Komponentenentwicklung unter Joomla! - Version 1.0

In diesem Tutorial wird erklärt, wie eine eigene Komponentenentwicklung stattfinden kann. Dafür wird jedoch keine Beispielkomponente entwickelt, sondern aufgezeigt, wie eigener Code in Joomla! / Mambo OS integriert werden muß, damit die Komponente im Backend und Frontend ordnungsgemäß arbeitet.
Ebenfalls wird die Installationsroutine behandelt und eine Auflistung der globalen Variablen.

Letzteres ist in dieser Version leider noch unzulänglich.

Erklärt wird hier nicht, wie PHP und MySQL funktionieren, da ich davon ausgehe, dass Komponenten nur von Menschen mit einem Grundwissen diesbezüglich entwickelt werden. Ungeachtet dessen, dass das natürlich auch den Umfang dieses Tutorials sprengen würde.

Dieses Tutorial ist im Zuge einer Zusatzausbildung, in der auch PHP gelernt wurde, entstanden. Vielleicht mag an einigen Stellen mein Code oder meine Kommentare etwas merkwürdig erscheinen – habt Nachsicht mit dem Lernenden - aber scheut euch nicht, Kritik und Verbesserungen zu üben. Wobei Lob natürlich auch gern entgegen genommen wird ;)

Wobei allerdings der meiste Code kopiert und mit eigenen Kommentaren versehen wurde.

Inhalt

1. Backend

1.1. Strukturen

1.1.1. Verzeichnisstruktur

/administrator/components/com_MeineKomponente
Eventuell eigene Sprachdateien, Images oder Hilfedateien in entsprechenden Unterverzeichnissen:
/administrator/components/com_MeineKomponente/help
/administrator/components/com_MeineKomponente/languages
/administrator/components/com_MeineKomponente/images
...

1.1.2. Dateienstruktur

Im Backend sind folgende Dateien notwendig:
admin.MeineKomponente.html.php
admin.MeineKomponente.php
toolbar.MeineKomponente.html.php
toolbar.MeineKomponente.php
Dazu können natürlich weitere Dateien entwickelt werden, die dann entsprechend der eigenen Verarbeitung dienen.

1.1.3. Version und Copyright

Im Kopf jeder Datei sollte – muß allerdings nicht – folgender Kommentar zu Anfang eingebunden werden (wobei weitere Hinweise natürlich möglich sind):

```
<?php
/**
* @version 1.0.0
* @package MeineKomponente
* @copyright Copyright (C) 2005 IhrName. All rights reserved.
* @license http://www.gnu.org/copyleft/gpl.html GNU/GPL, see LICENSE.php
*/
```

```
?>
```

1.2. Datei: admin.MeineKomponente.html.php

Diese Datei ist in erster Linie dafür verantwortlich, was im Backend angezeigt wird.

```
<?php

// Datei wird korrekt von Mambo/Joomla aufgerufen oder eben nicht und dann beendet
defined( '_VALID_MOS' ) or
   die( 'Direct Access to this location is not allowed.' );



// Funktion kann jederzeit wieder aufgerufen werden und somit die
// Administrationsebene  der Komponente angezeigt werden (Name beliebig. Wobei
// dieser Teil bei vielen Komponenten auch innerhalb einer Klasse eingebunden ist.).
        function anzeigeBackend() {
?>

...
Hier dann der eigene (HTML-)Code, der im Backend angezeigt wird.
...

// Abschluß der ‚anzeigeBackend'-Funktion
<?php
}
?>
```

1.3. Datei: admin.MeineKomponente.php

Der eigentliche Code. Also was passiert, wenn irgendwo geklickt, etc. wird? In dieser Datei wird alles verarbeitet. Wobei hier alles steht, was nicht der unmittelbaren Anzeige dient (z.B. DB-Abfragen)

```
// Der eigentliche Code muß zunächst in ein Formular eingeschlossen werden, da sonst
die Icons nicht reagieren (siehe auch Punkt 1.4.1)
<form name="adminForm">

<?php
defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );

// Handling der Tasks – Einbindung der Datei zum Handling der Ereignisse
require_once($mainframe->getPath('admin_html'));
```

```
// Abfrage der Icons oben rechts und Zuweisung an $task und die Abfrage der Formulardaten in $option (siehe unten)
$task=mosGetParam($_REQUEST,'task','');
$option = mosGetParam( $_REQUEST, 'option', '' );

?>

...
Platz für eigenen Code
...

// Die Auswertung von $task und damit der Icons
<?php
switch ($task) {
        // Beispiel:
        case 'save':
                save();
                break;


        default:
                vorbereitungBackend ();
                break;
}

function vorbereitungBackend()
{
SQL-Abfragen
…
// Aufruf der Funktion in admin.MeineKomponente.html.php
anzeigeBackend ();
}

function save()
{
...
}

…
Funktionen zur Verarbeitung von $task – wahlweise auch in anderen Dateien (ggf. includen)
...

?>


// Das Ende des einschließenden Formulars. Unsichtbar, da es nur zur Auswertung der Benutzereingaben gebraucht wird.
```

```
// Ohne dieser Zeile springt die Seite immer auf die Admin-Home-Seite zurück (in $option steht 'com_MeineKomponente').
<input type="hidden" name="option" value="<?php echo($option)?>" />
//Ohne dieser Zeile wird auf kein Ereignis reagiert
<input type="hidden" name="task" value="" />
</form>
```

1.4. Datei: toolbar.MeineKomponente.html.php

In dieser Datei wird die Iconleiste (Toolbar) oben rechts definiert.
Jede Funktion ermöglicht ein anderes Aussehen – also unterschiedliche Seiten, die ausgewählt wurden (Beispielsweise eine andere Toolbar, wenn ein Datensatz editiert wurde, etc.).

Bei einigen Methodenaufrufen lassen sich weitere Argumente übergeben. Eine detailierte Liste aller möglichen Icons und Übergabe-Parameter finden sich in der Datei (siehe auch Punkt 1.4.2):
/administrator/includes/menubar.html.php

1.4.1. Ereignisse auswerten

Um die einzelenen Toolbars (Icons) auswerten zu können, ist es erforderlich, die Datei, die Funktionen mit den Toolbars enthält, in einem Formular zu umschließen.

Die erste Zeile:

```
<form name="adminForm">
```

Und am Ende der Datei diese Zeilen einfügen:

```
<input type="hidden" name="option" value="<?php echo($option)?>" />
<input type="hidden" name="task" value="" />
</form>
```

Dazwischen kann dann beliebiger Code eingefügt werden.

Beispiel:

```
<form name="adminForm">
<?php
defined( '_VALID_MOS' ) or
   die( 'Direct Access to this location is not allowed.' );
function anzeigeBackend()
{
?>
```

(HTML-)Ausgabe

```
<?php
}
?>

<?php
function irgendwas()
{
?>
```

Andere Ausgabe

```
<?php
}
?>
<input type="hidden" name="option" value="<?php echo($option)?>" />
<input type="hidden" name="task" value="" />
</form>
```

Wichtig ist dabei, dass diese Zeilen am Anfang und Ende des Auswertungsbereichs stehen. Den Versuch, diese Zeilen lediglich in die einzelnen Funktionen zu schreiben, führte bei mir zu einem Fehlverhalten.

1.4.2. Mögliche Aufrufe (Kopie aus der Datei *menubar.html.php*)

Den einzelnen Toolbar-Icons können teilweise zusätzliche Parameter übergeben werden. Nachfolgend eine zusammenfassende Kopie aus der *menubar.html.php*. Beachten Sie auch den abschließenden Text zu 'help'.

```
/**
* Writes the start of the button bar table
*/
function startTable()

// Mit custom und customX lassen sich sehr einfach eigene Icons definieren
/**
* Writes a custom option and task button for the button bar
* @param string The task to perform (picked up by the switch($task) blocks
* @param string The image to display
* @param string The image to display when moused over
* @param string The alt text for the icon image
* @param boolean True if required to check that a standard list item is checked
*/
function custom( $task='', $icon='', $iconOver='', $alt='', $listSelect=true )

/**
```

```
* Writes a custom option and task button for the button bar.
* Extended version of custom() calling hideMainMenu() before submitbutton().
* @param string The task to perform (picked up by the switch($task) blocks
* @param string The image to display
* @param string The image to display when moused over
* @param string The alt text for the icon image
* @param boolean True if required to check that a standard list item is checked
*/
function customX( $task='', $icon='', $iconOver='', $alt='', $listSelect=true )

/**
* Writes the common 'new' icon for the button bar
* @param string An override for the task
* @param string An override for the alt text
*/
function addNew( $task='new', $alt='New' )

/**
* Writes the common 'new' icon for the button bar.
* Extended version of addNew() calling hideMainMenu() before submitbutton().
* @param string An override for the task
* @param string An override for the alt text
*/
function addNewX( $task='new', $alt='New' )

/**
* Writes a common 'publish' button
* @param string An override for the task
* @param string An override for the alt text
*/
function publish( $task='publish', $alt='Publish' )

/**
* Writes a common 'publish' button for a list of records
* @param string An override for the task
* @param string An override for the alt text
*/
function publishList( $task='publish', $alt='Publish' )

/**
* Writes a common 'default' button for a record
* @param string An override for the task
* @param string An override for the alt text
*/
function makeDefault( $task='default', $alt='Default' )

/**
* Writes a common 'assign' button for a record
* @param string An override for the task
* @param string An override for the alt text
```

```
*/
function assign( $task='assign', $alt='Assign' )


/**
* Writes a common 'unpublish' button
* @param string An override for the task
* @param string An override for the alt text
*/
function unpublish( $task='unpublish', $alt='Unpublish' )

/**
* Writes a common 'unpublish' button for a list of records
* @param string An override for the task
* @param string An override for the alt text
*/
function unpublishList( $task='unpublish', $alt='Unpublish' )

/**
* Writes a common 'archive' button for a list of records
* @param string An override for the task
* @param string An override for the alt text
*/
function archiveList( $task='archive', $alt='Archive' )

/**
* Writes an unarchive button for a list of records
* @param string An override for the task
* @param string An override for the alt text
*/
function unarchiveList( $task='unarchive', $alt='Unarchive' )

/**
* Writes a common 'edit' button for a list of records
* @param string An override for the task
* @param string An override for the alt text
*/
function editList( $task='edit', $alt='Edit' )

/**
* Writes a common 'edit' button for a list of records.
* Extended version of editList() calling hideMainMenu() before submitbutton().
* @param string An override for the task
* @param string An override for the alt text
*/
function editListX( $task='edit', $alt='Edit' )

/**
* Writes a common 'edit' button for a template html
* @param string An override for the task
```

```
* @param string An override for the alt text
*/
function editHtml( $task='edit_source', $alt='Edit HTML' )

/**
* Writes a common 'edit' button for a template html.
* Extended version of editHtml() calling hideMainMenu() before submitbutton().
* @param string An override for the task
* @param string An override for the alt text
*/
function editHtmlX( $task='edit_source', $alt='Edit HTML' )

/**
* Writes a common 'edit' button for a template css
* @param string An override for the task
* @param string An override for the alt text
*/
function editCss( $task='edit_css', $alt='Edit CSS' )

/**
* Writes a common 'edit' button for a template css.
* Extended version of editCss() calling hideMainMenu() before submitbutton().
* @param string An override for the task
* @param string An override for the alt text
*/
function editCssX( $task='edit_css', $alt='Edit CSS' )

/**
* Writes a common 'delete' button for a list of records
* @param string  Postscript for the 'are you sure' message
* @param string An override for the task
* @param string An override for the alt text
*/
function deleteList( $msg='', $task='remove', $alt='Delete' )

/**
* Writes a common 'delete' button for a list of records.
* Extended version of deleteList() calling hideMainMenu() before submitbutton().
* @param string  Postscript for the 'are you sure' message
* @param string An override for the task
* @param string An override for the alt text
*/
function deleteListX( $msg='', $task='remove', $alt='Delete' )

/**
* Write a trash button that will move items to Trash Manager
*/
function trash( $task='remove', $alt='Trash', $check=true )

/**
```

```
* Writes a preview button for a given option (opens a popup window)
* @param string The name of the popup file (excluding the file extension)
*/
function preview( $popup='', $updateEditors=false )

/**
* Writes a preview button for a given option (opens a popup window)
* @param string The name of the popup file (excluding the file extension for an xml file)
* @param boolean Use the help file in the component directory
*/
function help( $ref, $com=false )

/**
* Writes a save button for a given option
* Apply operation leads to a save action only (does not leave edit mode)
* @param string An override for the task
* @param string An override for the alt text
*/
function apply( $task='apply', $alt='Apply' )

/**
* Writes a save button for a given option
* Save operation leads to a save and then close action
* @param string An override for the task
* @param string An override for the alt text
*/
function save( $task='save', $alt='Save' )

/**
* Writes a save button for a given option (NOTE this is being deprecated)
*/
function savenew()

/**
* Writes a save button for a given option (NOTE this is being deprecated)
*/
function saveedit()

/**
* Writes a cancel button and invokes a cancel operation (eg a checkin)
* @param string An override for the task
* @param string An override for the alt text
*/
function cancel( $task='cancel', $alt='Cancel' )

/**
* Writes a cancel button that will go back to the previous page without doing
* any other operation
*/
function back( $alt='Back', $href='' )
```

```
/**
* Write a divider between menu buttons
*/
function divider()

/**
* Writes a media_manager button
* @param string The sub-drectory to upload the media to
*/
function media_manager( $directory = '', $alt='Upload' )

/**
* Writes a spacer cell
* @param string The width for the cell
*/
function spacer( $width='' )

/**
* Writes the end of the menu bar table
*/
function endTable()
```

Eine interessante Besonderheit stellt die Klassenmethode 'help' dar:
mosMenuBar::help('hilfe', true);
bedeutet das die Datei "hilfe.html"aufgerufen wird. Allerdings muß zuvor ein Unterverzeichnis *help* angelegt werden und die Datei muß sich in diesem Verzeichnis befinden.
Wird *false* angegeben, wird die "Joomla.org" Hilfe aufgerufen und das Schlüsselword *hilfe* übergeben. Befindet sich in der "globalen" Hilfe ein Eintrag zu dem Wort, wird dieser angezeigt.

```
<?php

defined( '_VALID_MOS' ) or
   die( 'Direct Access to this location is not allowed.' );

// Klasse mit Methoden – Namen frei vergebbar
class toolbarProgramm {
        function EditMenu()
        {
                // Start der Iconleiste
                mosMenuBar::startTable();
                // Speichern-Icon
                mosMenuBar::save();
                // Abbrechen-Icon
                mosMenuBar::cancel();
```

```
                // Neu-Icon
                mosMenuBar::addNew();
                // Editieren-Icon
                mosMenuBar::editList();
                // Löschen-Icon
                mosMenuBar::deleteList();
                // Ende der Iconleiste
                mosMenuBar::endTable();
        }

        function WeiteresMenu()
        {
                mosMenuBar::startTable();
                // Zurück-Icon
                mosMenuBar::back();
                // Kleiner Zwischenraum zwischen den Icons
                mosMenuBar::spacer();
                // Senkrechter Trennstrich zwischen den Icons
                mosMenuBar::divider();
                // Übernehmen-Icon
                mosMenuBar::apply();
                mosMenuBar::endTable();
        }
}

 ?>
```

1.5. Datei: toolbar.MeineKomponente.php

In dieser Datei wird die case-Anweisung zur Auswertung der Variablen ***$task*** geschrieben und damit festgelegt, welche Toolbar wann aufgerufen wird.
Die eigentliche Verarbeitung der einzelnen Icons findet in der Datei *admin.MeineKomponente.php* statt.

```
<?php
defined( '_VALID_MOS' ) or
   die( 'Direct Access to this location is not allowed.' );

// include support libraries
require_once( $mainframe->getPath( 'toolbar_html' ) );

// Handling der Tasks
$task = mosGetParam( $_REQUEST, 'task', '' );

switch ($task) {
        // Beispielaufruf
        case 'list'
```

```
        toolbarProgramm::WeiteresMenu();
        break;

        // Hier ist der Aufruf zur Standard-Toolbar der Komponente
        default:
        toolbarProgramm::EditMenu ();
        break;
}


?>
```

1.6. Zusammenfassung

Die beiden toolbar-Dateien regeln die Iconleiste oben rechts. Wobei die *toolbar.meine.**html**.php* für die sichtbare Anzeige zuständig ist und die *toolbar.meine.php* für die Verarbeitung der Iconereignisse insofern zuständig ist, dass hier festgelegt wird, welche Iconleiste bei welcher Auswahl angezeigt wird.

Ähnlich verhält es sich bei den beiden admin-Dateien.
Die *admin.meine.**html**.php* ist dafür verantwortlich, was im Backend angezeigt wird und die Datei *admin.meine.php* ist für die Verarbeitung der Ereignisse zuständig.

Wird mit Hilfedateien gearbeitet, dann müssen diese in einem Unterverzeichnis Namens *help* innerhalb des Admin-Komponenten-Verzeichnisses liegen und mit der Suffix *html* enden.

Bei eigenen Sprachdateien empfiehlt sich ein eigenes Unterverzeichnis Namens *languages*.

2. Symbole und Globales

2.1. Tabellenlisting

Um ein Tabellenlisting in gewohnter Form auszugeben, wird mit der Admin-CSS-Datei gearbeitet und die entsprechenden Klassen aufgerufen. Die CSS-Dateien liegen in:
administrator/templates/.../css/
Nur vosrichtshalber sei darauf hingewiesen, dass diese CSS-Datei Bestandteil des Admin-Templates ist und somit individuell aussehen kann. Ergo würde es nichts bringen, ein Backend zu entwickeln, welches bestimmte Farben in der CSS-Datei vorraussetzt!

Für den Kopf des "Managers" ist die CSS-Klasse *adminheading* verantwortlich, für die oberste Zeile des eigentlichen Listings *adminlist*. Die einzelnen Zeilen werden mit den beiden tr-Klassen *row0* und *row1* gekennzeichnet.

```
<?php
function TabList(
<table class="adminheading">
        <tr>
                <th>
                        // Das Icon vor der Tabellenüberschrift wird automatich
                          hinzugefügt (über der CSS-Datei)
                        Beispiel Listing
                </th>
                <td>
                        // Das Suchenfeld
                        Filter: <input type="text" name="search" value="<?php echo
                        $search;?>" class="text_area"
                        onChange="document.adminForm.submit();" />
                </td>
        </tr>
</table>

// Das folgende Code-Fragment habe ich aus der admin.modules.html.php entnommen
(also der Datei, die u.a für die Anzeige des Modulmanagers zuständig ist) und leicht
modifiziert
<table class="adminlist">
        <tr>
                <th width="20px">
                        #
                </th>
                <th width="20px">
                        // Checkbox mit der alle Checkboxen ausgewählt werden können
                        <input type="checkbox" name="toggle" value=""
                        onclick="checkAll(<?php echo count( $rows );?>);" />
                </th>
                <th class="title">
                        Name
                </th>
                <th nowrap="nowrap" width="10%">
                        Published
                </th>
                <th colspan="2" align="center" width="5%">
                        Reorder
                </th>
                <th width="2%">
                        Order
                </th>
                <th width="1%">
                        // Das Diskettensymbol zum speichern der Reihenfolge
```

```
                <a href="javascript: saveorder( <?php echo count( $rows )-1; ?> )">
                <img src="images/filesave.png" border="0" width="16" height="16"
                alt="Save Order" /></a>
        </th>
        <th nowrap="nowrap" width="7%">
                Access
        </th>
        <th nowrap="nowrap" width="5%">
                Pages
        </th>
        <th nowrap="nowrap" width="5%">
                ID
        </th>
</tr>
// ab hier die einzelnen Tabellenzeilen mit der Auflistung der erstellten Objekte
<?php
        // Zunächst eine Schleife, um die einzelnen Tabellenzeilen farblich zu
        unterscheiden – also die beiden row-CSS-Klassen zu erzeugen (wobei $rows
        zuvor übergeben werden muß! – Kann beispielsweise mit folgenden Zeilen
        erzeugt werden:
        $query=[SQL-ANWEISUNG]
        $database->setQuery( $query );
        $rows = $database->loadObjectList();
        Diese Zeilen sollten in der admin.MeineKomponente.php stehen und dann
        per Funktionsaufruf entsprechen übergeben werden.
        $k = 0;
        for ($i=0, $n=count( $rows ); $i < $n; $i++) {

        // Initialisierung der einzelnen Zeile
        $row = &$rows[$i];

        // Leider habe ich nachfolgende Klasse niergends finden können und kann
        daher nichts weiter zu ihr sagen …
        $access         = mosCommonHTML::AccessProcessing( $row, $i );
        $checked        = mosCommonHTML::CheckedOutProcessing( $row, $i );
        $published      = mosCommonHTML::PublishedProcessing( $row, $i );
?>
<tr class="<?php echo "row$k"; ?>">
        <td align="right">
                // Ausgabe: # (Laufende Nummer)
                <?php echo $pageNav->rowNumber( $i ); ?>
        </td>
        <td>
                 // Ausgabe: Checked-Formularfeld
                <?php echo $checked; ?>
        </td>
        <td>
                // Ausgabe: Name
                <?php echo $row->title;?>
        </td>
```

```
		<td align="center">
			// Ausgabe: Modul/Komponente veröffentlicht/unveröffentlicht
			<?php echo $published;?>
		</td>
		// Ausgabe: Icons für Sortierung aufwärts/abwärts
		<td>
			<?php echo $pageNav->orderUpIcon( $i, ($row->position ==
			@$rows[$i-1]->position) ); ?>
		</td>
		<td>
			<?php echo $pageNav->orderDownIcon( $i, $n, ($row->position ==
			@$rows[$i+1]->position) ); ?>
		</td>
		<td align="center" colspan="2">
			// Ausgabe: Zahlenfeld für Sortierung
			<input type="text" name="order[]" size="5" value="<?php echo
			$row->ordering; ?>" class="text_area" style="text-align: center" />
		</td>
		<td align="center">
			// Ausgabe: Berechtigungsstufe
			<?php echo $access;?>
		</td>
		<td align="center">
			// Ausgabe: Komponente wird auf einer/mehreren/allen Seiten
			angezeigt
			<?php
			if (is_null( $row->pages )) {
				echo 'None';
				} else if ($row->pages > 0) {
					echo 'Varies';
				} else {
					echo 'All';
				}
				?>
		</td>
		<td align="center">
			// Ausgabe: ID
			<?php echo $row->id;?>
		</td>
		</tr>
			// Abschluß der "row-Schleife"
			<?php
			$k = 1 - $k;
		}
		?>
</table>
```

Es gibt eine Reihe von globalen Klassen und Variablen in Mambo/Joomla.

Interessant ist z.B. die Datei *database.php* im Rootverzeichnis.
Ich bin selbst noch am entdecken und ausprobieren – werde aber diesen Teil weiter ergründen und das Tutorial entsprechend erweitern.
Für einen ersten Eindruck sollte jedoch das obige Beispiel zunächst genügen.

Wer mag, kann mir gern dazu schreiben und helfen.

3. Frontend

3.1. Strukturen

3.1.1. Verzeichnisstruktur

Das eigene Verzeichnis sollte, wie auch schon im Adminbereich mit *com_* beginnen. Das ist zwar nicht zwingend notwendig, erleichtert jedoch immens den Überblick …
/components/com_MeineKomponente/

3.1.2. Dateienstruktur

Im obigen Verzeichnis muß eine php-Datei mit dem Komponentennamen liegen. Mehr ist nicht notwendig. Eventuelle weitere Dateien ergeben sich aus der individuellen Programmierung.
MeineKomponente.php

3.2. Datei: MeineKomponente.php

Diese Datei ist für die Anzeige im Frontend verantwortlich und wird letztlich auch per Menüpunkt oder Verlinkung aufgerufen. Einfacher HTML-Code reicht völlig aus. Aber zur Anzeige von Datenbankausgaben, etc. ist natürlich etwas mehr erforderlich ;)

```
// Neben der bereits oben erwähnten PHP-Kommentar-Einbindung zu Beginn der Files,
// kann hier auch mit dem 'mos:comment'-Tag ein Kommentar eingebunden werden:
<mos:comment>
@version 1.0.0
@package MeineKomponente
@copyright Copyright (C) 2005 IhrName. All rights reserved.
```

@license http://www.gnu.org/copyleft/gpl.html GNU/GPL, see LICENSE.php
</mos:comment>

// Hier folgt der (HTML-)Code zur Anzeige der Komponente im Frontend

4. Installation

Zur Installation wird unbedingt eine XML-Datei benötigt. Bei Bedarf kann auch ein Install-PHP-File erstellt werden, welches aus der XML-Datei aufgerufen wird. Alles muß am Ende, einschl. der richtigen Verzeichnisse, in der die Dateien liegen werden, gepackt werden. Entweder als ZIP- oder als TAR-Datei.

4.1. XML-Datei

```
<?xml version="1.0" ?>
<mosinstall type="component">
Der Name Ihrer Komponente
  <name>MeineKomponente</name>
Erstellungsdatum
  <creationDate>29.12.2005</creationDate>
Ihr Name oder Firmenname
<author>Axel Tüting</author>
Lizensrechtliches. Wenn die Komponente kostenpflichtig ist, dann sollten Sie hier einen entsprechenden Vermerk eingeben
<copyright>This component is released under the GNU/GPL License</copyright>
<authorEmail>tueting@time4mambo.de</authorEmail>
<authorUrl>www.time4mambo.de</authorUrl>
Version der Komponente
<version>1.0</version>

Wenn keine Kurzbeschreibung vorgesehen ist, dann kann dieser XML-Tag auch weggelassen werden
<description>Diese Komponente verdient auch eine Kurzbeschreibung, die dann hier steht.</description>

Alle Files die im Frontend-Bereich installiert werden. Wenn Images oder Dateien, die in Unterverzeichnisse liegen, installiert werden, dann müssen sowohl hier die Pfade entsprechend angegeben werden, als auch im gepackten File diese Dateien in den entsprechenden Verzeichnissen liegen
<files>
        <filename>MeineKomponente.php</filename>
        <filename>index.html</filename>
</files>
```

```
        Wenn Grafiken vorhanden sein sollten – kann ansonsten weggelassen werden
        <images>
                <filename>images/meineGrafik.jpg</filename>
</images>
<install>
        <queries>
                Soviele query wie Tabellen, bzw. Inserts
                <query>DROP TABLE IF EXISTS
                `jom_MeineKomponente_settings`;</query>
                <query>CREATE TABLE `jom_MeineKomponente_settings` (…)</query>
                <query>INSERT INTO `jom_MeineKomponente_settings`(…);</query>
        </queries>
</install>
<uninstall>
        <queries>
                <query>DROP TABLE IF EXISTS
                'jom_MeineKomponente_settings`;</query>
</queries>
</uninstall>

Wenn kein Install-File vorhanden ist, kann dieser XML-Tag auch weggelassen werden
<installfile>
        <filename>install.MeineKomponente.php</filename>
</installfile>

Wenn kein Uninstall-File vorhanden ist, kann dieser XML-Tag auch weggelassen
werden
<uninstallfile>
        <filename>uninstall. MeineKomponente.php</filename>
</uninstallfile>
<administration>
        Zunächst das Menü im Backend:
        <menu>MeineKomponente</menu>
        Und das oder die Untermenüs, sofern vorhanden
        <submenu>
                task muß von Ihnen im Programmcode entsprechend verarbeitet sein
                <menu task="aufruf">Administration</menu>
        </submenu>
        <files>
                Hier müssen alle Dateien aufgeführt sein, die im Admin-Bereich stehen.
                Dabei gilt gleiches mit den Verzeichnissen wie weiter oben unter <files>
                <filename>index.html</filename>
                <filename>admin. MeineKomponente.php</filename>
                <filename>admin. MeineKomponente.html.php</filename>
                <filename>toolbar. MeineKomponente.html.php</filename>
                <filename>toolbar. MeineKomponente.php</filename>
        </files>
        Wenn Grafiken vorhanden sein sollten – kann ansonsten weggelassen werden
        <images>
                <filename>images/meineGrafik.jpg</filename>
```

```
        </images>
</administration>
</mosinstall>
```

Wenn Sie den obigen Code für Ihre Entwicklung kopieren, denken Sie bitte daran, meinen Namen durch Ihre Daten zu ersetzen ;)
So tauchen immer mal wieder Templates auf, in deren template-xml mein Name steht, weil der entsprechende Template-Designer achtlos den Code aus meinem Template-Tutorial kopiert hat …

4.2. Install-File

Eine extra Install-Datei ist nicht notwendig. Manche schreiben dort jedoch ein Dankeschön für die Installation der Komponente rein, andere schreiben dort aber auch Zusatzinstallationen rein, wie beispielsweise das Anhängen eigener Sprach-DEFINEs an den vorhandenen Sprachfiles. Weitere Möglichkeiten sind natürlich auch möglich.

```
<?php
function com_install()
{
…
}
?>
```

5. Uninstall

Für das Uninstall-File gilt das gleiche, wie für das Install-File. Eine Datei, die nicht vorhanden sein muß. Meist eine Ausgabe mit dem Hinweis, dass alles sauber deinstalliert wurde oder dem Dankeschön, fürs ausprobieren der Komponente.

```
<?php
function com_uninstall()
{
…
}
?>
```

6. Index.html

In den Verzeichnissen empfiehl sich eine einfache index.html zu legen, um unerlaubtes Aufrufen der Verzeichnisse zumindest mit einfachen Mitteln zu erschweren.

```
<html><body bgcolor="#FFFFFF"></body></html>
```

Diese Zeile als *index.html* abgespeichert reicht vollkommen.

7. Sprachdateien

Zum einen gibt es die Möglichkeit eigene Sprachdateien zu erzeugen, die dann im Admin-Verzeichnis oder/und im Komponentenbereich des Frontend der Komponente liegen sollten. Dann muß aber an eine Auswahlmöglichkeit innerhalb der Konfiguration für die Komponente gedacht werden.

Besser (meiner Meinung nach) ist es allerdings, seine eigenen DEFINEs an die vorhandene german.php und/oder english.php dran zu hängen, da dann der User automatisch bei Aufruf der Komponente gleich die richtige Sprache sieht.