MySQL Ajax Table Editor

Documentation

Getting Started

The best way to get started is by downloading MySQL Ajax Table Editor and modifying one of the examples to fit your needs.

Each MySQL Ajax Table Editor is powered by a configuration file which has the following sections:

- Initiate Editor Function
  - This is where all the table columns and most options are defined.
  - It is also where the MySQL Ajax Table Editor object is initialized.
  - This function contains a table columns array.
    - Each entry in this array defines a column to be displayed or edited.
    - Options and callback functions can be set in the table column arrays. This is what is referred to in the documentation as "Column Options" and "Column Callback Functions".
  - Initialization Of The MySQL Ajax Table Editor Object
    - Parameters
      - Table name
      - Primary column name (primary column to uniquely identify each row)
        - If the table doe not have unique primary column a mysql view can be used.
      - Error function (callback function to be executed when an error occurs)
      - Permissions string
      - Array of table columns
  - Once the MySQL Ajax Table Editor object has been initialized, other non-column specific options and callback functions can be defined by calling the set config function. This is what the documentation refers to as "Options" and "Callback Functions".
- Display html function
  - Function to display the basic html for the MySQL Ajax Table Editor framework.
  - This function is also used to set some configuration options (paid and pro versions).

Installation

1. Set the mysql variables in Common.php with the correct information to connect to your mysql database.
2. Create the example tables in your database by running the sql in the mate.sql file.
3. After you create the tables and have successfully connected to your database, the examples should be working. Point your browser to www.yoursite.com/path/to/mate/ and click on one of the example links. If you have any troubles installing please post a question on the forums.

Language Support

Currently MySQL Ajax Table Editor is available in Chinese, English, Dutch, French, German, Italian, Portuguese, Russian, Spanish and Turkish.

Creating a new translation can be done quickly by editing the php language variables file (php/lang/LangVars-en.php) and the javascript language variables file (js/lang/ang_vars-en.js). If you do create a new translation please send us the files and we will include the translation in the official release.

To use a different language you simply have to change the included javascript and php files. For example if you wanted to use Spanish instead of English you would change
require_once('../shared/php/mate/php/lang/LangVars-en.php');
to

require_once('../shared/php/mate/php/lang/LangVars-es.php');
and
<script type="text/javascript" src="js/lang/lang_vars-en.js"></script>
to
<script type="text/javascript" src="js/lang/lang_vars-es.js"></script>

Permissions

Permissions can be set for a column or for the entire table.
The available permissions are:
E - Edit
A - Add
C - Copy
V - View - Display a column on the view screen.
D - Delete
X - Export
Q - Quick Search
S - Advanced search
H - Hide - Allows users to show/hide columns.
O - Order - Allows users to change column position in the table, see paid demo for an example.
I - Icons - Displays icons on each row for edit, copy, view and delete options.
M - Multiple - Displays checkboxes on each row for multiple edit, copy, and delete actions. This is also used for user buttons with call back functions.
U - Allows the user to set the number of records displayed on each page.
T - Table - Display a column in the main table.
F - Filter - Use new filtering system **(paid and pro versions only)**.
In order to use the H and O options with the free and paid versions, you must have the mate_columns table created in your mysql database.

Set Table Permissions

```
function initiateEditor()
{
    ...
    $permissions = 'EAVDQCSXHOMF';
    $this->Editor = new AjaxTableEditor(
        $tableName,
        $primaryCol,
        $errorFun,
        $permissions,
        $tableColumns
    );
}
```

Set Column Permissions

```
function initiateEditor()
{
    $tableColumns['first_name'] = array(
        'display_text' => 'First Name',
        'perms' => 'EVCTAXQSHOF'
    );
    ...
}
```

Displaying Data

Table Screen

If a column is to be displayed in the table screen the T permission must be set for that column and for the table.

Table Screen Column Options

display_text - Set column label.

Example

```
function initiateEditor()
{
    $tableColumns['first_name'] = array('display_text' => 'First Name', 'perms' => 'EVCTAXQS');
    ...
}
```

display_mask - Apply mysql function(s) to the column data.

When using the join option, the display_mask option must be defined inside the join array.

Example

```
function initiateEditor()
{
    $tableColumns['full_name'] = array(
        'display_text' => 'Full Name',
        'perms' => 'VTXQ',
        'display_mask' => "concat(first_name,' ',last_name)"
    );
    ...
}
```

col_header_info - Set html attributes or css styles to go in the column header.

Example

```
function initiateEditor()
{
    $tableColumns['active'] = array(
        'display_text' => 'Active',
        'perms' => 'EVCTAXQ',
        'col_header_info' => 'width="150" style="border: 1px solid #333;"'
    );
    ...
}
```

sub_str - Truncate displayed data.

To avoid display issues, html tags will automatically be stripped when the data is truncated.

Example

```
function initiateEditor()
{
    $tableColumns['notes'] = array(
        'display_text' => 'Notes',
        'perms' => 'EVCTAXQSHO',
        'textarea' => array('rows' => 8, 'cols' => 25),
        'sub_str' => 30
    );
    ...
}
```

Table Screen Column Callback Functions

table_fun - Format, add or change data before it is displayed on the table screen.

Example

```
function initiateEditor()
{
        $tableColumns['url'] = array(
            'display_text' => 'URL',
            'perms' => 'EVCAXTQSFHO',
            'table_fun' => array(&$this,'formatLink'),
            'view_fun' => array(&$this,'formatLink')
        );
    ...
}

function formatLink($col,$val,$row)
{
    $html = '';
    if(strlen($val) > 0)
    {
        $html = '<a target="_blank" href="'.$val.'">'.$val.'</a>';
    }
    return $html;
}
```

Table Screen Options

tableInfo - Set html attributes that will be placed in the table tag.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('tableInfo','cellpadding="1" width="800" class="mateTable"');
}
```

oddRowColor - Set color of odd rows.

This option is only available in the free and paid versions. In the pro version the css classes "even" and "odd" can be modified to change the appearance of the rows.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('oddRowColor','#E0E0E0');
}
```

evenRowColor - Set color of even rows.

This option is only available in the free and paid versions. In the pro version the css classes "even" and "odd" can be modified to change the appearance of the rows.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('evenRowColor','#FFFFFF');
}
```

tableTitle - Set title on table screen.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('tableTitle','Employees');
}
```

displayNum - Set # of records to be displayed on each page.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('displayNum','30');
}
```

displayNumInc - Set the display increment for the user defined display drop down.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('displayNumInc','10');
}
```

maxDispNum - Set the maximum number of rows that can be displayed on one page.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('maxDispNum','100');
}
```

extraRowInfo - Add extra information to the table row tags (<tr> tags).

The strings #primaryColValue# and #rowNum# will be replaced with the corresponding values.

Example

```
function initiateEditor()
{
    ...
                $this->Editor->setConfig('extraRowInfo','onclick="showRowDetails(\'#primaryColValue#
\',\'#rowNum#\');" style="cursor: pointer;"');
}
```

paginationLinks - Use pagination links instead of select drop down to page through data.

This option is only available for the paid version.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('paginationLinks',true);
}
```

Table Screen Callback Functions

tableScreenFun - Callback function to manipulate html on the add screen.

Parameters - No data is passed to this callback function.

Return Values - No data must be returned by this function.

Remove Table HTML Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('tableScreenFun',array(&$this,'removeTableHtml'));
}

function removeTableHtml()
{
    $this->Editor->retArr[] = array('layer_id' => 'tableLayer', 'where' => 'innerHTML', 'value' => '');
}
```

modifyRowSets - Callback function to modify table row attributes and add additional attributes.

Parameters - An array of row sets (attributes), an array of row information and a row # will be passed to the callback function.

Return Values - An array of row sets must be returned by this function.

Change Background Color

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('modifyRowSets',array(&amp;$this,'changeBgColor'));
}

function changeBgColor($rowSets,$rowInfo,$rowNum)
{
    $rowSets['bgcolor'] = '#ffffff';
    if($rowInfo['account_type'] == 'Admin')
    {
        $rowSets['bgcolor'] = 'red';
    }
    return $rowSets;
}
```

Add Style

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('modifyRowSets',array(&amp;$this,'addStyle'));
}

function addStyle($rowSets,$rowInfo,$rowNum)
{
    if($rowInfo['account_type'] == 'Admin')
    {
        $rowSets['style'] .= ' font-weight: bold;';
    }
    return $rowSets;
}
```

Override Class

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('modifyRowSets',array(&amp;$this,'overrideClass'));
}

function overrideClass($rowSets,$rowInfo,$rowNum)
{
    if($rowInfo['account_type'] == 'Admin')
    {
        $rowSets['class'] = 'admin-row';
    }
    return $rowSets;
}
```

userColumns - Add your own user defined columns with and format them with a callback function.

Example

```
function initiateEditor()
{
    ...
    $userColumns[] = array('call_back_fun' => array(&$this,'getFullName'), 'title' => 'Full Name');
    $this->Editor->setConfig('userColumns',$userColumns);

}

function getFullName($row)
{
    $html = '<td>'.$row['first_name'].' '.$row['last_name'].'</td>';
    return $html;
}
```

View Screen

If a column is to be displayed in the view screen the V permission must be set for that column and for the table.

View Screen Column Options

display_mask - Apply mysql function(s) to the column data.

When using the join option, the display_mask option must be defined inside the join array.

Example

```
function initiateEditor()
{
    $tableColumns['full_name'] = array(
        'display_text' => 'Full Name',
        'perms' => 'VTXQ',
        'display_mask' => "concat(first_name,' ',last_name)"
    );
    ...
}
```

View Screen Options

viewRowTitle - Set the title on the view screen.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('viewRowTitle','View Employee');
}
```

View Screen Column Callback Functions

view_fun - Format, add or change data before it is displayed on the view screen.

Example

```
function initiateEditor()
{
        $tableColumns['url'] = array(
            'display_text' => 'URL',
            'perms' => 'EVCAXTQSFHO',
            'table_fun' => array(&$this,'formatLink'),
            'view_fun' => array(&$this,'formatLink')
        );
    ...
}

function formatLink($col,$val,$row)
{
    $html = '';
    if(strlen($val) > 0)
    {
        $html = '<a target="_blank" href="'.$val.'">'.$val.'</a>';
    }
    return $html;
}
```

Adding/Editing Data

To allow adding rows to a table the "A" permission must be set for the table. To allow input for a column on the add screen, the "A" permission must be set for that column.

To make a column editable, the "E" permission must be set for the column and for the table.

Add/Edit Column Options

default - Set the default value of a field.

Example

```
function initiateEditor()
{
    $tableColumns['status'] = array(
        'display_text' => 'Status',
        'perms' => 'EVCTAXQS',
        'default' => 'open'
    );
    ...
}
```

req - Make a field required.

Example

```
function initiateEditor()
{
    $tableColumns['first_name'] = array(
        'display_text' => 'First Name',
        'perms' => 'EVCTAXQS',
        'req' => true
    );
    ...
}
```

select_array - Associative array to generate a select drop down list.

The array keys will be the drop down values and the array values will be displayed in the drop down list.

Example

```
function initiateEditor()
{
    $statusArr = array(
        'open' => 'Open',
        'closed' => 'Closed',
        'pending' => 'Pending'
    );

    $tableColumns['status'] = array(
        'display_text' => 'Status',
        'perms' => 'EVCTAXQS',
        'select_array' => $statusArr,
        'default' => 'open'
    );
    ...
}
```

select_query - Query to generate a select drop down list.

The query must select 2 columns. The first column will be the drop down values and the second column will be displayed in the drop down list.

Example

```
function initiateEditor()
{
    $tableColumns['status'] = array(
        'display_text' => 'Status',
        'perms' => 'EVCTAXQS',
        'select_query' => "select status, status from status_table",
        'default' => 'open'
    );
    ...
}
```

hidden_add - Make an input field hidden on the add screen.

Example

```
function initiateEditor()
{
    $tableColumns['status'] = array(
        'display_text' => 'Status',
        'perms' => 'EVCTAXQS',
        'hidden_add' => true,
        'default' => 'open'
    );
    ...
}
```

hidden_edit - Make an input field hidden on the edit screen.

Example

```
function initiateEditor()
{
    $tableColumns['status'] = array(
        'display_text' => 'Status',
        'perms' => 'EVCTAXQS',
        'hidden_edit' => true
    );
    ...
}
```

mysql_add_fun - Use a mysql function when inserting a row.

The #VALUE# string will be replaced with the user input value.

Example 1

```
function initiateEditor()
{
    $tableColumns['password'] = array(
        'display_text' => 'Password',
        'perms' => 'EVCAXQT',
        'mysql_add_fun' => "PASSWORD('#VALUE#')",
        'mysql_edit_fun' => "PASSWORD('#VALUE#')",
    );
    ...
```

```
}
```

Example 2

```
function initiateEditor()
{
    $tableColumns['created'] = array(
        'display_text' => 'Created',
        'perms' => 'VCAXQT',
        'mysql_add_fun' => "NOW()",
    );
    ...
}
```

mysql_edit_fun - Use a mysql function when updating a row.

The #VALUE# string will be replaced with the user input value.

Example 1

```
function initiateEditor()
{
    $tableColumns['password'] = array(
        'display_text' => 'Password',
        'perms' => 'EVCAXQT',
        'mysql_add_fun' => "PASSWORD('#VALUE#')",
        'mysql_edit_fun' => "PASSWORD('#VALUE#')",
    );
    ...
}
```

Example 2

```
function initiateEditor()
{
    $tableColumns['updated'] = array(
        'display_text' => 'Updated',
        'perms' => 'EVCXQT',
        'mysql_edit_fun' => "NOW()",
    );
    ...
}
```

textarea - Use a text area in the add and edit screens.

Example

```
function initiateEditor()
{
    $tableColumns['notes'] = array(
        'display_text' => 'Notes',
        'perms' => 'EVCTAXQ',
        'textarea' => array('rows' => 5, 'cols' => 30)
    );
    ...
}
```

checkbox - Use a checkbox input in the add and edit screens.

When using the checkbox option you should also use the default option to set the default value.

Example

```
function initiateEditor()
{
    $tableColumns['active'] = array(
        'display_text' => 'Active',
        'perms' => 'EVCTAXQ',
        'checkbox' => array(
            'checked_value' => '1',
            'un_checked_value' => '0'
        ),
        'default' => '1'
    );
    ...
}
```

input_info - html attributes that will be added to the input, select or textarea tags.

Example

```
function initiateEditor()
{
    $tableColumns['user_name'] = array(
         'display_text' => 'User Name',
         'perms' => 'EVCTAXQS',
         'input_info' => 'onblur="checkUserName(this.value);"'
    );
    ...
}
```

file_upload - Upload a file to server (paid and pro versions).

delete_fun - Callback function that will be called to delete an uploaded file. When this option is set a delete icon will automatically be displayed next to the uploaded file name. See the pro demo for an example. This option is currently only available in the pro version.

upload_fun - Callback function that is called to handle the uploaded file. The following parameters will be passed to the callback function:

- The id of the inserted or updated row
- Column name where the upload was defined
- An array of information about the uploaded file
- An array of validation errors (array will be empty if no validation errors have occurred)

The upload function should return the array of validation errors which will then be displayed to the user if it is not empty. For more information see the "Upload To Directory" example below.

The following parameters are used when uploading the file to the database. For more information see the "Upload To Database" example below.

name - Specify the column where the name of the file should be stored.

type - Specify the column where the mime type of the file should be stored.

size - Specify the column where the size of the file should be stored.

max_size - Define the maximum file size (in bytes) that can be uploaded. A validation error will be displayed if the size is too big.

after_add_fun - Callback function to be called after a file has been added. The same parameters that are passed to the upload_fun callback will be passed.

after_edit_fun - Callback function to be called after a file has been updated. The same parameters that are passed to the upload_fun callback will be passed.

Upload To Directory

```
function initiateEditor()
{
    $tableColumns['file_name'] = array(
        'display_text' => 'Image',
        'perms' => 'EVCAXTQSFHO',
        'file_upload' => array(
            'upload_fun' => array(&$this,'handleUpload')
        ),
        'table_fun' => array(&$this,'formatImage'),
        'view_fun' => array(&$this,'formatImage')
    );
    ...
}

function formatImage($col,$val,$row)
{
    $html = '';
    if(strlen($val) > 0)
    {
                $html .= '<a target="_blank" href="uploads/'.$val.'"><img style="border: none;"
src="uploads/'.$val.'" alt="'.$val.'" width="100" /></a>';
    }
    return $html;
}

function handleUpload($id,$col,$filesArr,$valErrors)
{
    if(count($valErrors) == 0)
    {
        // Delete image file if it already existed
        $query = "select file_name from emp_upload_dir where id = '".$this->Editor->escapeData($id)."'";
        $result = $this->Editor->doQuery($query);
        if($row = mysql_fetch_assoc($result))
        {
            if(file_exists('uploads/'.$row['file_name']))
            {
                unlink('uploads/'.$row['file_name']);
            }
        }
        // Copy file to data directory and update database with the file name.
        if(move_uploaded_file($filesArr['tmp_name'],'uploads/'.$filesArr['name']))
        {
                                        $query = "update emp_upload_dir set file_name =
'".$this->Editor->escapeData($filesArr['name'])."' where id = '".$this->Editor->escapeData($id)."'";
            $result = $this->Editor->doQuery($query);
            if(!$result)
            {
                $valErrors[] = 'There was an error updating the database.';
                unlink('uploads/'.$filesArr['name']);
            }
        }
        else
        {
            $valErrros[] = 'The file could not be moved';
        }
    }
    return $valErrors;
}
```

Upload To Database

```
function initiateEditor()
```

```
{
        $tableColumns['file_data'] = array(
            'display_text' => 'Image',
            'perms' => 'EVCAXTQSFHO',
            'file_upload' => array(
                'type' => 'file_type',
                'size' => 'file_size',
                'name' => 'file_name'
            ),
            'display_mask' => 'file_name',
            'table_fun' => array(&$this,'formatImage'),
            'view_fun' => array(&$this,'formatImage')
        );
    ...
}

function formatImage($col,$val,$row)
{
    $html = '';
    if(strlen($val) > 0)
    {
                $html .= '<a target="_blank" href="DisplayFileFromDb.php?emp_id='.$row['id'].'"><img
style="border: none;" src="DisplayFileFromDb.php?emp_id='.$row['id'].'" alt="'.$val.'" width="100"
/></a>';
    }
    return $html;
}
```

null_array - Set an array that will be replaced with a null value in the database.

Example

```
function initiateEditor()
{
    $tableColumns['null_column'] = array(
        'display_text' => 'Null Column',
        'perms' => 'EVCTAXQSFHO',
        'null_array' => array('','0'),
    );
    ...
}
```

Add/Edit Column Callback Functions

Column callback functions are callback functions that can be defined for each column.

add_fun - Format, add or change data before it is displayed on the add screen.

Example

```
function initiateEditor()
{
    $tableColumns['active']= array(
        'display_text' => 'Active',
        'perms' => 'TAEVQS',
        'add_fun' => array(&$this,'replaceBool'),
        'display_mask' => 'replace(replace(active,"1","Yes"),"0","No")'
    );
    ...
}
```

```
function replaceBool($col,$val,$row)
{
    return str_replace(array('0','1'),array('No','Yes'),$val);
}
```

edit_fun - Format, add or change data before it is displayed on the edit screen.

Example

```
function initiateEditor()
{
    $tableColumns['active']= array(
        'display_text' => 'Active',
        'perms' => 'TAEVQS',
        'edit_fun' => array(&$this,'replaceBool'),
        'display_mask' => 'replace(replace(active,"1","Yes"),"0","No")'
    );
    ...
}

function replaceBool($col,$val,$row)
{
    return str_replace(array('0','1'),array('No','Yes'),$val);
}
```

on_add_fun - Format, add or change user input before insert.

In the pro version if the callback function returns false, the column will not be inserted. This can be used for columns that should only be inserted when they have a value.

Example

```
function initiateEditor()
{
    $tableColumns['active']= array(
        'display_text' => 'Active',
        'perms' => 'TAEVQS',
        'on_add_fun' => array(&$this,'replaceBool'),
        'display_mask' => 'replace(replace(active,"1","Yes"),"0","No")'
    );
    ...
}

function replaceBool($col,$val,$row)
{
    return str_replace(array('No','Yes'),array('0','1'),$val);
}
```

on_edit_fun - Format, add or change user input before update.

In the pro version if the callback function returns false, the column will not be updated. This can be used for password fields that should only be updated when they have a value.

Example

```
function initiateEditor()
{
    $tableColumns['active']= array(
        'display_text' => 'Active',
```

```
        'perms' => 'TAEVQS',
        'on_edit_fun' => array(&$this,'replaceBool'),
        'display_mask' => 'replace(replace(active,"1","Yes"),"0","No")'
    );
    ...
}

function replaceBool($col,$val,$row)
{
    return str_replace(array('No','Yes'),array('0','1'),$val);
}
```

val_fun - Check field validation on add and edit screens.

Example

```
function initiateEditor()
{
    $tableColumns['email'] = array(
        'display_text' => 'Email',
        'perms' => 'EVCTAXQS',
        'val_fun' => array(&$this,'valEmail')
    );
    ...
}

function valEmail($col,$val,$row)
{
    if(preg_match('/(@.*@)|(\.\.)|(@\.)|(\.@)|(^\.)/', $val) || preg_match('/^.+\@(\[?)[a-zA-Z0-9\-\.]+\.
([a-zA-Z]{2,3}|[0-9]{1,3})(\]?)$/',$val))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

format_input_fun - Format the input field on add and edit screens.

Example

```
function initiateEditor()
{
    $tableColumns['active']= array(
        'display_text' => 'Active',
        'perms' => 'TAEVQS',
        'format_input_fun' => array(&$this,'getRadioBtns'),
        'display_mask' => 'replace(replace(active,"1","Yes"),"0","No")'
    );
    ...
}

function getRadioBtns($colName,$value,$row)
{
    $opYes = 'checked="checked"';
    $opNo = '';
    if($value==0)
    {
        $opYes = '';
        $opNo = 'checked="checked"';
```

```
    }
      $html = '<label><input name="'.$colName.'" type="radio" id="'.$colName.'" value="1" '.$opYes.'
/>Yes</label><label><input name="'.$colName.'" type="radio" id="'.$colName.'" value="0"
onchange="updateRadioValueNo(this);"'.$opNo.' />No</label>';
    return $html;
}
```

Add/Edit Options

addRowTitle - Set the title on the add screen.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('addRowTitle','Add Employee');
}
```

editRowTitle - Set the title on the edit screen.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('editRowTitle','Edit Employee');
}
```

allowEditMult - Enable or disable multiple edit functionality.

MySQL ajax table editor has the ability to edit multiple rows. If you would like to disable that functionality and still have the ability to select multiple rows for delete and copy then you can set this variable to false.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('allowEditMult',false);
}
```

addInPlace - Enable in-line adding (pro version).

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('addInPlace',true);
}
```

editInPlace - Enable in-line editing (pro version).

Example

```
function initiateEditor()
```

```
{
    ...
    $this->Editor->setConfig('editInPlace',true);
}
```

persistentAddForm - Set whether the in-line add form will always be visible or not (pro version).

Set this option to false to make the in-line add form only visible after clicking an add button (default is true).

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('persistentAddForm',false);
}
```

Add/Edit Callback Functions

addScreenFun - Callback function to manipulate html on the add screen.

Parameters - No data is passed to this callback function.

Return Values - No data must be returned by this function.

Initiate CKEditor Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('addScreenFun',array(&$this,'addCkEditor'));
}

function addCkEditor()
{
    // Call javascript function to initialize ckeditor
    $this->Editor->retArr[] = array('where' => 'javascript', 'value' => 'addCkEditor("notes");');
    // Pro version code
    // $this->Editor->addJavascript('addCkEditor("'.$this->instanceName.'notes");');
}

<script type="text/javascript">
function addCkEditor(id)
{
    if(CKEDITOR.instances[id])
    {
     CKEDITOR.remove(CKEDITOR.instances[id]);
    }
    CKEDITOR.replace(id);
}
</script>
```

afterAddFun - Callback function that is called after a row is added.

Parameters - The new row's insert id, primary column name and an associative array of columns and values are passed to this callback function.

Return Values - No data must be returned by this function.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('afterAddFun',array(&$this,'changeTotal'));
    $this->Editor->setConfig('afterEditFun',array(&$this,'changeTotal'));
}

function changeTotal($id,$col,$info)
{
    // Code to update the total
}
```

editScreenFun - Callback function to manipulate html on the add screen.

Parameters - No data is passed to this callback function.

Return Values - No data must be returned by this function.

Initiate CKEditor Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('editScreenFun',array(&$this,'addCkEditor'));
}

function addCkEditor()
{
    // Call javascript function to initialize ckeditor
    $this->Editor->retArr[] = array('where' => 'javascript', 'value' => 'addCkEditor("notes");');
    // Pro version code
    // $this->Editor->addJavascript('addCkEditor("'.$this->instanceName.'notes");');
}

<script type="text/javascript">
function addCkEditor(id)
{
    if(CKEDITOR.instances[id])
    {
     CKEDITOR.remove(CKEDITOR.instances[id]);
    }
    CKEDITOR.replace(id);
}
</script>
```

afterEditFun - Callback function that is called after a row is edited.

Parameters - The primary column value, primary column name and an associative array of columns and values are passed to this callback function.

Return Values - No data must be returned by this function.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('afterEditFun',array(&$this,'changeTotal'));
    $this->Editor->setConfig('afterAddFun',array(&$this,'changeTotal'));
```

```
}

function changeTotal($id,$col,$info)
{
    // Code to update the total
}
```

viewScreenFun - Callback function to manipulate html on the add screen.

Parameters - No data is passed to this callback function.

Return Values - No data must be returned by this function.

Update View Buttons Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('viewScreenFun',array(&$this,'removeEditButton'));
}

function removeEditButton()
{
    $this->Editor->retArr[] = array(
        'layer_id' => 'viewRowButtons',
        'where' => 'innerHTML',
                    'value' => '<button class="ajaxButton" onclick="toAjaxTableEditor(\'update_html
\',\'\');">Back</button>'
    );
}
```

disableMultCbFun - Callback function that can be called to disable the multiple row checkbox.

Parameters - An associative array of columns and values for each row are passed to this callback function.

Return Values - True if the checkbox should be disabled and false if it should not.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('disableMultCbFun',array(&$this,'checkRowStatus'));
}

function checkRowStatus($info)
{
    if($info['status'] == 'closed')
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

addInPlaceFun - Callback function to manipulate html after the "add in place" form is displayed (pro version).

Parameters - Row number (to maintain compatibility with the editInPlaceFun callback) and the instance name

will be passed to this function.

Return Values - Nothing must be returned by this function.

Example

```
function initiateEditor()
{
...
$this->Editor->setConfig('addInPlaceFun',array(&$this,'initializeAutoComplete'));
}

function initializeAutoComplete($rowNum,$instanceName)
{
// Call javascript function to initialize auto complete
$this->Editor->retArr[] = array('where' => 'javascript', 'value' => 'initializeAutoComplete();');
}

<script type="text/javascript">
function initializeAutoComplete()
{
    // Javascript code to initialize auto complete.
}
</script>
```

editInPlaceFun - Callback function to manipulate html after the "edit in place" form is displayed (pro version).

Parameters - Row number and the instance name will be passed to this function.

Return Values - Nothing must be returned by this function.

Example

```
function initiateEditor()
{
...
$this->Editor->setConfig('editInPlaceFun',array(&$this,'initializeAutoComplete'));
}

function initializeAutoComplete($rowNum,$instanceName)
{
// Call javascript function to initialize auto complete
$this->Editor->retArr[] = array('where' => 'javascript', 'value' => 'initializeAutoComplete();');
}

<script type="text/javascript">
function initializeAutoComplete()
{
    // Javascript code to initialize auto complete.
}
</script>
```

Filtering/Searching Data

searchType - Set the default search type (either quick or advanced).

Example

```
function initiateEditor()
{
```

```
    ...
    $this->Editor->setConfig('searchType','advanced');
}
```

numAdvSearches - Set the default # of advanced searches when the page loads for the first time.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('defNumAdvSearches','2');
    $this->Editor->setConfig('numAdvSearches','2');
}
```

defNumAdvSearches - Set the default # of advanced searches when toggling between quick and advanced searches.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('defNumAdvSearches','2');
    $this->Editor->setConfig('numAdvSearches','2');
}
```

useHightlight - Turn highlighting on quick searches on or off (default is true).

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('useHightlight',false);
}
```

highlightHash - Set the highlight css for quick searches.

This option is only available for the free and paid versions of the software. For the pro version the highlight css class can be changed or overridden.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('highlightHash','color: red; font-weight: bold;');
}
```

sqlFilters - This option is used to add additional filters to the query.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('sqlFilters',"department = 'Engineering'");
```

```
}
```

data_filters - Filter data for a specific column.

criteria - Possible values are "any" or "all" the default is "all", which means all data filters will have to be matched for a row to be displayed. If criteria is set to "any" only one of the data filters will have to be matched for the row to be displayed.

Example

```
function initiateEditor()
{
    $tableColumns['first_name'] = array(
        'display_text' => 'First Name',
        'perms' => 'EVCAXQT',
        'data_filters' => array('filters' => array("like '%c%'","like '%f%'"), 'criteria' => 'any')
    );
    ...
}
```

Icons And Buttons

To display table icons, the I permission must be set for the table.

userIcons - Add custom icons.

For user icons to be displayed the I permission must be set.

Examples:

Example 1

```
function initiateEditor()
{
     ...
     $userIcons[] = array(
                'icon_html' => '<a onclick="iconAction('#primaryColValue#');" class="icon-class" title="icon-
title"></a>'
     );
     $this->Editor->setConfig('userIcons',$userIcons);
}
```

The string #primaryColValue# will be replaced with the primary column value. This string can be changed by setting the replaceWithId option.

Custom Function To Format Icons

```
function initiateEditor()
{
    ...
    $userIcons[] = array('format_fun' => array(&$this,'getUserIcons'));
    $this->Editor->setConfig('userIcons',$userIcons);
}

function getUserIcons($info)
{
    $iconHtml = '';
    $numIcons = 0;
     $iconHtml .= '<li class="delete"><a href="javascript: customDeleteRow(\''.$info['id'].'\');" title="Delete">
</a></li>';
```

```
    $numIcons++;
    return array('icon_html' => $iconHtml, 'num_icons' => $numIcons);
}
```

The current row's information will be passed to the callback function. The call back function must return an associative array in the following format array('icon_html' => 'html for icons', 'num_icons' => 'number of icons').

Custom Callback Function When Icon Is Clicked

```
function initiateEditor()
{
    ...
    $userIcons[] = array(
        'class' => 'delete',
        'title' => 'Delete',
        'call_back_fun' => array(&$this,'customDeleteRow'),
        'no_update' => false,
        'confirm_msg' => 'Are you sure?'
    );
    $this->Editor->setConfig('userIcons',$userIcons);
}

function customDeleteRow($info)
{
    $query = "delete from table_name where id = '".$this->escapeData($info['id'])."'";
    $result = $this->doQuery($query);
    if($result)
    {
        // code to delete associated records
    }
}
```

An associative array of the clicked row's information will be passed to the function.

Optional Parameters:
no_update - Set this to true if you do not want the table to be redrawn after the call back executes.
confirm_msg - Set a confirmation message that will appear before the call back executes.

userButtons - Add custom buttons.

Usage 1

```
function initiateEditor()
{
     ...
    $userButtons[] = array(
        'button_html' => '<button onclick="customJsFunction();">Custom Button</button>'
    );
    $this->Editor->setConfig('userButtons',$userButtons);
}
```

Usage 2

```
function initiateEditor()
{
     ...
    $userButtons[] = array(
        'label' => 'Archive',
        'call_back_fun' => array(&$this,'archiveRows'),
        'pass_id_array' => false,
        'confirm_msg' => 'Are you sure you would like to archive the selected rows?',
        'no_update' = false
```

```
    );
    $this->Editor->setConfig('userButtons',$userButtons);
}

function archiveRows($info)
{
    // Code to archive
}
```

Optional Parameters:
pass_id_array - Pass an array of ids to the callback function instead of executing the callback for each row, default is false.
confirm_msg - Set a confirmation message that will appear before the call back executes.
no_update - Set this to true if you do not want the table html to automatically update after the callback function is executed, default is false.

Usage 3

```
function initiateEditor()
{
    ...
    $userButtons[] = array(
        'label' => 'Button Label',
        'button_info' => 'onclick="runJsFun();" style="color: red;"'
    );
    $this->Editor->setConfig('userButtons',$userButtons);
}
```

iconTitle - Set a title for the icon column.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('iconTitle','Actions');
}
```

replaceWithId - String that will be replaced by the row id in the user icon html.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('replaceWithId','%primaryColValue%');
}
```

iconColPosition - Set the position of the add, edit, copy and delete icons.

Possible values are first and last.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('iconColPosition','first');
}
```

removeIcons - This option is used to allow editing, copying or deleting but remove the default icon.

This option can be used in conjunction with the userIcons option to insert custom icons or for conditional editing.

Example

```
function initiateEditor()
{
    ...
    // remove edit icon
    $this->Editor->setConfig('removeIcons','E');
}
```

Sorting/Ordering Data

orderByColumn - Set default column that the data will be ordered by.

Free Version

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('orderByColumn','first_name');
}
```

Paid And Pro Versions

```
function displayHtml()
{
    ...
    echo $html;

    // Set default session configuration variables here
    $defaultSessionData['orderByColumn'] = 'first_name';
    ...
}
```

ascOrDesc - Set default ascending or descending.

Free Version

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('ascOrDesc','asc');
}
```

Paid And Pro Versions

```
function displayHtml()
{
    ...
    echo $html;

    // Set default session configuration variables here
    $defaultSessionData['ascOrDesc'] = 'asc';
    ...
}
```

extraOrderByInfo - Set secondary order by columns.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('extraOrderByInfo','column_2 asc, column_3 desc');
}
```

Exporting Data

If a column is to be exported the X permission must be set for that column and for the table.

csv_export_fun - Format, add or change data before it is exported.

Column callback function to modify exported data.

Example

```
function initiateEditor()
{
    $tableColumns['active']= array(
        'display_text' => 'Active',
        'perms' => 'TAEVQS',
        'csv_export_fun' => array(&$this,'replaceBool'),
        'format_input_fun' => array(&$this,'getRadioBtns'),
        'display_mask' => 'replace(replace(active,"1","Yes"),"0","No")'
    );
    ...
}

function replaceBool($col,$val,$row)
{
    return str_replace(array('0','1'),array('No','Yes'),$val);
}
```

Copying Data

If column data is to be copied to a new row the C permission must be set for that column and for the table.

afterCopyFun - Callback function that is called after a row is copied.

Parameters - The new row's insert id and an associative array of columns and values are passed to this callback function.

Return Values - No data must be returned by this function.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('afterCopyFun',array(&$this,'copyAssociatedRecords'));
}

function copyAssociatedRecords($id,$info)
{
```

```
    // Code to copy associated records
}
```

on_copy_fun - Format, add or change data before copy.

Example

```
function initiateEditor()
{
    $tableColumns['created']= array(
        'display_text' => 'Created',
        'perms' => 'TACVQS',
        'on_copy_fun' => array(&$this,'updateToNow'),
    );
    ...
}

function updateToNow($col,$val,$row)
{
    return date('Y-m-d H:i:s');
}
```

Deleting Data

If table data is to be deleted, the D permission must be set for the table.

afterDeleteFun - Callback function that is called after a row is deleted.

Parameters - The deleted row's primary column value and primary column name are passed to this callback function.

In the pro version an associative array of deleted data is passed as the third parameter.

Return Values - No data must be returned by this function.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('afterDeleteFun',array(&$this,'deleteAssociatedRecords'));
}

function deleteAssociatedRecords($id,$col)
{
    // Code to delete associated records
}
```

Hiding Columns

To give users the ability to hide a column the H permission must be set for that column and for the table.

Hidden columns with the required validation option set will not be validated.

In the pro version custom validation callback functions will still be executed when a column is hidden, however in the paid and free versions they will not.

hidden - Make a column hidden by default.

In the free and paid versions hidden columns are not displayed on the add or edit screens. In the pro version hidden columns are displayed on the add and edit screens (unless editing in place).

In the free and paid versions no data about hidden columns is passed to callback functions. In the pro version data for hidden columns is passed to callback functions.

Example

```
function initiateEditor()
{
    $tableColumns['notes']= array(
        'display_text' => 'Notes',
        'perms' => 'EVCTAXQSHOF',
        'textarea' => array('rows' => 8, 'cols' => 25),
        'hidden' => true,
    );
    ...
}
```

showHideScreenFun - Callback function to manipulate html on the show/hide columns screen.

Parameters - No data is passed to this callback function.

Return Values - No data must be returned by this function.

This option is similar to the editScreenFun option which has example usage.

Ordering Columns

To give users the ability to change the order of a column the O permission must be set for that column and for the table.

Working With Dates

calendar - Make a javascript date selector calendar (free and paid versions).

format - The format option will set the date format. See all formats

reset - The reset option will add a reset icon to remove the date.

extra_info - The extra_info option is used to add extra text or html after the calendar (such as an asterick for required fields).

Example

```
function initiateEditor()
{
    $tableColumns['date'] = array(
        'display_text' => 'Date',
        'perms' => 'EVCTAXQS',
        'calendar' => array('format' => '%B %d, %Y', 'reset' => true, 'extra_info' => '')
    );
    ...
}
```

calendar - Make a javascript date selector calendar (pro version).

js_format - Set the date format. See all format options

options - Array to set other jquery ui datepicker options. See all date picker options

Example

```
function initiateEditor()
{
    $tableColumns['hire_date'] = array(
        'display_text' => 'Hire Date',
        'perms' => 'EATVXSQ',
        'display_mask' => 'date_format(`hire_date`,"%d %M %Y")',
        'order_mask' => 'date_format(`hire_date`,"%Y-%m-%d %T")',
        'range_mask' => 'date_format(`hire_date`,"%Y-%m-%d %T")',
        'calendar' => array(
            'js_format' => 'dd MM yy',
            'options' => array('showButtonPanel' => true)
        )
    )
    ...
}
```

order_mask - Use a MySQL function to change the date format when ordering data. (pro version).

Example

```
function initiateEditor()
{
    $tableColumns['hire_date'] = array(
        'display_text' => 'Hire Date',
        'perms' => 'EATVXSQ',
        'display_mask' => 'date_format(`hire_date`,"%d %M %Y")',
        'order_mask' => 'date_format(`hire_date`,"%Y-%m-%d %T")',
        'range_mask' => 'date_format(`hire_date`,"%Y-%m-%d %T")',
        'calendar' => array(
            'js_format' => 'dd MM yy',
            'options' => array('showButtonPanel' => true)
        )
    )
    ...
}
```

range_mask - Use a MySQL function to change the date format performing date range searches. (pro version).

Example

```
function initiateEditor()
{
    $tableColumns['hire_date'] = array(
        'display_text' => 'Hire Date',
        'perms' => 'EATVXSQ',
        'display_mask' => 'date_format(`hire_date`,"%d %M %Y")',
        'order_mask' => 'date_format(`hire_date`,"%Y-%m-%d %T")',
        'range_mask' => 'date_format(`hire_date`,"%Y-%m-%d %T")',
        'calendar' => array(
            'js_format' => 'dd MM yy',
            'options' => array('showButtonPanel' => true)
        )
    )
    ...
}
```

defaultJsCalFormat - Set the default date format for the javascript calendar (free and paid versions)..

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('defaultJsCalFormat','%B %d, %Y');
}
```

defaultDbDateFormat - Set the default database date format (pro version). Default is "Y-m-d H:i:s".

This option only needs to be changed when saving dates to the database in a non-standard format.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('defaultDbDateFormat','d F Y');
}
```

convertRangeDates - Auto convert user inputted dates (pro version). Default is true.

This option facilitates date range searches. When it is set to true, user inputted dates will automatically be converted to the the defaultDbDateFormat when performing date range searches. This allows users to enter any date format when searching on dates and the query will still be executed as expected.

User input will be converted ONLY if the calendar option is set.

Joining Tables

dbName - Set the database name for joining tables on different databases.

The join database can also be specified in the column join option.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('dbName','db_name');
}
```

join - Join a column to another table.

display_mask - The display_mask option is used to display different columns from the joined table or to apply mysql functions on the column.

type - The type option sets the join type. The default join type is inner.

db - The db option is used to join tables in a different database.

alias - Manually set the alias for the joined table. This is useful when pulling in other columns from the joined table or accessing the joined table with other options such as sqlFilters or orderByColumn.

real_column - With this option it is possible to use the same column to create multiple joins. The alias option can be used to pull in multiple columns from a joined table but if separate join instances are wanted on the same table

this option can be used.

Example 1

```
function initiateEditor()
{
    $tableColumns['employee_id'] = array(
        'display_text' => 'Name',
        'perms' => 'EVCTAXQ',
        'join' => array(
            'table' => 'employees',
            'column' => 'id',
            'display_mask' => "concat(employees.first_name,' ',employees.last_name)",
            'type' => 'left'
        )
    );
    ...
}
```

Alias Example

```
function initiateEditor()
{
    $tableColumns['employee_id'] = array(
        'display_text' => 'Name',
        'perms' => 'EVCTAXQ',
        'join' => array(
            'table' => 'employees',
            'column' => 'id',
            'display_mask' => "concat(employees.first_name,' ',employees.last_name)",
            'alias' => 'emp',
            'type' => 'left'
        )
    );

    // Bring in second column from joined table (notice the alias option above).
    $tableColumns['department'] = array(
        'display_text' => 'Department',
        'perms' => 'VTXQ',
        'display_mask' => "emp.department"
    );
    ...
}
```

customJoin - Define a custom join (pro version).

Custom joins are for reporting only. Editing and adding data is not supported when using custom joins.

Example

```
function initiateEditor()
{
    $tableColumns['employee_id'] = array(
        'display_text' => 'Name', 'perms' => 'EVCTAXQS',
        'display_mask' => "concat(emp.first_name,' ',emp.last_name)",
    );
    ...
     $this->Editor->setConfig('customJoin',"left join `employees` as `emp` on `login_info`.`employee_id` =
`emp`.`id`");
    ...
}
```

Custom Integrations

userActions - Set custom user actions that can be called from javascript events.

Example

```
function initiateEditor()
{
$tableColumns['user_name'] = array(
    'display_text' => 'User Name',
    'perms' => 'EVCTAXQS',
    'input_info' => 'onblur="toAjaxTableEditor(\'check_user_name\',this.value);"'
);
    ...
    $userActions['check_user_name'] = array(&$this,'checkUserName');
    $this->Editor->setConfig('userActions',$userActions);
}

function checkUserName($userName)
{
    $query = "select id from users where user_name = '".mysql_real_escape_string($userName)."'";
    $result = mysql_query($query);
    if($row = mysql_fetch_array($result,MYSQL_ASSOC))
    {
        $this->Editor->retArr[] = array(
            'where' => 'javascript',
            'value' => 'alert("This user name is already used")'
        );
    }
}
```

Javascript to call user action: "toAjaxTableEditor('action_name',this.value);". This javascript could be placed in an onchange tag in a drop down or an onblur tag in a text input or any other place where a javascript event can be triggered.

How to execute javascript and manipulate html from callback functions.

To execute javascript or manipulate html from a server side callback function, the global return array can be appended to.

Execute Javascript

```
function callBackFunction($col,$val,$row)
{
    $this->Editor->retArr[] = array(
        'where' => 'javascript',
        'value' => 'alert("test javascript")'
    );
}
```

Set Inner HTML

```
function callBackFunction($col,$val,$row)
{
    $this->Editor->retArr[] = array(
        'layer_id' => 'html_elem_id',
        'where' => 'innerHTML',
        'value' => '<p>new html goes here</p>'
    );
}
```

Set HTML Input Value

```
function callBackFunction($col,$val,$row)
{
    $this->Editor->retArr[] = array(
        'layer_id' => 'input_id',
        'where' => 'value',
        'value' => 'new value goes here'
    );
}
```

Pro Version

In the pro version the following functions can be used instead.

```
$this->Editor->addJavascript($javascript);
$this->Editor->setInnerHtml($layerId,$html);
$this->Editor->setHtmlValue($layerId,$value);
```

Dependencies

Free and Paid Versions

- Prototype Javascript Framework
- Script.aculo.us (only needed for hiding and ordering columns)
- Dynarch DHTML Calendar (optional calendar for selecting dates)

Pro Version

- jQuery Javascript Library
  - jQuery JSON Plugin
  - jQuery Cookie Plugin
- jQuery UI

Advanced SQL/Development

Grouping Data (pro version)

Grouping data is for reporting only. Editing and adding data is not supported when grouping data.

groupByClause - Define custom group by clause.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('groupByClause',"group by department");
}
```

havingFilters - Add custom having filters.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('havingFilters',"count(*) > 3");
```

```
}
```

having - Specify that column filters should be added to the having clause instead of the where clause.

Example

```
function initiateEditor()
{
    $tableColumns['count'] = array(
        'display_text' => 'Count',
        'perms' => 'VTXQSHOF',
        'display_mask' => "count(*)",
        'having' => true
    );
    ...
}
```

customJoin - Define a custom join (pro version).

Custom joins are for reporting only. Editing and adding data is not supported when using custom joins.

Example

```
function initiateEditor()
{
    $tableColumns['employee_id'] = array(
        'display_text' => 'Name', 'perms' => 'EVCTAXQS',
        'display_mask' => "concat(emp.first_name,' ',emp.last_name)",
    );
    ...
     $this->Editor->setConfig('customJoin',"left join `employees` as `emp` on `login_info`.`employee_id` =
`emp`.`id`");
    ...
}
```

viewQuery - Display the generated query.

Example

```
function initiateEditor()
{
    ...
    $this->Editor->setConfig('viewQuery',true);
}
```

Version Differences

Paid/Pro Versions

- In the paid and Pro versions session data is not stored in the php session variable.
- In the paid and pro versions some configuration options should be set differently than in the free version. Any setting that could potentially be changed later on by the user (such as the order by column, order by direction etc) should be set in the default session data array in the display html function instead of in the initiate editor function. Examples on how to set these options can be seen in the scripts that come with the paid and pro versions.

Pro Version

Currently there are several differences between the pro and the other versions of MySQL Ajax Table Editor. The free and paid versions will be updated soon to eliminate many of these differences.

- The pro version uses the jquery javascript library whereas the paid and free versions use the prototype javascript library.
- Hidden columns behavior is different in the pro version. See the "Hidding Columns" section for more info.
- The pro version uses cookies to store hidden columns, column order and other user options. This means the mate columns table is not needed as in other versions.
- Session state is also stored in a cookie (for example filters, page #, sorting etc) so that when the user leaves the page and then comes back the table is in the same state. This is very convenient for the user but can cause issues while developing. If this feature is causing issues during development simply comment out the pertinent javascript cookie code in the display html function.
- The pro version supports multiple tables in one page. To distinguish between the different instances there is a instance name configuration option.
    - The instance name is passed as the last parameter to all callback functions.
    - To call javascript functions with the MySQL Ajax Table Editor framework they must be prefraced with the instance name. For example instead of toAjaxTableEditor(params) it would be instance_name.toAjaxTableEditor(params).