Ext.FormPanel` correspond to many of the standard HTML form input elements, and are listed below:

Checkbox A regular checkbox, for displaying boolean values (use the checked property

for this).

CheckboxGroup A group of checkboxes. Each item is a checkbox and can be checked or unchecked separately.

ComboBox a dropdown box, offering a choice of texts.

DateField an edit field with a button to select a date.

DisplayField a simple read-only field.

Hidden a hidden field.

HtmlEditor a simple HTML editor.

Label a label to display values.

NumberField an edit field that only allows to edit numerical values.

Radio a radio button.

RadioGroup a group of radio buttons, each item is a `Radio` control, and only 1 can be checked.

SliderField A slider, the field value is the slider position.

TextArea A simple text editing area (a memo).

TextField A simple, one-line edit control.

TimeField A edit control to edit a time value, a button can be used to select a time.

All these controls descend from 'Field', which introduces the following configuration properties:

fieldLabel a label to be displayed in front of the control.

labelSeparator a text which is displayed between the control and 'fieldLabel'.

name the name of the field: this is the name under which the field value will be submitted to the webserver.

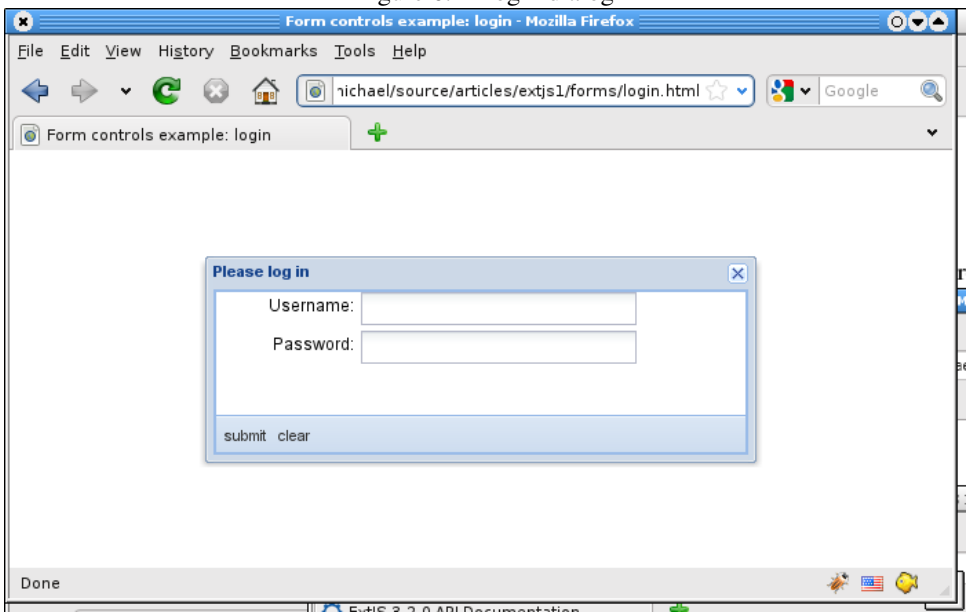
readOnly if the field is read-only, i.e. cannot be modified.

submitValue can be set to false: in that case, the field value will not be submitted to the webserver with the other valyes.

The following example shows how to use this information to :

```
function ShowLogin() {
  var LoginWindow = new Ext.Window({
     title: "Please log in",
     layout: 'fit',
     width: 400,
     height: 150,
     items : [{
        xtype: "form",
        labelAlign: "right",
        items: [{
            xtype: "textfield",
```

Figure 6: A login dialog




```
            fieldLabel: "Username",
            width: 200,
            height: 24
          }, {
            xtype: "textfield",
            fieldLabel: "Password",
            inputType: "password",
            width: 200,
            height: 24
          }
        ],
        bbar: [
          { text: 'submit' },
          { text: 'clear' }
        ]}
      ]
    });
    LoginWindow.show(this);
}
Ext.onReady(ShowLogin);
```

Besides demonstrating the creation of a form, the example shows how the labels in front of the edit fields are aligned right by the `labelAlign` property. It also shows that the 'inputType' property of the 'TextField' control can be used to create a password-entry. Finally, it shows that the `bbar` ("Bottom Bar") property of `Ext.Form.FormPanel` functions similar to the 'items' property: it should contain a series of buttons (or constructor configurations for buttons), which will be put in the bottom toolbar. The `tbar` property of `Ext.Form.FormPanel` is similar to `bbar` but is shown at the top of the form. The resulting dialog window is shown in figure 6 on page 13

8 Conclusion

Much more can be said about ExtJS GUI construction. in this article only the basics have been demonstrated: how to use ExtJS to create a basic GUI, which is rendered in the browser. Many topics are still untouched: Layouting is a very important part of ExtJS. Much can be said about customization of the look and feel using CSS. Not all controls have been treated, and form submission also has not been shown. Working with data, grids and templates for rendering data is also a wide area in need of attention, and finally the possibilities for RPC communication between the GUI and the webserver nshould be covered as well. Some of these things we hope to present in a future contribution.

References