ZFDatagrid

Version 0.8 - Manual

http://www.zfdatagrid.com

Author: Bento Vilas Boas

www.bentovilasboas.com

Revision: Dan Farrow

www.squarebracket.net

09/11/2011

Index

Introduction

ZFDatagrid is a presentation layer that allows you to display and manage data from a variety of sources into a variety of outputs.

It fully integrates with Zend Framework and makes heavy use of its components to run.

What's New

Crud Operations

Disable CSRF

Disable Columns for CRUD operations

Define input type

Decorators Improvements

Special tags for operations

Table

Save params in session

Never Show Order Images

Start at a specific record

Complex rows with rowspan/colspan

Core

Special Decorators ({{format}} {{callback}} {{helper}})

Field placeholder tags

Issues

More than 200 issue reports closed. Mostly related with:

   CRUD operations

   Filters

   Ajax

   PDF

Where to start?

You will need to download the latest version of ZFDatagrid from its project page at http://code.google.com/p/zfdatagrid/downloads/list. You will find there a few packages available. If this is your first contact with ZFDatagrid I advise you to download the sample project that contains all needed files and a variety of sample grids.

If you are already a user of ZFDatagrid you can download just the lib file and unpack it in your library folder. Don't forget to check the upgrade section for changes that may affect your existing code.

After you download the sample project, just unzip it into your webserver document root, run the provided SQL file and then edit */application/config.ini* with your site URL and credentials to access the database.

After doing that, go to your server URL, usually *http://localhost/grid* assuming grid is the folder that the sample project is in.

You should now see the same result as you can see here: http://zfdatagrid.com/grid

For OFC charts, I recommend that you download the OFC package from the ZFDatagrid project download page because the official one does not follow PEAR naming conventions.

My first grid

Now that your setup is done you can start to create your own grids. First thing you need to know is that ZFDatagrid provides a factory method to allow you to easily create a new instance.

```
$grid = Bvb_Grid::factory('Table',Zend_Config $config, 'id');
```

The first argument is the output format. ZFDatagrid will load the deployment class of the format you specify here.

For example, if you specify `Table`, ZFDatagrid will load the class `Bvb_Grid_Deploy_Table`. You could also specify `Pdf, Word, Json etc.`

The second argument is the optional grid configuration object. This allows you to easily change column attributes using configuration files rather than editing your php code. ZFDatagrid has a static method called `Bvb_Grid::setDefaultConfig($config)` which allows you to specify a default config for all subsequent grids. This is especially useful if your grids share many common parameters, e.g. deploy params, field options, etc. By using the default config along with a specific config for each grid you can keep your code clean and readable.

The third argument is the optional grid id. This allows you to have multiple grids per page without any conflicts. Every parameter set by the grid will be suffixed by this id, allowing ZFDatagrid to distinguish which parameter belongs to which grid.

Mod_rewrite

By default ZFDatagrid will build url like index.php?, you can enable mod_rewrite in ZFDdatagrid

```
Bvb_Grid::useModRewrite(true);
```

Data sources

Now that we have the grid instantiated we just need to define the data source.

The currently supported data sources are:

- `Bvb_Grid_Source_Zend_Table`
- `Bvb_Grid_Source_Zend_Select`

- `Bvb_Grid_Source_Doctrine`
- `Bvb_Grid_Source_Array`
- `Bvb_Grid_Source_Xml`
- `Bvb_Grid_Source_Json`
- `Bvb_Grid_Source_Csv`

The difference between the first and the second is simple, but very important. `Bvb_Grid_Source_Zend_Table` is intended to deal with models i.e. instances of `Zend_Db_Table`, while `Bvb_Grid_Source_Zend_Select` is intended to deal with a `Zend_Db_Select` class.

If you are working with models, don't attempt to pass the select object from the model to an instance of `Bvb_Grid_Source_Zend_Select`. Instead you should pass your model to an instance of `Bvb_Grid_Source_Zend_Table`, which will access the model's select object for you, and will also resolve and build your model's dependencies. You will find this especially useful for CRUD (Create, Read, Update and Delete) operations.

Here's an example of how to use a model as the source:

```
$model = new MyModel();
$source = new Bvb_Grid_Source_Zend_Table($model);
$grid->setSource($source);
```

Defining others sources is just as simple. Just instantiate the correct `Bvb_Grid_Source` class with your object as argument and pass the new instance to `$grid->setSource()`

Here is an example of how to use an instance of `Zend_Db_Select` as the source:

```
$grid = Bvb_Grid::factory('Table');
$select = $db->select()->from('some_table');
$grid->setSource(new Bvb_Grid_Source_Zend_Select($select));
```

You also need to define directory location for images:

```
$grid->imagesUrl(public/images/');
```

...call the deploy method and then pass the returned value to the view:

```
$myGrid = $grid->deploy();
$this->view->grid = $myGrid;
```

In the view you just need to `echo $this->grid` and you are done. Your first grid is complete. Now let's make some more magic.

Using sources

ZFDatagrid accepts a variety of data sources. After setting your source you can access it using the `getSource` method

```
$source = $grid->getSource()
```

This method only works with `Zend_Table,` `Zend_Select` and `Bvb_Grid_Source_Doctrine`. Any other source will return `true`.

Bvb_Grid_Source_Zend_Table

Use this source when you are using your own models.

```
$source = new Bvb_Grid_Source_Zend_Table(new Bugs());
$grid->setSource($source);
```

This will be very useful when performing CRUD operations. ZFDatagrid will resolve dependent tables and present correspondent fields names/values, and the CRUD operations will be executed using your model instance.

Source Specifics

You can use $_referenceMap in your model to help ZFDatagrid do a better job with multiple tables.

```
class Me_Model_Operations extends Zend_Db_Table{
    protected $_name = 'operations';
    protected $_primary = 'operation_id';
    protected $_referenceMap = array(

        'Model' => array(
            'columns' => 'model_id',
            'refTableClass' => 'Me_Model_Models',
            'refColumns' => 'model_id',
            'refBvbColumns' => 'label',
        ),

        'Refsystem' => array(
            'columns' => 'refsystem_id',
            'refTableClass' => 'Me_Model_Coordinatereferencesystem',
            'refColumns' => 'coord_ref_sys_code',
```

```
                'refBvbColumns' => 'coord_ref_sys_name',
            ),

        'Responsible' => array(
            'columns' => 'responsible_id',
            'refTableClass' => 'Me_Model_Users',
            'refColumns' => 'user_id',
            'refBvbColumns' => array('Alias Field' => 'real_name'),
        ),

        'Site' => array(
            'columns' => 'site_id',
            'refTableClass' => 'Me_Model_Sites',
            'refColumns' => 'site_id',
            'refBvbFollow' => true,
        ),
    );

}
```

refBvbColumns: will reference columns from the source that should be displayed

refBvbFollow: used to explore relations declared in mapped models in order to display other fields

Bvb_Grid_Source_Zend_Select

Use this when you have a `Zend_Db_Select` object.

```
$source = new Bvb_Grid_Source_Zend_Select($selectObject);
$grid->setSource($source);
```

Bvb_Grid_Source_Doctrine

Use this source when dealing with a Doctrine Query or Doctrine Record.

```
$source = new Bvb_Grid_Source_Doctrine($doctrineQuery);
$grid->setSource($source);
```

Bvb_Grid_Source_Array

This source accepts an array of arrays. The first argument is the array itself and the second, optional, is an array with fields names

```
$source_array = array(
            array('Alex', '12', 'M'),
            array('David', '1', 'M'),
            array('Emma', '2', 'F'),
            array('Jessica', '3', 'F'),
            array('Richard', '3', 'M'),
            array('Lucas', '3', 'M')
       );

$source = new Bvb_Grid_Source_Array($source_array, array('name', 'age', 'sex'));
$grid->setSource($source);
```

Bvb_Grid_Source_Xml

Use this source when you want to present an XML source. The first argument is the URL of the XML source and the second is the node to be presented. The third, optional, argument is the name of the column.

```
$source = new Bvb_Grid_Source_Xml('http://zfdatagrid.com/feed/', 'channel,item');
$grid->setSource($source);
```

Note: When using a URL as a source the cache option should be used for optimized performance

Bvb_Grid_Source_Json

For displaying a JSON source file or URL. The first argument is the location, the second the node to be presented and the third, optional, is the name of the column.

```
$source = new Bvb_Grid_Source_Json('media/files/json.json', 'rows');
$grid->setSource($source);
```

Bvb_Grid_Source_Csv

Use this to display information from a CSV file. The first argument is the file location, the second is the columns name to be presented and the third, optional, is the separator. By default the separator value is a comma ",".

```
$source = new Bvb_Grid_Source_Csv('media/files/grid.csv');
$grid->setSource($source);
```

Column options

I'm pretty sure you will want to make some changes to the columns that will be displayed. This is an essential feature of ZFDatagrid and provides some great options.

Columns options are managed using the `$grid->updateColumn` method.

When updating columns you must specify the output column name and not the field name. If you gave the field an alias you must use the alias when updating its column.

And how do you update a column? Like this:

```
$grid->updateColumn('field',array('options…'));
```

Here is a list of the available options for fields:

- callback
- class
- colspan
- decorator
- escape
- format
- hidden
- helper
- hRow
- newRow
- order
- orderField
- position
- remove
- rowspan
- search
- searchField
- searchType
- searchSqlExp
- searchTypeFixed
- style
- title
- translate

Field placeholder tags

Some options will receive optional arguments. If we want to pass the field value to a callback, decorator, or anything else we specify the field between two sets of curly brackets `{{field}}`. We call this a field placeholder, and it will be replaced by the actual value of the field for each record.

Field placeholders can be used in arguments to decorators, format plugins, callbacks, etc.

Note that after a field has been modified by a format plugin, the `{{field}}` placeholder will be replaced by the modified content, not the original field value.

You can access the original, unmodified field value using placeholders in the following format: `{{=field}}`

To enable access to unmodified field values in this way, you must set `enableUnmodifiedFieldPlaceholders` option to true in your grid config:

```
grid.enableUnmodifiedFieldPlaceholders = 1;
```

or use `setDeployOption()`:

```
$grid->setDeployOption('enableUnmodifiedFieldPlaceholders',1);
```

Special Decorators

You can use the result of a callback, helper and format option in your decorator.

Use one of those options `{{format}}` `{{callback}}` `{{helper}}`.

> *Note: These special tags will only be available when you make use of the corresponding action*

Crud Links

You can also make use of these tags `{{addUrl}}` `{{editUrl}}` `{{deleteUrl}}` to create the respective links to the operation.

> *Note: These tags will only work when the user has permission to perform the respective action*

Details of the available options

callback

You can use your own callback to format field content.

The params will be passed to your callback function. As usual you can use field identifiers to pass the actual field values.

```
$grid->updateColumn('ID', array(
        'callback'=> array(
                'function'=> array($this,'function'),
                'params'=> array('{{Name}}', '{{ID}}')
        )
));
```

class

This option will specify a CSS class to be applied to the table cell

```
$grid->updateColumn('field',array('class'=>'css_class'));
```

colspan

Number of columns the field spans in display.

Possible values

`''   (empty string)`	The full table width is used
`n < 0 (negative number)`	The table width less this value is used: if the table has 5 columns and you specified `'colspan' => -1`, it would be rendered in HTML with the attribute `colspan='4'`
`n (any other number)`	Passed straight to the markup: `colspan = 'n'`

decorator

Use this option to add you own text or html to the displayed value. As the name indicates, it decorates the field.

```
$grid->updateColumn('field',array('decorator'=>'This field value is: {{field}}'));
```

escape

Defines the callback to be applied to escape the field's value. The default escape callback is `htmlspecialchars`

Note: All column values are escaped using the default callback unless otherwise specified

Possible values

`true`	The displayed value will be escaped using the default callback `htmlspecialchars`
`false`	The displayed value will not be escaped
`'my_callback_function'`	The displayed value will be escaped using the specified callback function

```
//Won't escape the field
$grid->updateColumn('field',array('escape'=>false));

//Will escape the field based on default escape
$grid->updateColumn('field',array('escape'=>true));

//Will escape the field based on the provided callback
$grid->updateColumn('field',array('escape'=>'my_callback_function'));
```

format

Format is a simple way for you to format content (date, time, currency, boolean, etc). Please check the **Formatting content** section below for more information.

```
$grid->updateColumn('field',array('format'=>'number'));

$grid->updateColumn('field',array(
        'format'=>array(
                'number',array('param1','param2')
        )
));
```

helper

This option will call a View Helper.

```
$grid>updateColumn( 'field' ,array(
        'helper'=>array(
                'name'=>'formInput',
                'params'=>array(
                        'name','value'
                )
        )
));
```

Note: Please refer to the view helper you want to use for possible parameters

hidden

This option is similar to the `remove` option, but while the `remove` option is made by the `Bvb_Grid_Data` class, the class responsible for hiding the column is the deploy class. `JqGrid` uses this option.

All other classes will treat this as an alias to `remove`.

Possible values

`true`	The column will be hidden, but its value will be accessible with a {{field_placeholder}}
`false (default)`	The column will be displayed

```
$grid->updateColumn('field',array('hidden'=>true));
```

hRow

This is very useful for you to group results. Imagine you have a list of countries and want to group them by continent like here: http://zfdatagrid.com/grid/default/site/hrow

Possible values

`true`	This column will be used to group rows
`false (default)`	The column will not be used to group rows

Note: You can only use this option once per grid. The last defined value will prevail over the others.

```
$grid->updateColumn('field',array('hRow'=>true));
```

newRow

If != false, cell starts on a new row. May be an associative array with options for class and style:

```
$grid->updateColumn('field',array(
        'newRow' => array(
                'class' => $class_for_the_new_row,
                'style' => $style_for_the_new_row
        )
));
```

order

Use this option if you don't want to give the user the possibility to order the grid by this field. One reason may be the use of an SQL expression for outputting the field.

Possible Values

`true (default)`	The user will be able to order the grid by this field
`false`	The user will not be able to order the grid by this field

```
$grid->updateColumn('field',array('order' => true));
```

orderField

This is useful if you are using decorators or any other custom content. Instead of ordering by the field that corresponds to that column, the order will be made by the specified field.

```
$grid->updateColumn('field',array('orderField' => 'field2'));
```

position

Use this option to specify the position where a column will appear

Possible Values

`first`	The column will appear in the leftmost position
`last`	The column will appear in the rightmost position
`next`	The column will appear in the next available position

n   The column will appear in the nth position e.g. 1,2,3...

```
$grid->updateColumn('field',array('position' => 'last'));

$grid->updateColumn('field',array('position' => 'first'));

$grid->updateColumn('field',array('position' => '2'));

$grid->updateColumn('field',array('position' => '3'));
```

remove

This option is used to remove the field from the grid, but not from the query. For example, you may want to use the value of the id field in a decorator, but not to show the id in the grid.

Possible values

`true`	The column will be hidden, but its value will be accessible with a {{field_placeholder}}
`false (default)`	The column will be displayed

```
$grid->updateColumn('field',array('remove' => true));
```

rowSpan

Works just like rowspan in HTML.

Possible values

n   The HTML rowspan attribute will be set to the specified number *n* e.g. 1,2,3...

search

How the search will be performed, or if it will be performed.

To disable search on a particular field:

```
$grid->updateColumn('field',array('search' => false));
```

To use full-text search - simple usage:

```
$grid->updateColumn('field',array(
        'search' => array(
                'fulltext' => true
        )
));
```

Specify indexes (if no index is specified the field name will be used):

```
$grid->updateColumn('field',array(
        'search' => array(
                'fulltext' => true,
                'indexes' => 'field,field2'
        )
    ));
```

Specify in boolean mode (default):

```
$grid->updateColumn('field',array(
        'search' => array(
                'fulltext' => true,
                'extra' => 'boolean'
        )
    ));
```

Specify with query expansion:

```
$grid->updateColumn('field',array(
        'search' => array(
                'fulltext' => true,
                'extra' => 'queryExpansion'
        )
   ));
```

searchType

The search operand that will be used.

Possible values

sqlExp
like
llike
rlike
equal, =
>=
>
!=, <>
<=
<
r: (will apply a regexp)
NULL=> :isnull
NOT NULL => :isnotnull
EMPTY => :empty

```
$grid->updateColumn('field',array('searchType' => '='));
$grid->updateColumn('field',array('searchType' => '>='));
$grid->updateColumn('field',array('searchType' => 'llike'));
```

searchSqlExp

Use this to perform an sql expression.

```
$grid->updateColumn('Name',array(
        'searchType' => 'sqlExp',
        'searchSqlExp' => 'Name !={{value}}'
));
```

Note: Remember that the {{value}} tag will be replaced with the field value

searchTypeFixed

The user has the ability to alter the search operand by using one of the available options (see the searchType option above for a list of available operators)

If you don't want to give the user this option, set `searchTypeFixed` to false.

```
$grid->updateColumn('field',array('searchTypeFixed'=>false));
```

style

A CSS style to be used

```
$grid->updateColumn('field',array('style'=>'text-decoration:underline'));
```

title

Titles will be displayed on top of every grid or page if you are deploying it to PDF. If you don't define any title, ZFDatagrid will remove any non-alphanumeric characters from your field name and apply the `ucwords` function.

```
$grid->updateColumn('username',array('title'=>"User Title "));
```

translate

Whether to translate field's values. Useful for locale data.

```
$grid->updateColumn('country',array('translate'=>true));
```

Formatting content

One of the grid options that may be applied to a field is the format option. This will format field content.

This is especially useful for locale formatting, like currency, date/time, etc.

ZFDatagrid ships with 4 plugins to format content.

- Array
- Currency
- Date
- Image

Array

This is used when the field value is an array. It will give a tree like presentation to the results.

Currency

This will format user content to a currency value. The first parameter is the locale to apply to the content - `Zend_Locale::isLocale()` will be called to verify that it's a valid locale. If no option is provided, the formatter will look in the registry for a key with the name `Zend_Locale`

```
$grid->updateColumn('field', array(
        format => array(
                'currency',
                array('locale'=>'pt_PT')
        )
));
```

Date

Will format content to date/time. It accepts only one argument and can be an instance of `Zend_Locale`, a string that indicates the locale to apply, or an be an array with 3 arguments.

- `locale`
- `date_format`
- `format_type`

```
$grid->updateColumn('field', array(format=>array('date',array('locale'=>'pt_PT'))));
```

Image

This is a simple plugin that will create a img tag. The field content is the URL to be displayed. It accepts an array of arguments `key => value` that will be transformed into tag options

```
$grid->updateColumn('field', array(
        format => array(
                'number',
                array(
                        'border' => '0',
                        'class' => 'css_class'
                )
        )
));
```

Add custom plugins

Custom plugins must extend `Bvb_Grid_Formatter_FormatterInterface`:

```
interface Bvb_Grid_Formatter_FormatterInterface
{
        /**
         * Constructor.
         *
         * @param array $options
         */
        public function __construct ($options = array());

        /**
         * Formats a given value
         *
         * @param $value
```

```
        */
        public function format ($value);


}
```

You can add custom directories to load plugins from using:

```
$grid = Bvb_Grid::factory();
$grid->addFormatterDir('Bvb/Grid/Formatter','Bvb_Grid_Formatter');
$grid->addFormatterDir('My/Grid/Formatter','My_Grid_Formatter');
```

Customizing Deployment

Customizing Table

Ajax

Working with ajax and ZFDatagrid couldn't be simpler. Just specify that you wan to use ajax. ZFDatagrid will create a div with the id you specified as argument.

Please note that ajax and CRUD operations are not possible in the same grid. If you set ajax to be on in a grid with CRUD operations, ajax will be disabled.

```
$grid->setAjax('myId');
```

Filters

By default ZFDatagrid will try to provide the user the best way to filter results. For example, if you have a enum field in your database, rather than a text input, a select menu will be presented to filter the column.

When the user selects a value the results will be filtered to show only rows containing the selected value.

CSS Class

```
$filters = new Bvb_Grid_Filters();
$filters->addFilter('username',array('class'=>'my_class'));
$grid->addFilters($filters);
```

CSS Style

```
$filters = new Bvb_Grid_Filters();
$filters->addFilter('username',array('style'=>'margin:0px;'));
$grid->addFilters($filters);
```

Auto Filters

The examples below should be placed after the `$filters` object has been created, and before the call to `$grid->addFilters($filters)`, like this:

```
$filters = new Bvb_Grid_Filters();

// Paste example code here...

$grid->addFilters($filters);
```

Distinct

Once specified the system will select all distinct values for a given field and present them to users as a dropdown menu making it easy to filter the results for each possible value.

```
$filters->addFilter('username' => array(
        'distinct' => array(
                'field' => 'id',
                'name' => 'username'
        )
));
```

That will result in a SQL query like this:

```
SELECT id as key, name as value FROM <table> WHERE <previous conditions>
```

You can also define the field orientation

```
$filters->addFilter('username' => array(
        'distinct' => array(
                'field' => 'id',
                'name' => 'username',
                'order' => 'name ASC'
        )
));
```

Secondary Table

This will fetch all values from a secondary table and present them.

```
$filters->addFilter('FIELD', array(
        'table' => array(
                'table' => 'TABLE',
                'field' => 'SELECT_VALUE',
                'name' => 'SELECT_NAME',
                'order' => 'name asc'
        )
));
```

Manual filters (Array)

```
$filters->addFilter('username',array(
        'values' => array(
                'key' => 'value',
                'another_key' => 'another_value'
        )
));
```

Note: Order parameter accepts (field | name) (ASC | DESC).
The first option is not the field name but the option key.
The above code will order by username ASC.

Show Filters in export

This option will add the current filters at the end of the document (PDF Only)

```
$grid->setShowFiltersInExport(true);
```

Add your own

```
$grid->setShowFiltersInExport(array('My'=>'Filter'));
```

A merge with the filters values inserted by user will be made.

Transform

This option is intended to modify the value provided by the user. In most cases it will be useful for date/time transformation, or currency

```
function my_function ($value)
{
    return str_replace("/","-",$value);
}


$filters = new Bvb_Grid_Filters();
$filters->addFilter('date',array('transform'=>'my_function'));
$grid->addFilters($filters);
```

If a user searches for *2010/12/12* the value passed to your query will be `2010-12-12`

Render

The render option allows you to render a specific type of field. As an example, date or number range selection.

By default you can use date, number, select, text

```
$filters->addFilter('date'=>array('render'=>'date'));
```

Note: This will render a date range selection

Custom Filters

If you use JS libraries or if you have specific rendering requirements you can build a custom filter.

Custom filters must implement the `Bvb_Grid_Filters_Render_RenderInterface`, but for convenience you can extend the `Bvb_Grid_Filters_Render_RenderAbstract` class which already implements the interface and builds the common operations.

But you also need to define filters render location with the following code:

```
$grid->addFiltersRenderDir('My/Filters/Render/', 'My_Filters_Render');
```

One of the most common uses will be to use it with a datePicker implementation.

As an example you can use the following code (assuming you already loaded jquery)

```php
class My_Filters_Render_Date extends Bvb_Grid_Filters_Render_Table_Date{
        public function render()
        {
                return '<span>' . $this->__('From:') . "</span>" .
                        $this->getView()->datePicker(
                                $this->getFieldName() . '[from]',
                                "",
                                array(
                                        'dateFormat' => 'yy-mm-dd',
                                        'defaultDate' => $this->getDefaultValue('from')
                                ),
                                array_merge(
                                        $this->getAttributes(),
                                        array('id' => 'filter_' .
                                                $this->getFieldName() . '_from')
                                )
                        ) .
                        "<br><span>" . $this->__('To:') . "</span>" .
                        $this->getView()->datePicker(
                                $this->getFieldName() . '[to]',
                                "",
                                array(
                                        'dateFormat' => 'yy-mm-dd',
                                        'defaultDate' => $this->getDefaultValue('to')
                                ),
                                array_merge(
                                        $this->getAttributes(),
                                        array('id' => 'filter_' .
                                                $this->getFieldName() . '_to')
                                )
                        );
        }
}
```

And then add your filter

```php
$filters->addFilter('my_field', array('render' => 'date'));
```

Callback

Using a callback will greatly improve you ability to enhance your filtering system

```
$filters->addFilter('lastname',array(
        'callback' => array(
            'function' => array(
                    $this,
                    'customFilter'
            ),
            'params' => array(
                    'XXX',
                    'YYY'
            )
        )
));
```

Along with the params you defined, 3 more will be merged:

- `field =>`   The field name

- `value =>`   The field value

- `select =>`   The select Object (`Zend_Db_Select` or other)

      Note: It is YOUR responsibility to apply any condition to the select object

      Note 2: Any field not specified when adding filters will have search disabled.

External Filters

Using external filters give you the ability to change filters positions on a page, removing them from the grid.

You need to create your form elements and give them an id. Then define your custom function that will handle the filter.

```
function people ($id, $value, $select)
{
    $select->where('Population>?', $value);
}

function filterContinent ($id, $value, $select)
{
    $select->where('Continent=?', $value);
}

$grid->addExternalFilter('new_filter', 'people');
$grid->addExternalFilter('filter_country', 'filterContinent');
```

Your view should look something like this:

```
Only show countries with at least:
<?php    $continents = array('Europe');
         $array = array(
                 ''             => '--choose--',
                 10000000       => '10 Millions',
                 100000000      => '100 Millions',
                 500000000      => '500 Millions',
                 1000000000     => '1 Billion'
         );
         echo $this->formSelect(
                 'new_filter',
                 $this->grid->getParam('new_filter'),
                 array('onChange' => 'gridChangeFilters()'),
                 $array
         );
?>
people from
<?php
         echo $this->formCheckbox(
                         'filter_country',
                         $this->grid->getParam( 'filter_country' ),
                 array('onChange'=>'gridChangeFilters()'),
                         $continents
         );
?>
```

Mass Actions

Mass actions allow you to perform the same action on several records, by adding an extra column into the left of the grid and adding checkboxes into them.

This is an automated process, your primary keys will be added to field value and you will receive a post argument with the name `postMassIds` with commas separating the various values.

```
$massActions =
    array(
            array(
                    'url' => $grid->getUrl(),//For form action
                    'caption' => 'Remove (Nothing will happen)',//Select Captions
                    'confirm' => 'Are you sure?'
            ),//Confirmation message (Optional)
            array(
                    'url' => $grid->getUrl() . '/nothing/happens',
                    'caption' => 'Some other action',
                    'confirm' => 'Another confirmation message?'
                    )
            );


$grid->setMassActions($massActions);
```

Detailed View

If you have a lot of fields to show they probably will take more space than the available on screen. A good option is to set a detailed view page for the record.

> *Note: This is optional. If you don't set anything, all fields will be showed on the grid page.*

To define which fields will appear in the detailed view:

```
$grid->setDetailColumns(array());
$grid->setDetailColumns('name','address','age','country');
```

If you want to show all your fields on the detailed view, leave the method empty:

```
$grid->setDetailColumns();
```

Multiple Grids

It's very common the need to have multiple grids per page. It's also important that no conflicts exist between them. ZFDatagrid handles this by prefixing the grid id to each grid's URL parameters.

Here's the way to set a grid's id:

```
$grid = Bvb_Grid::factory('Table','myId');
```

You can also define the grid id using:

```
$grid->setGridId('myId');
```

...but remember that you must call this before calling:

```
$grid->deploy();
```

To get the current `gridId` use:

```
$grid->getGridId();
```

Default CSS classes

You can define alternating CSS classes that will be applied to the table's `td` elements:

```
$grid->setRowAltClasses('odd','even');
```

Extra Columns

You can add extra columns to the grid, placed on the right or left:

```
$right = new Bvb_Grid_Extra_Column();
$right    ->setPosition('right')
        ->setName('unique_name')
        ->setDecorator("<strong>{{Population}}</strong>");
$left = new Bvb_Grid_Extra_Column();
$left ->setPosition('left')
        ->setName('unique_name2')
        ->setTitle('Left')
        ->SetDecorator("<input type='checkbox' name='number[]'>");
$grid->addExtraColumns($right, $left);
```

Options available for extra columns

Name	Purpose	Expects
position	Define column position	left|right
name	The name for the column	unique_value
title	The title for the column	Any text
class	CSS class	A CSS class to apply
decorator	Add extra html to the column	HTML | text
style	CSS style	Style to be applied to the column
helper	Easy access to View Helpers	A registered View Helper
callback	Execute a callback function	A valid callback function
format	Format column value	A registered format
order	Column position	integer

Here's a full example:

```
$left = new Bvb_Grid_Extra_Column();

$left     ->position('left')
          ->name('Left')
          ->format('number')
          ->callback(array(
               'function' => 'my_function',
               'params' => array(
                    'my','params'
               )
          ))
          ->decorator("|{{field}}|")
          ->helper(array(
               'formText' => array(
                    'name',
                    'value',
                    array()
               )
          ))
          ->class('extraFields')
          ->title('MyTitle')
          ->style('text-decoration:underline;');
```

Extra Rows

You may need to add info in some part of the grid. This can be achieved with Extra Rows:

```
$rows = new Bvb_Grid_Extra_Rows();

$rows->addRow('beforeHeader',array(
        '', // empty field
        array('colspan' => 1, 'class' => 'myclass', 'content' => 'my content'),
        array('colspan' => 2, 'class' => 'myotherclass','content' => 'some '),
        array('colspan' => 1, 'class' => 'myclass', 'content' => 'flowers:) '),
    ));

$grid->addExtraRows($rows);
```

The first argument is the position in the grid. The available positions are:

- `beforeHeader`
- `afterHeader`
- `beforeTitles`
- `afterTitles`
- `beforeFilters`
- `afterFilters`
- `beforeSqlExpTable`
- `afterSqlExpTable`
- `beforePagination`
- `afterPagination`

Note: You don't need to worry about colspan from extra fields. It's auto calculated.
You only use the colspan option if the number of columns you want differs from the number in the grid

The following options are available for each field:

- `colspan =>`   The colspan to be applied. Please see the note above
- `class =>`   A CSS class to be applied
- `content =>`   The content to be displayed

Don't Show Order Images

You can omit order arrows, and make them appear only when a columns is sorted:

```
$grid->setAlwaysShowOrderArrows(false);
```

Never Show Order Images

Using this method no images will appear, even when a field is sorted:

```
$grid->setShowOrderImages(false);
```

Disable Order

```
$grid->setNoOrder(true);
```

Disable FIlters

```
$grid->setNoFilters(true);
```

Change records per page

You can let a user choose how many records are showed per page

```
$grid->setPaginationInterval(array(
        10 => 10,
        20 => 20,
        50 => 50,
        100 => 100
));
```

Start at a specific page

We can choose a value that the grid should start from with the `$grid->placePageAtRecord` method. For example:

```
$grid->placePageAtRecord('PRT','green');
```

The first argument is the field identifier or primary key value that ZFDatagrid should look for. The second argument, optional, is a CSS class to be applied to the row. In this example ZFDatagrid will automatically position the user at the page where the specified record is found.

Note: ZFDatagrid checks all rows until the requested id is found, so avoid using this with large data sets

Define row CSS class based on a condition

Lets imagine you want to add a CSS class attribute `class="green"` to every row in which population value is greater than 20000:

```
$grid->setClassRowCondition("{{Population}} > 20000","green");
```

The first argument is the condition. The field placeholder `{{Population}}` will be replaced by the value of that field.

When comparing strings please remember to enclose the field identifier with single quotation marks so PHP will treat it as a string rather than a number:

```
$grid->setClassRowCondition("'{{field_identifier}}' = 'example'","green");
```

You can add a third argument that is a CSS class name to be applied if the condition is not true:

```
$grid->setClassRowCondition("{{Population}} > 20000","green","orange");
```

Define cell CSS based on a condition

This is the same behavior as for a Row, but you need to specify the column that the condition will be applied:

```
$grid->addClassCellCondition('Population',"{{Population}} > 200000","red");
```

The first parameter is the column name, the second is the condition. As expected, you can also add a third argument in case the condition is not true:

```
$grid->addClassCellCondition('Population',"{{Population}} > 200000","red","green");
```

Render Specific Parts separately

You may need to render some parts of the grid separately on your page. You can achieve this by calling the `render()` method in your view:

```
$this->grid->render('pagination');

//Will display:
//<tr><td>_content_</td></tr>
```

The render method accepts a second argument which, if set to `true`, will surround the rendered part with the global start and end HTML:

```
$this->grid->render('pagination',true);

//Will display:
//<table><tr><td>_content_</td></tr></table>
```

The following parts are available to `render()`:

- `detail`
- `message`
- `form`
- `start`
- `extra`
- `header`
- `titles`
- `filters`
- `grid`
- `pagination`
- `end`

Changing table CSS classes

You can change the default CSS classes applied to the default table template in your config file. Check the corresponding section below for a list of all classes.

Apply filters when input is changed

By default a button to apply filters will be placed at the top of the table. And only when pressing that button or hitting 'enter' will apply the search. You can changes this behavior so filters are applied when user changes filter content (`onchange` event)

```
$grid->setUseKeyEventsOnFilters(true);
```

Save parameters in session

This option will save current grid parameters in session, so when user returns to the grid, filters, a sorting is preserved.

```
$grid->saveParamsInSession(true);
```

Customizing JqGrid

Setting locale

If you like to set default locale for your project call following for example in bootstrap:

```
Bvb_Grid_Deploy_JqGrid::$defaultJqgI18n = "sk";
```

To set locale for grid instance call:

```
$grid->jqgI18n = "sk";
```

Publishing results

Results from `Bvb_Grid_Deploy_JqGrid` object are sent to the web browser in situations to display the grid and to provide data to grid.

You need to call `deploy()` to render the grid on your HTML page. This function returns the HTML part of the grid which then needs to be added to your controller response. This function also adds javascript code to ZendX jQuery view helper and that also needs to be added to the controller response.

You could do the following in your view script:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
…
  <!-- next line will add all necessery JavaScript and CSS includes →
  <?php echo $this->jQuery();?>
…
</head>
<body>
…
<!-- next line will output HTML part of grid →
<?=$this->grid->deploy();?>
...
</body></html>
```

If you are using `Zend_Layout` put the jQuery part into your layout script.

The other situation when results of `Bvb_Grid_Deploy_JqGrid` are sent to theweb browser is when the front-end requests data through ajax. These requests are recognized in the `ajax()` function which will send the requested data and stop `Zend Application`. The `ajax()` function should be called before `deploy()` function.

Chaining jqGrid methods

You may use `Bvb_Grid_Deploy_JqGrid::cmd()` method to call jqGrid methods. The methods are generated to Javascript as chain of commands in jqGrid's new API format.

`Bvb_Grid_Deploy_JqGrid::cmd()` also fixes some inconsistency in jqGrid's API:

- Allows jqGrid post data module methods to be chained (`setPostData()`, `appendPostData()`, `setPostDataItem()`, `removePostDataItem()`)
- It is not possible to call `.trigger()` in the new API style

Example of how to use it in view script:

```php
<?php $this->jQuery()->onLoadCaptureStart(); ?>

$('#filterShowAllCheckbox').bind('change', function() {
        <?=$this->grid
        ->cmd('setPostDataItem', 'includeAll', new
Zend_Json_Expr('$(this).is(":checked")'))
    ->cmd('trigger', 'reloadGrid');
?>
});

<?php $this->jQuery()->onLoadCaptureEnd(); ?>
<?php echo $this->grid->deploy(); ?>
```

The value from the jQuery command `$(this).is(":checked")` will be set in `$_POST['includeAll']`

Take care that the value is a String and not a Boolean.

Customizing Csv

When deploying to a CSV file, you may want to change some pre-defined options, especially when dealing with large amounts of data.

```php
$grid->setDeployOption('set_time_limit',200);
$grid->setDeployOption('memory_limit','50M');
$grid->setDeployOption('skipHeaders',true);
$grid->setDeployOption('store',true);
```

Customizing OFC

You can add a set of values using:

```
$grid->addValues('GNP', array(
        'set_colour' => '#FF0000',
        'set_key' => 'Gross National Product',
        'chartType' => 'Bar_Glass'
));
```

You can define axis labels using:

```
$grid->setXLabels('Name');
```

And define chart type:

```
$grid->setChartType($type);
```

Any other option you may need for a specific set of results, just pass them as an option when using the method `addValues()`. ZFDatagrid will proxy that method into Open Flash Chart.

Note: Please download the OFC library from the ZFDatagrid project page.
The official release does not comply with PEAR naming conventions and will not work with ZFDatagrid.

Customizing Others

No customization is available for others

Adding your own deploy classes

You can create your own deploy class and use as any other shipped class. Add your class dir to the ZFDatagrid by doing:

```
Bvb_Grid::addDeployPrefixPath($dir, $prefix)
```

Other Useful Methods

Columns as objects

Instead of using:

```
$grid->updateColumn();
```

you can use objects to represent every column:

```
$column = new Bvb_Grid_Column('FIELD_NAME');
$column   ->title('Field Title')
          ->class('css')
          ->decorator('{{Country}} <em>({{Capital}})</em>');
```

Note: Use the options you want to set as method name, hidden, style, title, callback, etc.

To update the column in the grid use:

```
$grid->updateColumns($column);
```

Define which export methods are available

Set the available export formats for your grid by using the `setExport()` method:

```
$grid->setExport('print','pdf','word','…etc…');
```

By default these methods are available:

```
array('pdf', 'word', 'wordx', 'excel', 'print', 'xml', 'csv', 'ods', 'odt', 'json');
```

Define char encoding

```
$grid->setCharEncoding('utf8');
```

Set default escape function

This callback will be applied to source content. Default value is `htmlspecialchars`

Please refer to columns options->escape for more info

```
$grid->setDefaultEscapeFunction($callback);
```

setDefaultFiltersValues

This method will pre-fill and filter source data.

```
$grid->setDefaultFiltersValues(array('field'=>'value'));
```

Note: This option will only pre-filter results. The user has the ability to remove filters

setRecordsPerPage

This method will define the number of records to show per page. 0 (zero) will show all records

```
$grid->setRecordsPerPage(30);
```

Note: The default value is 15

setShowFiltersInExport

This method will show user's filters when exporting results.

```
$grid->setShowFiltersInExport(true);
```

You can add extra 'filters' by passing an array to the method

```
$grid->setShowFiltersInExport(array('User'=>'Admin'));
```

isExport

This method will return true if user is exporting content

```
$grid-> isExport();
```

setView

Defines the view to be used

```
$grid->setView($view);
```

Columns to show

You can define the columns you want to show for the various outputs:

```
$grid->setTableGridColumns('field1','field2','field5');
$grid->setPdfGridColumns('field1','field3','field4');
```

This option is available to every deploy class. Just use the following method format:

```
$grid->set_DeployClass_GridColumns();
```

> *Note: When setting columns to show you must also specify any extra columns you defined.*

Please refer to http://zfdatagrid.com for the phpDoc generated documentation and extra methods.

Events

In order to improve ZFDatagrid flexibility, events where added.

You can listen to an event using the following code:

```
$grid->listenEvent('crud.before_update', $callback);
```

The code above will be triggered before a CRUD operation takes place. And then is your callback you can access the event data by using:

```
function My_Callback(Bvb_Grid_Event $event)
{
    $params = $event->getParams();
    $param = $event->getParam('my_param');
    $name = $event->getName();
    $object = $event->getObject();
    $event->setParam($param);
}
```

Sending extra params

You can pass a third optional param when listening events. Those params will be merged with ZFDatagrid ones.

```
$grid->listenEvent('crud.before_update', $callback, array('key'=>'value'));
//In your event
$event->getParam('key');
```

Event Prefix

Setting an event prefix will prefix a custom string to the beginning of every event name.

```
Added Events prefix.
$grid->setEventsPrefix('users');//Or something else
```

```
config file
grid.eventsPrefix = users
```

Priority

You can specify an event priority by passing a forth parameter

```
$grid->listenEvent($event, $callback, array $params = array(), $priority = 10);
```

List of events

Name	Param(s)	Description
grid.set_source	Source *(Bvb_Grid_Source_Interface)*	When setting the grid data source
grid.all_fields_added	fields*(array)*	When all fields are added to the grid
grid. update_column	field*(string)*, options*(array)*	When updating a field option
source.add_condition	filter*(string)*, op*(string)*, field*(string)*	When adding a condition to a data source. Filter: the value to filter; op: the operand to be applied; field: the field where values will be applied
grid.init_deploy	N/A	When the method deploy is called
grid.before_filters	N/A	Before filters get applied to a data source
grid.add_extra_filters'	filters*(array)*	When extra filters are added
grid.add_extra_columns	columns*(array)*	When extra columns are added
grid.add_extra_rows	rows*(array)*	When extra rows are added
grid.set_mass_actions	source*(Bvb_Grid_Source_Interface)*	When mass actions are added to the system
crud.set_values	form*(Zend_Form)*	When form values are defined
crud.before_insert	table*(string)*, connectionId*(string)*, values*(array)*	Before adding a new record to the source
crud.after_insert	table*(string)*, connectionId*(string)*, values*(array)*, insertId*(string)*	After adding a new record to the source
crud.before_update	table*(string)*, connectionId*(string)*, newValues*(array)*, oldValues*(array)*, condition*(array)*	Before updating a new record. oldValues are the record values fetched immediately before the update takes place. newValues are the values to be updated. An array containing conditions to apply to the update query
crud.after_update	table*(string)*, connectionId*(string)*, newValues*(array)*, oldValues*(array)*,	After updating a record. newValues: These values are fetched immediately after the record is updated.

	condition[(array)]	
crud.before_delete	table[(string)], connectionId[(string)], values[(array)], condition[(array)]	Triggered before deleting a record
crud.after_delete	table[(string)], connectionId[(string)], values[(array)], condition[(array)]	Triggered after deleting a record
crud.set_form	form[(Zend_Form)]	When adding the form instance to the grid.
crud.elements_loaded	elements[(array)]	When form elements are added to the Zend_Form instance.
crud.form_built	form[(Zend_Form)]	when form elements are built in Zend_Form instance

An instance o Bvb_Grid is always trough the event. To access it do:

```
$event->getSubject();
```

Crud Operations

It's important to note that Form elements are only built after passing your form instance to the grid with

```
$grid->setForm(Bvb_Grid_Form $form)
```

Only after this you can manage your form elements with:

```
$grid->getForm(1)->getElement('element')->something()
```

How does it work?

By default the form is built based on the table definition.

All data validation is added automatically and forms fields are also built based on fields definition.

`Bvb_Grid_Form` is just a proxy for `Zend_Form` or any other class that extends `Zend_Form`.

You can define you form like this:

```
$form = new Bvb_Grid_Form($class='Zend_Form', $options  =array());
//If you use your custom form class, change Zend_Form for your class.
// The second argument are the options to be passed to the class constructor().
```

You can define which operations are available for users:

```
$form->setAdd(true|false);
$form->setEdit(true|false);
$form->setDelete(true|false);
```

You can optional add a add button to the top of form by doing this:

```
$form->setAddButton(true|false);
```

You can also add a 'Save and Add New' button that will redirect user to the add form after a successful insert.

```
$form->setSaveAndAddButton(true);
```

One interesting thing is that you may use Zend_Form just to validate your data and then do something else with it before saving to the database. For example:

```
$form->setIsPerformCrudAllowed(false);

//Won't insert any data to the source when performing any operation

$form->setIsPerformCrudAllowedForAddition(true|false); //Won't insert any data when
adding new data

$form->setIsPerformCrudAllowedForEdition(true|false); //Won't update any data when
editing

$form->setIsPerformCrudAllowedForDeletion(true|false); //Won't delete any data when
deleting a record
```

Data recording is inside a `try{...} catch{..}` block, so you can throw an exception within your callbacks (see below) to make the validation fail.

What if you don't want a specific field?

You can include or exclude fields from the form by setting this:

```
$form->setAllowedFields(array(
        'times','next'
));//Only these fields will appear in the form

$form->setDisallowedFields(array(
        'time','next'
));//These fields won't appear in the form

$form->setFieldsBasedOnQuery(true|false);
//Only the fields from the query will be in the form
```

Finally we need to add the form to your grid:

```
$grid->setForm($form);
```

After adding the form you can get it by doing:

```
$grid->getForm(1);
```

...and do whatever you want with it:

```
$grid->getForm(1)->getElement('my_field')->setValue('value');
```

Why the number 1 when getting the form instance?

Because ZFDatagrid now supports bulk actions, every record is now a subForm.

You can achieve the same result doing

```
$grid->getForm()->getSubForm(1);
```

Forms are numbered starting from one.

Bulk Actions

Setting up bulk actions is incredibly easy. All you have to do is:

```
$form = new Bvb_Grid_Form();

$form     ->setBulkAdd(2) //Replace 2 with the number of forms you want to display
          ->setBulkDelete(true)
          ->setBulkEdit(true)
          ->setAdd(true)
          ->setEdit(true)
          ->setDelete(true)
          ->setAddButton(true);

$grid->setForm($form);
```

You can change the display layout of form elements from a vertical position to horizontal.

```
$grid->setUseVerticalInputs(false);
```

Different Table

If you want to manually override the table where values will be inserted, use:

```
$form->setTable(string $table);
```

Special Features

There are a few special methods that may be very, very useful to a lot o us.

Imagine that you have an ACL based system and that a record with the id 14 can only be removed by an admin.

But the id to be removed is contained in the URL - so what stops me renaming it? Nothing.

We can't prevent this behavior but we can make it useless by forcing some values to be added to the condition:

```
$form->setOn(Delete|Edit)AddCondition(array('user'=>'12'));
//This will execute the query:
//DELETE FROM table where id = '14' AND user='12';
```

Other thing we may need is to force some fields to be filled by values that cannot be set by the user. The most common example is the user id. We can do achieve that by doing:

```
$form->setOn(Add|Edit)Force(array('user_id'=>1));
```

The above code will add another field to the data being submitted.

Built-in Elements

If you specify a Zend_Form instance when creating a new Bvb_Grid_Form it may already contain elements, and you may only wan to use them and do not want for ZFDatagrid to create new ones. For that use:

```
$form->(set|get)UsePreDefinedFormElements(true);
```

Disable CSRF

This method won't create a hash element to valid form input. Please use this method wisely.

```
$form->setUseCSRF(false);
```

Show delete confirmation page

Instead a JavaScript confirmation window, you can show a full page with record details for delete confirmation.

```
$form->setDeleteConfirmationPage(true);
```

Columns Position

To change columns position:

```
$grid->setCrudColumnsPosition('left');//default
$grid->setCrudColumnsPosition('right');
```

Disable Columns for CRUD operations

You may want to remove those extra columns auto created when defining crud operations.

Use to remove the columns and button:

```
$form->setEditColumn(false);
$form->setDeleteColumn(false);
$form->setaddButton(false);
```

...and then you can use this tags in your decorators:

```
{{addUrl}} {{editUrl}} {{deleteUrl}}
```

Define input type

You can change the input type created by default using the `$form->setInputsType()` method:

```
$form->setInputsType('bug_description'=>'textarea','user_password'=>'passowrd');
```

Default Decorators

To speed things up default decorators are used and use the `table > tr > td` structure:

```
groupDecorator = array('FormElements', array('HtmlTag', array('tag' => 'td', 'colspan' =>
'2', 'class' => 'buttons')), array(array('row' => 'HtmlTag'), array('tag' => 'tr')));

subformGroupDecorator = array('FormElements', array('HtmlTag', array('tag' => 'td',
'colspan' => '90', 'class' => 'buttons')), array(array('row' => 'HtmlTag'), array('tag'
=> 'tr')));

elementDecorator = array('ViewHelper', 'Description', 'Errors', array(array('data' =>
'HtmlTag'), array('tag' => 'td', 'class' => 'element')), array(array('label' => 'Label'),
array('tag' => 'td')), array(array('row' => 'HtmlTag'), array('tag' => 'tr')));

subformElementDecorator = array('ViewHelper', 'Description', 'Errors', array(array('data'
=> 'HtmlTag'), array('tag' => 'td', 'class' => 'element')), array(array('label' =>
'Label'), array('tag' => 'td','class'=>'elementTitle')), array(array('row' => 'HtmlTag'),
array('tag' => 'tr')));

subformElementTitle = array(array('Label', array('tag' => 'th')));

subformElementDecoratorVertical = array('ViewHelper', 'Description', 'Errors',
array(array('data' => 'HtmlTag'), array('tag' => 'td', 'class' => 'element')));

fileDecorator = array('File', 'Description', 'Errors', array(array('data' => 'HtmlTag'),
array('tag' => 'td', 'class' => 'element')), array(array('label' => 'Label'), array('tag'
=> 'td')), array(array('row' => 'HtmlTag'), array('tag' => 'tr')));

buttonHiddenDecorator = array('ViewHelper');

formDecorator = array('FormElements', array('HtmlTag', array('tag' => 'table', 'style' =>
'width:99%')), 'Form');

subFormDecorator = array('FormElements', array('HtmlTag',array('tag' => 'table', 'style'
=> 'margin-bottom:5px; width:100%', 'class' => 'borders')));

subFormDecoratorVertical = array('FormElements', array('HtmlTag', array('tag' => 'tr')));
```

Change Decorators

If you want to change your decorators use the following methods. NOTE: You must change the decorators before adding the form to the grid,

```
$form->(set|get)groupDecorator();
$form->(set|get)elementDecorator();
$form->(set|get)buttonHiddenDecorator();
$form->(set|get)formDecorator();
```

Don't use decorators

If you don't want ZFDatagrid to setup any element decorator use this:

```
$form->setUseDecorators(false);
```

Templates

The ability to customize table designs is a very common feature request. Fortunately, ZFDatagrid has support for templates.

Which deployment classes use templates?

At this moment templates are supported by

- Tables
- PDF
- Print
- Word
- Word 2007
- Open Document Text
- Open Document Spreadsheet

How do they work?

First Step

Your template must extend the related `Bvb_Grid_Template_*` base template.

If you are creating a new template for a table you must declare your class like this:

```
class My_Template_Table extends Bvb_Grid_Template_Table{}
```

Second Step

Register your template in the grid.

```
$grid->addTemplateDir('My/Template','My_Template','table')
```

1. The dir location

2. The class name

3. Template type

The available templates types are:

- `table`
- `print`
- `pdf`
- `ods`
- `odt`
- `word`
- `wordx`

Third Step

Define the template: This can be as simple as:

```
$grid->setTemplate($templateName,$TemplateType(table, print, pdf));
```

Passing parameters to the template

Templates are classes without inheritance. User defined information can be passed to the template via the following calls:

```
$grid->addTemplateParam('name','value');
$grid->addTemplateParams(array);
$grid->setTemplateParams(array);
```

Attention: Using the set method, all previous params will be removed.

Getting Params from the Template

After setting the params you can access them within your template in the options property.

```
$myParams = $this->options['userDefined'];
```

Translation

Internationalization is almost a must have feature. ZFDatagrid makes this incredible easy.

You just need to register your `Zend_Translator` instance in the Registry like this:

```
Zend_Registry::set('Zend_Translate',Zend_Translate_Instance $translate);
```

Or set the current `Zend_Translator` instance to the grid

```
$grid->setTranslator(Zend_Translator $translator);
```

What is translated?

- Pagination
- Fields Titles
- Strings like: remove order, remove filters.
- PDF supporting text: (title, page, subtitle, etc,)
- Results form CRUD operations. Please note that validation messages are from `Zend_Form`

Cache

When performing intensive queries or using sources as files, is a good idea to use cache for performance improvement.

How?

It's really easy.

```
$grid->setCache(array('use'=>array('db'=>'true|false'), 'instance'=>Zend_Cache $cache
,'tag'=> 'my_tag' ));
```

And affects

When using cache all of your queries will be put on cache and they will use the tag provided.

After performing CRUD operations all the cache based on the tag will be erased.

SQL aggregate expressions

If you are displaying data like annual results, or other similar, you may need to add some functions to calculate the grand total, average, or something else.

Defining SQL expressions will add an extra row at the bottom of the table with the expressions you defined:

Examples of adding SQL expressions

```
$this->setSqlExp(array(
        'credit'=>array(
                'functions'=>array('COUNT'),
                'value'=>'credit'
        )
));


//If no value option is defined the field name will be used.
```

This will count all records from the credit field

```
$this->setSqlExp( array('credit'=>array('functions'=>array('AVG'))));
```

This will make the average from the credit field

More Options

You can also apply a format to the field and add a CSS class

```
$grid>setSqlExp( array(
'credit'=>array('functions'=>array('AVG'),'value'=>'credit','format'=>'currency','class'=
>'annual_results')));

//If this field is already using a format from the grid the same format will be applied
here.

// If you don't want that please define the format option as an empty string

$grid->setSqlExp( array('credit'=> array('functions'=>array('AVG'), 'value'=>'credit',
'format'=>'', 'class'=>'annual_results')));
```

Other sources

You can still use SQL expressions even if you are using other source then a DB. At this moment six are supported: `min, max, avg, sum, product, count`

Zend_Config

Please note that you can also use:

```
$grid->setDeployOptions(array()));
```

to change a config value:

```
$grid->setDeployOptions(array('title'=>'My Custom Title','subtitle'=>date('Y-m-d')));
```

...or:

```
$grid->setDeployOption ('title'=>'My Custom Title');
```

You can use a config file to setup the current grid.

```
$grid = Bvb_Grid::factory('Table',Zend_Config $config);
```

You can also pass an array instead of a Zend_Config instance.

ZFDatagrid has a static method that allows you to define the default configuration to be applied to all grids.

```
Bvb_Grid::setDefaultConfig(Zend_Config $config);
```

With this method you can have a base configuration and then use a config file per grid for more organization.

Below is a list of supported options in a config file or array.

Fields

```
fields.FIELD_NAME.option = Value
```

Full list

```
fields.Name.title = "Field Title" ; Title to display
fields.Name.remove = false ; Defaults 0
fields.Name.style = "" ; Defaults ""
fields.ID.escape = 1
fields.ID.translate = 1
fields.Name.hidden = false ; Defaults 0
fields.Name.decorator = "{{Name}}"
fields.Name.hRow = false ; Defaults 0
fields.Name.position = 2 ; Field presentation position
fields.Name.order = 1 ; If a field can be ordered. Defaults 1
fields.Name.orderField = "ID" ; Defaults null
fields.Name.class = "width_150" ; Defaults null
fields.Name.searchType = "="
fields.Name.searchField = "" ; Field where the search will be performed
fields.Name.searchTypeFixed = (0|1) ; Defaults 0
fields.Name.search.fulltext = true
fields.Name.search.indexes = Name
fields.Name.search.extra = queryExpansion
fields.Name.format.function = date
fields.Name.format.params.locale = "fr_FR"
fields.Name.callback.function = "my_function"
fields.Name.callback.class = $this
fields.Name.callback.params.something = {{Name}}
```

Table template classes

```
template.table.cssClass.table = borders
template.table.cssClass.topRow = querySupport
template.table.cssClass.massActions = massActions
template.table.cssClass.massSelect = massSelect
template.table.cssClass.sqlExp = sum
template.table.cssClass.tableFooter = barra_tabela
template.table.cssClass.tableFooterExport = paginationExport
template.table.cssClass.tableFooterPagination = paginationNumbers
template.table.cssClass.detailRight = detailRight
template.table.cssClass.detailLeft = detailLeft
```

```
template.table.cssClass.noRecords  = noRecords
template.table.cssClass.filters = subtitulo
template.table.cssClass.hBar = hbar
template.table.cssClass.formMessageOk = alerta
template.table.cssClass.formMessageError = alerta_red
template.table.cssClass.gridLoading = gridLoading
```

Records per page

```
grid.recordsPerPage = 30
```

Deploy Formats

CSV

```
deploy.csv.dir = "media/temp" ; Dir to save documents
deploy.csv.save = 1
deploy.csv.display = 1
deploy.csv.download = 1
deploy.csv.memory_limit = "128M"
deploy.csv.set_time_limit = 3600
```

Excel

```
deploy.excel.dir = "media/temp" ; Dir to save documents
deploy.excel.name = "barcelos" ; Document name
deploy.excel.save = 1
deploy.excel.download = 1
```

JSON

JSON output format takes no arguments.

ODS (Open Document Format (Text))

```
deploy.ods.dir = "media/temp" ; Dir to save documents
deploy.ods.name = "my_name" ; Document name
deploy.ods.save = 1
deploy.ods.download = 1
```

ODT (Open Document Format (spreadsheet))

```
deploy.odt.dir = "media/temp" ; Dir to save documents
deploy.odt.name = "my_name" ; Document name
deploy.odt.save = 1
deploy.odt.download = 1
deploy.odt.title = "Document title"
deploy.odt.subtitle = "Document subtitle"
deploy.odt.logo = "PATH_TO_LOGO"
deploy.odt.footer = "Footer message"
```

OFC (Open Flash Chart)

```
deploy.ofc.files.json = "/grid/public/scripts/json/json2.js"
deploy.ofc.files.js = "/grid/public/scripts/swfobject.js"
deploy.ofc.files.flash = "/grid/public/flash/open-flash-chart.swf"
deploy.ofc.options.set_bg_colour = '#FFFFFF'
deploy.ofc.title = "Chart Title"
deploy.ofc.dimensions.x = 900
deploy.ofc.dimensions.y = 400
deploy.ofc.type = "Bar_Glass"
```

PDF

```
deploy.pdf.title = "Page Title"
deploy.pdf.dir = "media/temp" ; Dir to save documents
deploy.pdf.save = 1
deploy.pdf.download = 1
deploy.pdf.logo = "public/images/logo.png"
deploy.pdf.title = "DataGrid Zend Framework"
deploy.pdf.subtitle = "Easy and powerful - (Demo document)"
deploy.pdf.footer = "Downloaded from: http:; www.petala-azul.com"
deploy.pdf.size = "a4" ; (A4|LETTER) Default is A4
deploy.pdf.orientation = "LANDSCAPE" ; (LANDSCAPE|PORTRAIT) Default is Portrait
deploy.pdf.page =" Page N." ; Text before page numbers
```

PDF COLORS

```
deploy.pdf.colors.title = #000000
deploy.pdf.colors.subtitle = #111111
```

```
deploy.pdf.colors.footer = #111111
deploy.pdf.colors.header = #AAAAAA
deploy.pdf.colors.row1 = #EEEEEE
deploy.pdf.colors.row2 = #FFFFFF
deploy.pdf.colors.sqlexp = #BBBBBB
deploy.pdf.colors.lines = #111111
deploy.pdf.colors.hrow = #E4E4F6
deploy.pdf.colors.text = #000000
deploy.pdf.colors.filters = #F9EDD2
```

Print

```
deploy.print.dir = "media/temp" ; Dir to save documents
deploy.print.name = "my_name" ; Document Name
deploy.print.save = 1
deploy.print.download = 1
deploy.print.title = "Document title"
```

Table

```
deploy.table.imagesUrl = "/grid/public/images/" ; The URL for images
deploy.table.template = "outside" ; The template to be used
deploy.table.templateDir = "My_Template_Table" ; Other dir with templates
```

Word

```
deploy.word.dir = "media/temp" ; Dir to save documents
deploy.word.name = "my_name" ; Document Name
deploy.word.save = 1 ; Save the file on disk
deploy.word.download = 1 ; Download file
```

Word (.docx)

```
deploy.wordx.dir = "media/temp" ; Dir to save documents
deploy.wordx.name = "my_name" ; Document Name
deploy.wordx.save = 1
deploy.wordx.download = 1
deploy.wordx.title = "Document title" ; Title for document
deploy.wordx.subtitle = "Document subtitle" ; Subtitle for document
deploy.wordx.logo = "path/to/logo" ; PATH to logo
deploy.wordx.footer = "Footer message" ; Footer message
```

XML

```
deploy.xml.dir = "media/temp" ; Dir to save documents
deploy.xml.save = 1
deploy.xml.display = 1
deploy.xml.download = 1
deploy.xml.name = "The_file_name.xml"
```

Extra Rows

```
extra.row.one.position = beforePagination
extra.row.one.colspan=5
extra.row.one.content = My Content
extra.row.one.class = css_class;
extra.row.two.position = afterPagination
extra.row.two.colspan=5
extra.row.two.content = This is just an example
extra.row.two.class = my_class
```

Upgrading

0.7 to 0.8

Cache

Cache 'use' param, has been renamed to 'enable'

New:

```
$grid->setCache(array('enable' => true|false, 'instance' => $cache, 'tag' => 'grid'));
```

Old:

```
$grid->setCache(array('use' => array('form'=>false,'db'=>false), 'instance' => $cache,
'tag' => 'grid'));
```

Callbacks/Events

Callbacks where removed in favor of events. A list of available events will be available soon. But for crud operations something like this can be done:

```
$grid->listenEvent('crud.before_update', $callback);
```

Mass actions

They have their own class now. Changes:

OLD:

```
$actions->setMassActions(array(array('url' => $grid->getUrl(),
                                     'caption' => 'Remove (Nothing will happen)',
                                     'confirm' => 'Are you sure?'),
                               array('url' => $grid->getUrl() . '/nothing/happens',
                                     'caption' => 'Some other action',
                                     'confirm' => 'Another confirmation message?')));
```

NEW:

```
$actions = new Bvb_Grid_Mass_Actions();
$actions->setMassActions(array(array('url' => $grid->getUrl(),
                                     'caption' => 'Remove (Nothing will happen)',
                                     'confirm' => 'Are you sure?'),
                               array('url' => $grid->getUrl() . '/nothing/happens',
                                     'caption' => 'Some other action',
                                     'confirm' => 'Another confirmation message?')));


$grid->setMassActions($actions);
```

Refactoring:

```
$grid->removeParam(s) => $grid->clearParam(s)
```

0.6.5 to 0.7

Setting grid columns

Until now when setting grid columns, extra columns where always there. From now on, when defining grid columns you must also specify any extra column you want to see. Use the column name.

0.6 to 0.6.5

* Abstract classes and interfaces where prefixed with the file name.
`Bvb_Grid_Formatter_Interface => Bvb_Grid_Formatter_FormatterInterface`

* Callbacks in forms are now called using `call_user_func_array` instead `call_user_func`

* Forms are now built with subForms. So you will see `1-` prefixed in all of your input ids. Input names also changed to arrays. This does not affect any previous PHP code, just if you are using JS libraries to beautify your forms.

0.5 to 0.6

Factory Pattern

Now you don't have to instantiate the deploy class. Use this:

```
$grid = Bvb_Grid::factory('table',$options = array(),$gridId = '');
```

Uniformed fields calls

Whenever you are, you must call the field name by its output and not by name. Table prefixes are also over.

If your raw query looks like this:

```
SELECT id, username as name, age, status, online FROM users where id='1';
```

you must reference to the field username as `name`. If you are using joins and you have two fields with the same name on different tables you must give one of them an alias, otherwise the first will be overwritten by the last.

No direct Access

You cannot set class attributes directly. You now must use the `set()` method.

Instead of:

```
$grid->cache = array();
```

you now have to do:

```
$grid->setCache(array());
```

CRUD Operations

CRUD operations now use `Zend_Form` to render the form. No more custom forms. Please refer to the CRUD operations section for more information

Data Sources

ZFDatagrid now uses a interface to build the grid, so no more calls like this:

```
$grid->setDataFromArray();
$grid->setDataFromXml();
```

You must now use:

```
$grid->setSource(new Bvb_Grid_Source_...);
```

Changes in the query() method.

```
$grid->query($select);//Can be used only for Zend_Db_Select and Zend_Db_Table_Abstract
Sources
```

Renamed methods

- `Bvb_Grid_ExtraColumns` have been renamed to `Bvb_Grid_Extra_Column`

Known Issues

- UNION queries not supported
- You may have some problems when using fields with the same name as the table name.
- Some users reported that in some situations they get duplicated content when using Zend_Layout
- Some users are reporting some issues when exporting results to charsets from oriental countries.

Contributors

Lead Developer

Bento Vilas Boas

JqGrid Implementation/Core Contributor

Martin Minka

Doctrine Implementation

Solomon James, The Faulkner, Martin Parsiegla

Documentation Revision

Dan Farrow

Community members, special reference to:

Braun Akos

David Quintard

Guus Teley

Ivo Monteiro

Thomas

Vlad Ovchinnikov

Vlatko Basic