Mambo Component Tutorial

Daily Message Component

By Joseph Leblanc
contact@jlleblanc.com

Table of Contents

Introduction

This guide should help you build a functioning component, with a complete backend. After reading through this site, you should understand how some of the core classes in Mambo function so that you can employ the best methods for building your own component. A finished, fully installable version of the Daily Message component is available at http://www.jlleblanc.com/comtutor/com_dailymessage.zip. This component tutorial is valid as of Mambo 4.5 1.0.7.

Different people will understand the functionality of Mambo components in different ways. Developers with significant previous PHP experience may wish to start with dailymessage.php (the file generating the frontend display) and admin.dailymessage.php (generates the backend display). Others will want to start with the XML document which maps out every code source, image, and SQL query.

The component developed in this tutorial contains these files:

dailymessage.php

admin.dailymessage.php

admin.dailymessage.html.php

dailymessage.class.php

toolbar.dailymessage.php

toolbar.dailymessage.html.php

install.dailymessage.php / uninstall.dailymessage.php

dailymessage.xml

dailymessage.php

This file displays what site visitors will see when the Daily Message component is loaded. Although this code is simple, it is all that is needed to generate HTML for any Mambo installation. It is the only file that is copied to the components/com_dailymessage directory. Let's step through the component:

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

This line secures your component by making sure that Mambo and only Mambo is calling this file.

```
global $database;
```

The $database object is declared elsewhere in Mambo. 'global' makes it available here.

```
// get configuration information
$database->setQuery("SELECT * FROM #__joe_dailymessage_conf LIMIT 1");
$rows = $database->loadObjectList();
$row = $rows[0];
```

Running a MySQL query in Mambo is a two step process. First, you use the setQuery() function to enter the SQL statement. Then you use a function such as loadObjectList() to run and retrieve the results. The line $row = $rows[0] stores the variables from the first object in the list in the array $row. In this case, we are retrieving the configuration information from the database, which should only be one record.

Also, in MySQL queries through Mambo, '#_' is replaced with the appropriate table name prefix. This makes it possible for applications aside from Mambo capable of sharing the same database (currently 'mos'). This functionality can be helpful if you are running on a budget host that only allows one database.

```
$bold = $row->bold;
$italic = $row->italic;
$underline = $row->underline;
$showdate = $row->showdate;
```

After loading the result into an array, we are able to retrieve the individual values from the query result into variables we can use later.

```
if($bold == 1)
{
	$starttags .= "<b>";
	$endtags = "</b>" . $endtags;
}

if($italic == 1)
{
	$starttags .= "<i>";
	$endtags = "</i>" . $endtags;
}

if($underline == 1)
{
	$starttags .= "<u>";
	$endtags = "</u>" . $endtags;
}
```

These statements build ending and beginning tag strings based on the settings from the database. These strings will be used below to format the message.

```
//get data and output accordingly
$database->setQuery("SELECT * FROM #__joe_dailymessage");
$rows = $database->loadObjectList();
```

Using the same $database object as above, we set the query to now pull from the table containing messages, then pull all the objects into $row.

```
?><table><?
```

Now we start outputting HTML code. The ?> allows us to escape from PHP temporarily to output what will be rendered as HTML until we reach a <? token. This produces the same results as echo().

```
foreach($rows as $row)
{
	if($showdate == 1)
		echo "<tr><td>" . mosFormatDate($row->date) . "</td>";
	else
		echo "<tr>";
	echo "<td>" . $starttags . $row->message . $endtags . "</td></tr>";
}
```

This loop cycles through each object in $rows and loads the contents into $row. Then we are able to pull the information out and build strings that will render properly formatted HTML.

```
?></table>
```

Outputs the closing table tag.

admin.dailymessage.php

This file controls which screen the administrator sees when configuring the component. Most of the actual HTML is generated through the HTML_joeDailyMessage [there's something to be said for narcissism :)] found in the admin.dailymessage.html.php file.

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

// ensure user has access to this function
if (!($acl->acl_check( 'administration', 'edit', 'users', $my->usertype, 'components',
        'all' )
        | $acl->acl_check( 'administration', 'edit', 'users', $my->usertype,
        'components', 'com_newsfeeds' ))) {
        mosRedirect( 'index2.php', _NOT_AUTH );
}
```

This performs a check to make sure that the user logged in is the administrator.

```
require_once( $mainframe->getPath( 'admin_html' ) );
require_once( $mainframe->getPath( 'class' ) );
```

These two lines include the files admin.dailymessage.html.php and dailymessage.class.php. The getPath() function returns the appropriate full paths and filenames.

```
$id = mosGetParam( $_REQUEST, 'cid', array(0) );
if (!is_array( $id )) {
        $id = array(0);
}
```

The main screen has a form with a list of checkboxes bearing the name 'cid.' The mosGetParam() function retrieves this array and stores it in $id. If the array is not present (such as when the first page loads), $id is set to an empty array to prevent errors later.

```
switch($act)
{
        case "configure":
        switch($task) {
                case "save":
                        saveConfiguration($option);
                        break;
```

```
                default:
                        listConfiguration($option);
                        break;
        }
        break;

        default:
        switch ($task) {
                case "save" :
                        save($option);
                        break;

                case "saved" :
                        echo "Saved!";
                        listMessages($option);
                        break;

                case "edit" :
                        edit( $option, $id );
                        break;

                case "new" :
                        $id = '';
                        edit( $option, $id);
                        break;

                case "delete" :
                        del($option, $id);
                        break;

                case "publish" :
                        publishMessage($option, '1', $id);
                        break;

                case "unpublish" :
                        publishMessage($option, '0', $id);
                        break;

                case "listMessages" :
                        default:
                        listMessages($option);
                        break;
        }
        break;
}
```

Now would be a good time to discuss the difference between $task and $act. When the administrator clicks on a button such as "publish" or "save," the selection must be communicated to the component. This is accomplished through the variable $task. These buttons are used for many different forms

and it is necessary for the component to know which form is being dealt with at the moment. This is where the variable $act comes in. On the main components menu, the selection for Daily Message has two submenu options: Edit Messages and Configure. These options are referred to by $act as 'all' and 'configure,' respectively.

The Daily Message component uses nested switch statements to determine which course of action to take. The outer switch statement is based on the $act variable, as both 'all' and 'configure' use overlapping task names. So the value of $act is first determined, then $task.

```
function saveConfiguration($option) {
        global $database;
        $row = new joeDailyMessageConf($database);
```

Now the definition of functions to handle various tasks begins. This first one will update the database with new variables when someone clicks the 'save' button on the configuration page. First, the $option variable is passed to the function and should be set to 'com_dailymessage' by the Mambo environment. This will allow the code to later forward the user back to the administration panel for this component. Next, the $database object is pulled from outside the function. Then we declare an object $row of type joeDailyMessageConf [defined in dailymessage.class.php], which is an extension of mosDBTable. Objects based on mosDBTable have many functions simplifying the process of recording information into the database.

```
// bind it to the table
if (!$row -> bind($_POST)) {
        echo "<script> alert('"
                .$row -> getError()
                ."'); window.history.go(-1); </script>\n";
        exit();
}
```

This code attempts to use the inherited mosDBTable function bind() to get the form variables from the post and copy them into the $row object. If the attempt is unsuccessful (for instance, the variable names do not match or are of the wrong type), a Javascript error is displayed and the code is halted. The Javascript error is displayed as a popup alert with a button that returns to the previous page.

```
// store it in the db
if (!$row -> store()) {
	echo "<script> alert('"
		.$row -> getError()
		."'); window.history.go(-1); </script>\n";
	exit();
}
```

This piece of code attempts to store the information in the $row object into the database. Error handling is similar to that of the section above.

```
	mosRedirect("index2.php?option=$option&act=configure",
		"Configuration Saved");
}
```

Finally, if everything is successful, the user is redirected back to the configuration page and the message "Configuration Saved" is displayed at the top of the screen in friendly orange lettering. The variable $option, passed to the function from before, contains the string 'com_dailymessage' and tells Mambo to pull up the administration panel for the Daily Message component. Notice that the URL has the 'act' variable set to 'configure.' If this is not done, the user will be forwarded back to the default Daily Message administration page: message editing.

```
function listConfiguration($option)
{
	global $database;
	$database->setQuery("SELECT * FROM #__joe_dailymessage_conf"  );
	$rows = $database -> loadObjectList();
```

This function loads the current configuration from the database and displays it on a form where the variables can be edited to suit the desires of the administrator. The appropriate query is called and the result is loaded into the object $rows as a list of objects.

```
if ($database -> getErrorNum()) {
	echo $database -> stderr();
	return false;
}
```

If there was an error processing the query, this code will display that error and stop the function, yet still allowing Mambo to finish loading the page without the configuration data.

```
        HTML_joeDailyMessage::listConfiguration($option, $rows);
}
```

This calls the function listConfiguration from the object HTML_joeDailyMessage from the dailymessage.class.php file, passing along the component name and object with configuration information.

```
function publishMessage( $option, $publish=1 ,$cid )
{
        global $database, $my;
        if (!is_array( $cid ) || count( $cid ) < 1) {
                $action = $publish ? 'publish' : 'unpublish';
                echo "<script> alert('Select an item to $action');
                window.history.go(-1);</script>\n";
                exit;
        }
```

When the administrator wants to publish a message, this function sets the 'published' flag on that message to '1.' First, we determine whether or not the administrator chose items to publish. If not, an error must be displayed referring back to the previous page. The line where $action is set is a shorthand version of an 'if' statement. The variable is set to 'publish' if it is true that $publish equals 1, otherwise, the variable is set to 'unpublish.' A Javascript alert is displayed, forwarding the user back.

```
$cids = implode( ',', $cid );
```

If we have items to publish, we set the string $ids to be a list of all the variables in the array $id and separate the variables with commas. This will simplify insertion into the SQL statement.

```
$database->setQuery( "UPDATE #__joe_dailymessage SET published='$publish'"
. "\nWHERE id IN ($cids)"
);
if (!$database->query()) {
        echo "<script> alert('".$database->getErrorMsg()."'); window.history.go(-1);
                </script>\n";
        exit();
}
```

A query is sent to the database and any errors are displayed through Javascript.

```
        mosRedirect( "index2.php?option=$option" );
}
```

Finally, the administrator is forwarded back to the main page of the Daily Message administration interface.

```
function save($option) {
        global $database;
        $row = new joeDailyMessage($database);

        // bind it to the table
        if (!$row -> bind($_POST)) {
                echo "<script> alert('"
                        .$row -> getError()
                        ."'); window.history.go(-1); </script>\n";
                exit();
        }

        // store it in the db
        if (!$row -> store()) {
                echo "<script> alert('"
                        .$row -> getError()
                        ."'); window.history.go(-1); </script>\n";
                exit();
        }
        mosRedirect("index2.php?option=$option", "Saved");
}
```

This function is nearly identical to saveConfiguration(), only it declares row to be of type joeDailyMessage and redirects to the default (message editing) page with the message "Saved."

```
function del($option, $cid) {
        global $database;
        if (!is_array($cid) || count($cid) < 1) {
                echo "<script> alert('Select an item to delete');
                window.history.go(-1);</script>\n";
                exit();
        }
```

A list of daily messages to delete is passed to this function. If there are no items in the array $cid [or $cid is not an array], an appropriate error message is displayed and the administrator is forwarded back to the previous page.

```
if (count($cid))
{
        $ids = implode(',', $cid);
        $database->setQuery("DELETE FROM #__joe_dailymessage \nWHERE id IN
        ($ids)");
}
```

If messages to be deleted have been selected, set a up a query that will remove them from the database.

```
        if (!$database->query()) {
                echo "<script> alert('"
                .$database -> getErrorMsg()
                ."'); window.history.go(-1); </script>\n";
        }
        mosRedirect("index2.php?option=$option");
}
```

The deletion query is executed and the administrator is forwarded back to the default screen.

```
function edit($option, $uid) {
global $database;

if($uid){
        $row = new joeDailyMessage($database);
        $row -> load($uid[0]);
}
```

When the administrator selects a daily message to edit, this function loads up the appropriate message from the database into the $row object. Since the administrator can only edit one message at a time, the first id in the $uid array [0] is chosen.

```
HTML_joeDailyMessage::edit($option, $row);
}
```

The edit() function is called from the HTML_joeDailyMessage class, which generates the appropriate HTML output for the screen. The database row and the current component name (com_dailymessage) are passed so that appropriate code can be generated.

```
function listMessages($option) {
global $database;

$database->setQuery("SELECT * FROM #__joe_dailymessage ORDER BY id"  );
```

This final function is called to display the default screen. If no task has been chosen, this function will list all of the daily messages so that the administrator can edit, delete, and publish/unpublish them. The query is set to retrieve the entire contents of the daily message table. They are ordered by id as that field is set as an automatically incrementing field. This way, unique keys can be generated and messages will be listed in the order of creation.

```
        $rows = $database -> loadObjectList();
        if ($database -> getErrorNum()) {
                echo $database -> stderr();
                return false;
        }
        HTML_joeDailyMessage::listMessages($option, $rows);
        }

?>
```

The query is executed and if there are any errors, they are simply written on the resulting page. The listMessages() function in the HTML_joeDailyMessage class is then called to generate the necessary HTML for the list screen.

admin.dailymessage.html.php

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

This line makes sure this code is being called from Mambo and not by some stray visitor.

```
class HTML_joeDailyMessage{
```

This class is primarily called form admin.dailymessage.php and handles the majority of the HTML output for the administrator interface.

```
function edit( $option, &$row ) {
```

This function takes the name of the component (com_dailymessage) and a database row loaded from code in admin.dailymessage.php and displays a screen where the administrator can edit a daily message.

```
?>

<form action="index2.php" method="post" name="adminForm" id="adminForm"
        class="adminForm">
<table border="0" cellpadding="3" cellspacing="0">

<tr>
<td>Message: </td>
<td><input type="text" size="50" maxsize="100" name="message" value="<?php
        echo $row->message; ?>" /></td>
</tr>

<tr>
<td>Date: </td>
<td><input size="30" name="date" value="<? echo $row->date; ?>"></td>
</tr>

<tr>
<td>Published: </td>
<td><? echo mosHTML::yesnoSelectList( "published", "", $row->published ); ?>
</td>
</tr>
```

If a database row has been supplied, the fields of the form are populated. Noticed that the 'Published' element is handled by the mosHTML class function yesnoSelectList. The mosHTML class is declared in classes/mambo.php and is available in throughout Mambo. This function

generates a dropdown list with the values 'no' and 'yes'. If the value of the variable $row-published is 0, 'no' will be selected. If the value is 1, 'yes' will be selected. This can be very helpful for creating administrative interfaces and is handled more gracefully than a checkbox.

```
</table>
<input type="hidden" name="id" value="<?php echo $row->id[0]; ?>" />
<input type="hidden" name="option" value="<?php echo $option; ?>" />
<input type="hidden" name="task" value="" />
</form>
<? }
```

Finally, the hidden variables containing the daily message id, the name of the component, and the task are placed in the form and output ends. The 'task' variable is actually set to null so that the Javascript from the toolbar can fill it in later depending on which button is clicked.

```
function listConfiguration($option, &$rows)
{
```

This function is called to display the configuration panel where the administrator can format the display of the daily messages.

```
?>
<form action="index2.php" method="post" name="adminForm">

<table cellpadding="4" cellspacing="0" border="0" width="100%"
        class="adminlist">
<?
$row = $rows[0];
?>
<tr><td>Bold</td><td><? echo mosHTML::yesnoSelectList( "bold", "",
        $row->bold ); ?></td></tr>
<tr><td>Italic</td><td><? echo mosHTML::yesnoSelectList( "italic", "",
        $row->italic ); ?></td></tr>
<tr><td>Underline</td><td><? echo mosHTML::yesnoSelectList( "underline", "",
        $row->underline ); ?></td></tr>
<tr><td>Show Date</td><td><? echo mosHTML::yesnoSelectList( "showdate", "",
        $row->showdate ); ?></td></tr>
</table>
```

This form is populated with the data from the first row in the mos_joe_dailymessage_conf table [there should only be one row]. The yesnoSelectList() function makes it simple to ensure the correct values are represented on the form.

```
<input type="hidden" name="option" value="<?php echo $option; ?>" />

<input type="hidden" name="task" value="" />

<input type="hidden" name="configid" value=<? echo $row->configid ?> />

<input type="hidden" name="act" value="configure" />

</form>

        <?
}
```

Finally the option, task, configid, and act variables are hidden in the form. The 'configid' variable ensures that we update the same configuration record, although there should only be one. The 'act' variable tells the admin.dailymessage.php code to either save or cancel for the configuration screen and not the message screen. This is handled by the switch statement towards the beginning of that file.

```
function listMessages( $option, &$rows ) {
        ?>
```

Finally, this function renders the default screen the administrator sees when clicking on 'Edit Messages' on the main menu.

```
<form action="index2.php" method="post" name="adminForm">

<table cellpadding="4" cellspacing="0" border="0" width="100%"
        class="adminlist">
<tr>
<th width="20"><input type="checkbox" name="toggle" value=""
        onclick="checkAll(<?php echo count($rows); ?>);" /></th>
<th class="title" width="25%">Message</th>
<th>Date</th>
<th width="25%">Published</th>
</tr>
```

This table header labels all of the columns and provides a master checkbox that will check every message on the form when clicked.

```
<?php
$k = 0;
for($i=0; $i < count( $rows ); $i++) {
        $row = $rows[$i];
        ?>
```

This starts a loop that cycles through each of the database rows in the $rows array.

```
<tr class="<?php echo "row$k"; ?>">
```

Pulls the style for the row from CSS based on whether this is row 0 or row 1. We start with row 0 and change it to row 1, back and forth, to create an alternating effect making the table easier to read.

```
<td><input type="checkbox" id="cb<?php echo $i;?>" name="cid[]" value="<?php
        echo $row->id; ?>" onclick="isChecked(this.checked);" /></td>
```

Displays a checkbox and provides a way of identifying it later when the form is submitted. Each checkbox id is stored as cbX, where X is the numerical position in the list the checkbox resides at. The name is specified as an array and the value is set to the id for the row found in the database.

```
<td><a href="#edit" onclick="return listItemTask('cb<?php echo
        $i;?>','edit')"><?php echo $row->message; ?></a></td>
```

Creates links containing the text of the message. When clicked, the link will automatically check the box at the beginning of the row and tell the Javascript to submit the form with the task of 'edit.'

```
<td><? echo mosFormatDate($row->date); ?></td>
```

Outputs the date of the message, displayed according to the local format through the mosFormatDate() function which is declared in classes/mambo.php.

```
<td align="center">
<?php
if ($row->published == "1") {
        echo "<a href=\"javascript: void(0);\" onClick=\"return
                listItemTask('cb$i','unpublish')\"><img src=\"images/publish_g.png\"
                border=\"0\" /></a>";
} else {
        echo "<a href=\"javascript: void(0);\" onClick=\"return
                listItemTask('cb$i','publish')\"><img src=\"images/publish_x.png\"
                border=\"0\" /></a>";
}
?>
</td>
```

This displays either the 'published' or 'unpublished' icon, depending on the status of the message. If the icon is clicked, the form is submitted with the opposite status as the task.

```
<?php $k = 1 - $k; ?>
```

Changes the variable that determines the background color of the row. Ones become zeros and zeros become one.

```
</tr>

<?php }
?>

<input type="hidden" name="option" value="<?php echo $option; ?>" />

<input type="hidden" name="task" value="" />

<input type="hidden" name="boxchecked" value="0" />

</form>
</table>

<? }
```

Hides the variables option, task, and boxchecked at the end of the form. Both 'boxchecked' and 'task' are later modified by Javascript. Boxchecked is used to make sure that items are checked before a task is executed.

```
}

?>
```

dailymessage.class.php

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

Yet again, this code should not be called from outside Mambo.

```
class joeDailyMessage extends mosDBTable {
	var $id = null;
	var $message = null;
	var $date = null;
	var $published = null;
```

This class extends the modDBTable class, which provides many useful functions that make it easier to transfer data to and from the database. First, the variables identical to those found in the database table are declared. This way, other pieces of code can create joeDailyMessage objects, load them up with the variables desired, then send them to the database.

```
	function joeDailyMessage(&$db){
		$this->mosDBTable('#__joe_dailymessage', 'id', $db);
	}
}
```

This constructor takes a database object passed by reference and passes it along to the inherited mosDBTable() constructor, along with the table name [#_ shorthand for 'mos'] we wish to reference and name of the field containing the primary key.

```
class joeDailyMessageConf extends mosDBTable {
	var $bold = null;
	var $italic = null;
	var $underline = null;
	var $showdate = null;
	var $configid = null;

	function joeDailyMessageConf(&$db){
		$this->mosDBTable('#__joe_dailymessage_conf', 'configid', $db);
	}
}
```

The joeDailyMessageConf class works in exactly the same way as the joeDailyMessage class, containing the variables found in the mos_joe_dailymessage_conf table and referencing 'configid' as the key.

```
?>
```

toolbar.dailymessage.php

For all components, Mambo loads toolbar code for the administrator interface. The creation of this code is fairly straightforward as the functionality is very well defined. The code in this file is essentially one switch statement that loads the appropriate toolbar based on the 'task' and 'act' variables.

```
<?
defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );
require_once( $mainframe->getPath( 'toolbar_html' ) );
```

First, make sure the code is running within mambo and load up the toolbar.dailymessage.html.php file, which will handle the HTML output for the menu.

```
switch ( $task ) {
	case 'edit':
		menuDailyMessage::EDIT_MENU();
		break;

	case 'new':
		menuDailyMessage::EDIT_MENU();
		break;

	default:
	switch($act) {
		case "configure":
			menuDailyMessage::CONFIGURE_MENU();
			break;

		default:
			menuDailyMessage::DEFAULT_MENU();
			break;
	}
	break;
}
```

Notice that both the 'new' and 'edit' tasks share the same menu. This does not cause a conflict, because the menu merely launches tasks and is not functionally dependent on the current task. The current task and 'act' merely determine which options will be displayed. All of the menus are called through functions in the menuDailyMessage class contained in the toolbar.dailymessage.html.php file.

```
?>
```

toolbar.dailymessage.html.php

This file defined the class menuDailyMessage which has three functions for the three different menu arrangements used. These are laid out through the mosMenuBar class functions which handle standard toolbars for the administrator interface.

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

Of course, the code should only be executed from Mambo.

```
class menuDailyMessage{
	function DEFAULT_MENU() {
		mosMenuBar::startTable();
		mosMenuBar::publish('publish');
		mosMenuBar::unpublish('unpublish');
		mosMenuBar::divider();
		mosMenuBar::addNew('new');
		mosMenuBar::editList('edit', 'Edit');
		mosMenuBar::deleteList( ' ', 'delete', 'Remove' );
		mosMenuBar::endTable();
	}
```

For most of these functions, the first parameter is the 'task' variable to be submitted. For the deleteList() function, the first variable is an optional alternative message for the "are you sure you want to delete this" alert. The second variable is the 'task', and the final variable provides the image 'alt' property to display. This function is available for the other buttons as well. The startTable() and endTable() functions output the appropriate beginning and end of table code, as well as some Javascript to handle the buttons. The divider() function displays a vertical bar that visually separates the icons.

```
function EDIT_MENU() {
	mosMenuBar::startTable();
	mosMenuBar::back();
	mosMenuBar::spacer();
	mosMenuBar::save('save');
	mosMenuBar::endTable();
}
```

The function back() simply displays the 'cancel' icon with the text 'back.' When clicked, it returns to the previous page. The spacer() functions provides some extra blank space between icons.

```
	function CONFIGURE_MENU() {
		mosMenuBar::startTable();
		mosMenuBar::save('save');
		mosMenuBar::endTable();
	}
}

?>
```

install.dailymessage.php / uninstall.dailymessage.php

The install.dailymessage.php and uninstall.dailymessage.php files provide a place to output additional messages when the component is installed and uninstalled. Daily Message does not have any such additional output.

install.dailymessage.php:

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

uninstall.dailymessage.php:

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

dailymessage.xml

```
<?xml version="1.0" ?>
```

This tag defines the version of XML used for the document. Most XML files in circulation are written in version 1.0. You can read more about XML at www.w3c.org.

```
<mosinstall type="component">
```

The mosinstall tag tells Mambo what to extension to prepare for. This code is installing a component as opposed to a module or template.

```
<name>DailyMessage</name>
<creationDate>06/03/2004</creationDate>
<author>Joseph LeBlanc</author>
<copyright>This component in released under the GNU/GPL License</copyright>
<authorEmail>contact@jlleblanc.com</authorEmail>
<authorUrl>www.jlleblanc.com</authorUrl>
<version>1.0</version>
```

These tags identify the component with a name, a creation date, an author, a copyright, an author e-mail and website, and a version number. This information is later displayed in the component administration section.

```
<files>
<filename>dailymessage.php</filename>
</files>
```

The <files> tags enclose a list of files for the frontend display which are individually enclosed by <filename> tags. In this case, only one file is uses

```
<install>
<queries>
        <query>
        DROP TABLE IF EXISTS `#__joe_dailymessage`;
        </query>
        <query>
        CREATE TABLE `#__joe_dailymessage` (
        `id` INT NOT NULL AUTO_INCREMENT,
        `message` TEXT NOT NULL,
        `date` DATETIME NOT NULL,
        `published` TINYINT(1) NOT NULL,
        PRIMARY KEY (`id`)
        )
        </query>
```

```
        <query>
        DROP TABLE IF EXISTS `#__joe_dailymessage_conf`;
        </query>
        <query>
        CREATE TABLE `#__joe_dailymessage_conf` (
        `bold` TINYINT(1) NOT NULL,
        `italic` TINYINT(1) NOT NULL,
        `underline` TINYINT(1) NOT NULL,
        `showdate` TINYINT(1) NOT NULL,
        `configid` TINYINT(4) NOT NULL
        )
        </query>
        <query>
        INSERT INTO `#__joe_dailymessage_conf`(bold, italic, underline, showdate,
        configid) values(0, 0, 0 , 1, 1);
        </query>
</queries>
</install>
```

The installation queries that set up the database tables used by the component are enclosed by <query> tags, which are all enclosed by <install> tags. The string '#_' in the queries is replaced with the Mambo database prefix [usually 'mos']. Two tables are created: one for the daily messages and one for the component configuration. Also, initial values are inserted into the configuration table.

```
<uninstall>
<queries>
        <query>
                DROP TABLE IF EXISTS `#__joe_dailymessage`;
        </query>
        <query>
                DROP TABLE IF EXISTS `#__joe_dailymessage_conf`;
        </query>
</queries>
</uninstall>
```

Similar to the installation tags, the <uninstall> tags enclose the queries necessary to remove the tables used by the component. These queries are only executed when the administrator desires to remove the component.

```
<installfile>
        <filename>install.dailymessage.php</filename>
</installfile>

<uninstallfile>
        <filename>uninstall.dailymessage.php</filename>
</uninstallfile>
```

The <installfile> and <uninstallfile> tags enclose the names of the files to copy that handle additional output for the installation and uninstallation of the component. These are copied to the administrator/com_dailymessage folder.

```
<administration>
        <menu>Daily Message</menu>
        <submenu>
                <menu act="all">Edit Messages</menu>
                <menu act="configure">Configure</menu>
        </submenu>
```

The <administration> tags enclose everything needed for the backend interface. The <menu> tags enclose the title that appears on the Mambo administration menu under 'Components.' The <submenu> tags enclose the menu items that appear when 'Daily Message' is selected from the 'Components' menu. The property 'act' is used by the backend code to determine which administration screen to display.

```
<files>
        <filename>admin.dailymessage.php</filename>
        <filename>admin.dailymessage.html.php</filename>
        <filename>dailymessage.class.php</filename>
        <filename>toolbar.dailymessage.php</filename>
        <filename>toolbar.dailymessage.html.php</filename>
</files>
```

The <files> tags within the <administration> tags designate which files to copy to the administration/com_dailymessage folder.

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