ExtPHP

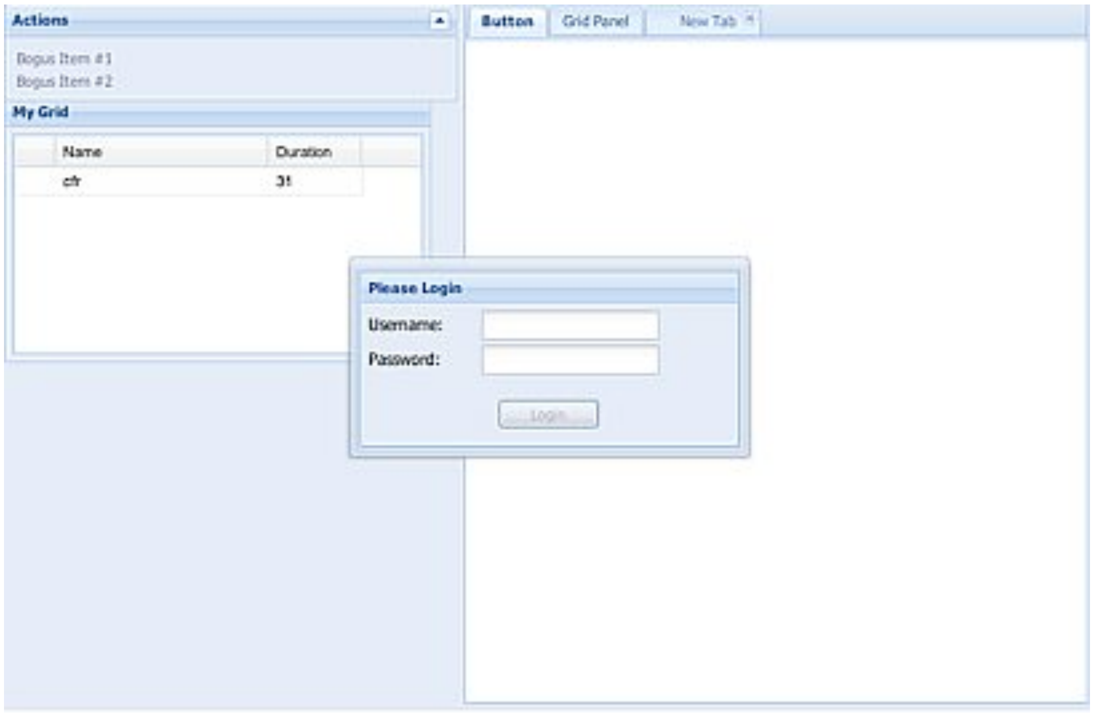
V0.1 “Technology Preview”

User Guide

Contents

About ExtPHP

ExtPHP is a wrapper for ExtJS. This is version 0.1, so I expect that a lot of things can be improved upon and your feedback is greatly appreciated. ExtPHP can be used to write both intrusive and non-intrusive Javascript, just like ExtJS itself. Use it responsibly. One of the many advantages of this design is that unknown/misspelled/misused methods are detected in your PHP editor rather than forcing you to debug your JavaScript code in your web browser.



Name	Duration
cfr	31

Converting ExtJS

Run **convertextjs.php**

1 You can modify the *DEBUGLEVEL* constant from 0 (no logging) to 5 (insanely verbose).

2 The script will look in your **docs/extjs** directory and generate the PHP library based on the content of ExtJS' documentation.

3 The generated PHP classes can be found in **libs/extphp**

4 All JavaScript declarations are converted to PHP classes following these rules:

- All dots are replaced by underscores
- The class name is fully qualified, including all package levels. For instance, *Ext.Toolbar.SplitButton* becomes *Ext_Toolbar_SplitButton*
- Additionally a configuration class, to be passed to the class' constructor, is created with the *_Config* suffix.

  For instance: *Ext_Toolbar_SplitButton_Config*

Note: The script is very generic but:

- It may break with newer releases of ExtJS, requiring minor adjustements,
- This is still an early release and some things are not fully implemented yet, such as default values and multiple arguments types (eg 'String/Object')

Instantiating an ExtJS Component

First, let's create a configuration object:

```
$wincfg = new Ext_Window_Config();
```

Now we can create our component class and pass the configuration object to it:

```
$wincfg->layout('fit')->width(300)->height(150)->closable(false)->resizable(false)->plain(true)-
>items($login)
```

or a more agile syntax:

```
$jswin = new Ext_Window(
	$wincfg->
		layout('fit')->
		width(300)->
		height(150)->
		closable(false)->
		resizable(false)->
		plain(true)->
		items($login)
);
```

A this point, we have created a PHP variable called *jswin*. This is fine if we wish to manipulate it in our PHP script. If you need to pass a variable to JavaScript, so that it can be handled in the web page itself, you need to declare a JavaScript variable:

```
$win = new JsVariable('win', $jswin);
```

Now, we have created a JavaScript variable, called *win* and this variable's value is a reference to our **Ext.Window** object (**Ext_Window** in PHP).
We can use it to call our object's methods:

```
$win->show();
```

On Missing Methods

Let's imagine that you want to create a new panel:

```
$mypanel = new Ext_Panel(
	$cfg->
		width(340)->
		associate('region', 'west')
);
```

See the last call? It would be better if we could write:

```
$mypanel = new Ext_Panel(
	$cfg->
		width(340)->
		region('west')
);
```

So, what happened?
ExtJS' documentation is fairly good. However, now and then, a method falls between the cracks. In our example, *region(...)* is unfortunately not in the documentation and we were forced to called **associate(...)** to work around this omission. This is what makes **associate(...)** so important, however make sure to only use it when the real method is unavailable; using it systematically would defeat the purpose of ExtPHP's design.

When is the Code Generated?

1 When you instantiate an object, the corresponding JavaScript code is stored in its internal buffer.

2 When you call an object's method, again, code is added to its internal buffer.

3 For added flexibility, this code is not added to our main JavaScript buffer yet.

4 When you think that the code is ready to be added to your output, call the object's **jsrender()** method.

5 However, if your object has a **show()** method, you should call it instead.

You only need to call either of these methods if your object is not added to another object.
For instance, if you create a couple panels and add them to a **tabpanel** and add this **tabpanel** to a **viewport**, you will only need to call the viewport's **jsrender()** method.

Adding the Generated Code to your Page

JsWriter::get() will return the main buffer's content, therefore a good way to call your JavaScript code is:

```
new JsReady(JsWriter::get());
```

Declaring a Pure JavaScript Function

Sometimes, this is the simplest way to do things. Here is an example:

```
$button = new Ext_Button(
        $cfg->
                renderTo('button1-div')->
                text('Button 1')->
                handler(
                        new JsFunction(null, "alert('You clicked the button');")
                )
);
```

Here, we simply declared our JavaScript function directly. When you click this button, the **alert(...)** function will be called.

Arrays

Whenever possible, arrays will be automatically converted to their equivalent JSON.

Reference Manual

class JsWriter

This is a class whose methods are static. Think of it as the helper that writes the ExtJS code.

JsWriter::write($txt)

Add the content of *$txt* to its internal buffer.

JsWriter::get()

Return the content of the class' internal buffer, ready for display.

JsWriter::reset()

Empty its internal buffer.

JsWriter::JSON($phpstruct)

Take a php structure -likely an array- and convert it to JSON (JavaScript Obect Notation)

class JsReady

new JsReady($txt)

The JavaScript code passed as argument to this class' constructor will be executed when the web page is ready.

class JsFunction

new JsFunction($args, $body)

Create a new JavaScript class. The class body is contained in $body.
Example:

```
new JsFunction(null, "alert('Hello, World!');");
```

class JsLitteral

new JsLitteral($body)

Create an object that will be left untouched by **JWriter::JSON(...)**

class JsVariable

new JsVariable($var_name[, $value])

Create a JavaScript variable declaration. A JavaScript declaration is created and the variable is also maintained as a PHP variable.

Note:	$value is optional.

assign($value)

Assign a new value to the variable.

value()

Return the variable's value.

name()

Return the variable's name wrapped in a **JsLitteral** object.

any_other_function_name(any_set_of_arguments)

These calls will be delegated to JavaScript directly.

class ConfigTemplate

Parent class for all **xxx_Config** classes.

associate($var_name, $value)

Create a configuration key-value pair. This method is obviously inherited by all configuration objects, which is convenient when a class isn't fully wrapped in PHP. You should never need to use this class unless you are wrapping additional libraries.

class ClassTemplate

Parent class for all classes wrapped from ExtJS.
You should never need to use this class unless you are wrapping additional libraries.

For More Information

1 Browse the ExtJS documentation files in **docs/extjs**

2 Read the ExtJS tutorials at http://extjs.com

3 Play with the sample code found in **docs/samples/extphptest.php**

Acknowledgments

Cover page photograph by **Robert Parviainen**
http://www.flickr.com/photos/rtv/120547075/

The amazing ExtJS was created by the no less amazing **Jack Slocum**