Phorum Developer Reference Manual

Maurice Makaay, Brian Moon, Thomas Seifert, Andy Taylor, and Joe Curia

August 24, 2009

Contents

1 Templates **7**
- 1.1 Introduction . . . 7
- 1.2 Template structure . . . 7
- 1.3 How to start your own template . . . 9
- 1.4 The Phorum template language . . . 10
  - 1.4.1 Introduction . . . 10
  - 1.4.2 General syntax . . . 10
  - 1.4.3 Data types . . . 11
    - 1.4.3.1 Integers . . . 11
    - 1.4.3.2 Strings . . . 12
    - 1.4.3.3 PHP constants . . . 13
    - 1.4.3.4 Template variables . . . 13
  - 1.4.4 Statements . . . 14
    - 1.4.4.1 Display a variable . . . 15
    - 1.4.4.2 In line comments . . . 15
    - 1.4.4.3 DEFINE . . . 16
    - 1.4.4.4 VAR . . . 16
    - 1.4.4.5 IF .. ELSEIF .. ELSE 17
    - 1.4.4.6 LOOP . . . 18
    - 1.4.4.7 INCLUDE . . . 18
    - 1.4.4.8 HOOK . . . 19
  - 1.4.5 Need the power of PHP? . . . 20

2 Modules **21**
- 2.1 Introduction . . . 21
- 2.2 Terminology . . . 21
  - 2.2.1 Modules . . . 21
  - 2.2.2 Hacks . . . 22
  - 2.2.3 Add-ons . . . 22
  - 2.2.4 Hooks . . . 22
  - 2.2.5 Hook functions . . . 23
- 2.3 Writing your own modules . . . 23
  - 2.3.1 Introduction . . . 23
  - 2.3.2 Module information . . . 23

2.3.3 Module file structure . . . 25
2.3.3.1 Introduction . . . 25
2.3.3.2 Single file modules . . . 25
2.3.3.3 Multiple file modules . . . 27
2.3.4 Supporting multiple languages . . . 29
2.3.5 Module data storage . . . 30
2.3.5.1 Introduction . . . 30
2.3.5.2 Storing data for messages . . . 30
2.3.5.2.1 From hooks that get an editable message array as their argument . . . 31
2.3.5.2.2 From other hooks . . . 32
2.3.5.3 Storing data for users . . . 33
2.3.5.3.1 Custom profile fields for users . . . 33
2.3.5.3.2 From hooks that get an editable user array as their argument . . . 34
2.3.5.3.3 From other hooks . . . 35
2.3.6 Building URLs for Phorum . . . 35
2.3.6.1 Introduction . . . 35
2.3.6.2 Build URLs for Phorum PHP scripts: phorum_get_url() 36
2.3.6.3 Build URLs to files in the Phorum tree . . . 37
2.3.7 Implementing a settings screen for your module . . . 38
2.3.7.1 Building input forms . . . 38
2.3.7.2 Error and success feedback messages . . . 38
2.3.7.3 Saving module settings to the database . . . 38
2.3.7.4 Prevent settings.php from being loaded directly . . . 39
2.3.7.5 Full module settings page example . . . 39

3 Module hooks **41**
3.1 Introduction . . . 41
3.2 Message search . . . 41
3.2.1 search_redirect . . . 41
3.2.2 search_output . . . 42
3.3 Templating . . . 42
3.3.1 javascript_register . . . 42
3.3.2 javascript_filter . . . 43
3.3.3 css_register . . . 43
3.3.4 css_filter . . . 45
3.4 Login/Logout . . . 45
3.4.1 before_logout . . . 45
3.4.2 after_logout . . . 45
3.4.3 password_reset . . . 46
3.4.4 after_login . . . 47
3.4.5 failed_login . . . 48
3.5 Private message system . . . 49
3.5.1 pm_delete_folder . . . 49
3.5.2 pm_delete . . . 50

3.5.3 pm_list . . . 50
3.5.4 pm_read . . . 51
3.6 Buddies system . . . 51
3.6.1 buddy_list . . . 51
3.7 Module hooks . . . 52
3.7.1 bbcode_register . . . 52
3.8 Message handling . . . 53
3.8.1 posting_init . . . 53
3.8.2 posting_permissions . . . 53
3.8.3 posting_custom_action . . . 54
3.8.4 before_editor . . . 55
3.8.5 check_post . . . 56
3.8.6 before_edit . . . 57
3.8.7 after_edit . . . 57
3.8.8 before_post . . . 58
3.8.9 after_message_save . . . 58
3.8.10 after_post . . . 59
3.9 Miscellaneous . . . 60
3.9.1 ajax_<call> . . . 60
3.9.2 database_error . . . 61
3.9.3 external . . . 61
3.9.4 scheduled . . . 62
3.10 Request initialization . . . 63
3.10.1 parse_request . . . 63
3.10.2 common_pre . . . 64
3.10.3 common_no_forum . . . 64
3.10.4 common_post_user . . . 65
3.10.5 common . . . 66
3.10.6 page_<phorum_page> . . . 66
3.11 Page output . . . 67
3.11.1 phorum_shutdown . . . 67
3.11.2 get_template_file . . . 67
3.11.3 start_output . . . 68
3.11.4 after_header . . . 69
3.11.5 before_footer . . . 69
3.11.6 end_output . . . 70
3.12 File storage . . . 70
3.12.1 after_detach . . . 70
3.12.2 before_attach . . . 71
3.12.3 after_attach . . . 72
3.12.4 system_max_upload . . . 72
3.12.5 file_retrieve . . . 73
3.12.6 file_purge_stale . . . 73
3.13 Admin interface . . . 74
3.13.1 admin_general . . . 74
3.13.2 admin_css_file . . . 74

3.14 Moderation . . . 75
3.14.1 email_user_start . . . 75
3.14.2 send_mail . . . 76
3.14.3 moderation . . . 76
3.14.4 before_delete . . . 77
3.14.5 delete . . . 78
3.14.6 move_thread . . . 78
3.14.7 close_thread . . . 79
3.14.8 reopen_thread . . . 79
3.14.9 after_approve . . . 80
3.14.10 hide_thread . . . 80
3.14.11 after_merge . . . 81
3.14.12 after_split . . . 81
3.15 Page data handling . . . 81
3.15.1 index . . . 81
3.16 User data handling . . . 82
3.16.1 user_save . . . 82
3.16.2 user_register . . . 83
3.16.3 user_get . . . 83
3.16.4 user_list . . . 84
3.16.5 user_delete . . . 85
3.16.6 before_register . . . 85
3.16.7 after_register . . . 86
3.17 User authentication and session handling . . . 87
3.17.1 user_authenticate . . . 87
3.17.2 user_session_create . . . 88
3.17.3 user_session_restore . . . 89
3.17.4 user_session_destroy . . . 91
3.18 Control center . . . 92
3.18.1 cc_panel . . . 92
3.18.2 cc_save_user . . . 92

List of Tables

2.1 Keys and values in module information 24

Introduction

This is the Phorum developer reference manual for Phorum version 5.2.x and up. It is not intended for use with older versions of Phorum, although a lot of information will apply.

Please keep in mind that this manual is neither complete, nor final. If you have any remarks about it, please let us know in the development forum on our website. With your contribution, we hope to make this manual a useful tool for Phorum users in understanding and working with our software.

The Phorum development team
Phorum.org

Chapter 1

Templates

1.1 Introduction

Phorum uses a template system for separating application code from presentation code. Application code contains all the logic that is needed for running Phorum. This is PHP code which is maintained by programmers. Presentation code is used to translate the data that is generated by the application code into a HTML page that can be viewed by the end user. This Presentation code can be maintained by HTML designers.

The big advantages of this type of system are that HTML designers will not be bothered with complicated PHP code and that it is easy to create multiple presentation styles for Phorum.

Although there is no application logic in the templates, it is still possible to put presentation logic in there. Presentation logic is only used for things like making decisions on what to show and how to show it and for processing data that has been generated by the application code. For writing presentation logic, a very simple custom programming language is available (more on that will follow when we talk about the Section 1.4).

1.2 Template structure

A template set is a collection of files that together form a single template. All template sets are stored in their own subdirectory under the directory `{phorumdir}/templates`. If we asume that we have three templates `default`, `template1` and `template2`, then the directory structure for storing these templates would look like this:

```
{phorum dir}
 |
 +-- templates
    |
    +-- default
```

```
|
+-- template1
|
+-- template2
```

Inside these template subdirectories, the files for the templates are stored. There, the the following files can be found:

info.php This is a PHP file that is used for describing some properties of the template. This file can define the following variables:

- `$name`
  Mandatory variable. This variable hold the name that you want to give to the template. This is the name that will be displayed in template selection boxes. The name of the directory for the template will only be used by Phorum internally.
- `$version`
  Mandatory variable. This variable holds the version number for your template. It's used so you can track what version of the template is installed for Phorum. You can use any type of version numbering you like. If you do not know what to use, then simply give your first version of the template version 1, the second number 2, and so on.
- `$template_hide`
  Optional variable. If set to a true value, the template will be hidden from user select boxes where the end user can choose the template that he wants to use.

Example 1.2.1 Template information file: $info.php

```
<?php
// Prevent loading from outside the Phorum application.
if (!defined("PHORUM")) return;

// Template information.
$name = "A brilliant template";
$version = "1.2-beta";
$template_hide = 1;
?>
```

.tpl and .php files These are the files that hold the actual template code. When the Phorum application wants to display a template, it is always referenced by its basename (i.e. without any file extension like `.php` or `.tpl` after it). If the file `<templatebasename>.php` exists in the template directory, then Phorum will use that file as the template input. Else, `<templatebasename>.tpl` will be used.

An example: if Phorum wants to display the "header" template, it will first search for `header.php` in the template directory. If that file does not exist, it will use `header.tpl` instead.

PHP files (`*.php`) contain pure PHP/HTML code. In Phorum template files (`.tpl`) you can additionally make use of the Section 1.4.

Using this system, template authors can completely revert to using pure PHP-code for templates, without using the template language at all. The Phorum development team does not recommend doing this. To keep templates simple, always try to stick to the combination of HTML code and the template language.

Other files and subdirectories In most cases these will be image files which are stored in a subdirectory `images` of the template. But template authors are free to add whatever subdirectories and files they like to the template directory (e.g. Flash based page components, CSS stylesheets, audio files, JavaScript libraries, etc.).

Combining all this, the full tree for a typical template would look like this:

```
{phorum dir}
 |
 +-- templates
     |
     +-- templatename
          |
          +-- info.php
          |
          +-- *.tpl
          |
          +-- images
               |
               +-- *.gif, *.jpg, *.png
```

1.3 How to start your own template

Although you can start writing a new template totally from scratch, it is of course much easier to take an existing template and modify that one for your needs. Here are the steps that you have to take for accomplishing this:

- **Copy the default template**
  Take the default template directory from `{phorumdir}/templates/default` and copy it over to another directory, for example `{phorumdir}/templates/mytpl`.
- **Edit info.php for your template**
  Edit `{phorumdir}/templates/mytpl/info.php`. In this file you have

to edit at least the `$name` variable, e.g. to `$name = "My very own template";`

You can hide the template from the user template selection boxes by setting `$template_hide = 1`. If you do this, you can only select this template through the admin interface.

- **Configure Phorum to use your template**
  Open Phorum's admin page `{phorumurl}/admin.php` and go to "Default Settings". There you will find the "Template" option. Set that option to your own template. All forums that inherit their settings from the default settings will use the template automatically. For other folders and forums, you will have to go to their settings pages to set their template to the default template as well.

That is it! You are now using your own template. From here on, you can start tweaking the template files in your `{phorumdir}/templates/mytpl` directory.

Phorum uses its own template language to allow for dynamic templates without using PHP. More information on this can be found in the section about the Phorum template language.

1.4 The Phorum template language

1.4.1 Introduction

The largest part of the code that can be found in Phorum template files (`*.tpl`) is plain HTML. To be able to use and display the dynamic data that has been generated by Phorum (like message information, lists of private messages and search results), Phorum uses a custom template language which can be used to mix the HTML code with dynamic data. The template language is a very simple programming language with only a few statements to use. This section will describe the template language in detail.

1.4.2 General syntax

Templates are built using HTML code. Embedded in this HTML code, there can be template language statements. All template statements in the templates are surrounded by "{" and "}" characters. Here's a simple example of what a template could look like:

Example 1.4.1 Template example

```
<html>
 <head>
<title>{HTML_TITLE}</title>
 </head>
 <body>
Your username is: {USER->username}

{IF USER->username "george"}
  <b>Hello, George!</b>
{/IF}
 </body>
</html>
```

Because curly braces have a special meaning in the templates, you have to take care when using them for other things than Phorum template code. This applies to plain PHP code, JavaScript code and CSS code that you use in your templates. To prevent the template engine from getting confused, you can add a space after "{" and before "}". Examples:

```
Code that will cause problems if used in a template file:

PHP: if ($a == $b) {print "They are the same!\n";}
JavaScript: if (a == b) {alert("They are the same!\n");}
CSS: #phorum .thing {font-size: 110%;}

What it should be written like:

PHP: if ($a == $b) { print "They are the same!\n"; }
JavaScript: if (a == b) { alert("They are the same!\n"); }
CSS: #phorum .thing { font-size: 110%; }<sbr/>
```

1.4.3 Data types

The template language supports four data types to use in statements:

- Integers
- Strings
- PHP Constants
- Template variables

1.4.3.1 Integers

Integers are formatted as a sequence of numbers.

Example 1.4.2 Integer values

```
403
90
4231
```

Here is an example of template code in which integers are used:

Example 1.4.3 Code using integer values

```
{VAR INTEGERVAR 1000}
The variable INTEGERVAR is {INTEGERVAR}.

{IF INTEGERVAR 333}
  The INTEGERVAR has the value 333.
{/IF}
```

1.4.3.2 Strings

Strings are sequences of characters within quotes (both double and single quotes can be used).

Example 1.4.4 String values

```
"this is a string value"
"My 1st string!"
'Single quoted string is possible too'
```

Now if you need the quote which you used to surround the string with inside the string itself, you must escape it using \" or \'. This is consistent with the way that PHP strings are escaped.

Example 1.4.5 Escaped quotes in string values

```
"this is a \"string\" value"
'Single quoted \'string\' value'
"You can use both \" and ' for strings!"
```

Here are some examples of template code in which strings are used:

Example 1.4.6 Code using string values

```
{VAR QUESTION "Do you know what \"fubar\" means?"}
{VAR CORRECT "That was the right answer!"}
{VAR INCORRECT "No.. you were wrong!"}

{IF ANSWER 'Fucked Up Beyond All Recognition'}
  {CORRECT}
{ELSE}
  {INCORRECT}
{/IF}
```

1.4.3.3 PHP constants

It is possible to define constants within PHP. This is done using the define() PHP statement. Here's an example:

```
<?php define("MY_CONSTANT", "The constant value") ?>
```

You can reference PHP constants from the template language by using its name, without any quotes. So the constant that was defined in the code above, can be used like this in a template:

Example 1.4.7 Code using a PHP constant definition

```
The value of my PHP constant is {MY_CONSTANT}
```

Apart from defining your own PHP constants, you can also use constants that are already defined by PHP. Two useful constants to use are `true` (value = 1) and `false` (value = 0). Using these, you can write template code like this:

Example 1.4.8 Code using built-in PHP constants

```
{VAR SOME_OPTION true}

{IF SOME_OPTION true}
  The option SOME_OPTION is true.
{/IF}
```

1.4.3.4 Template variables

About the most important data type for the template language is the template variable. Template variables are used by Phorum to store dynamic data, which can be used by your templates. You can also use the variables for storing dynamic data of your own from the templates. Template variables can contain both simple values and complex arrays of data.

You can reference a template variable by using the variable's name, without any quotes. This is the same type of notation as the one that is used for referencing PHP constants (see Section 1.4.3.3). If there are both a constant and a variable with the same name, the value of the contstant will take precedence over the template variable.

Example 1.4.9 Template variables

```
NAME
HTML_TITLE
MESSAGES
```

In case the variable represents an array, you can reference the array elements by using the following pointer notation:

Example 1.4.10 Referencing elements in a template variable array

```
ARRAYVARIABLE->SIMPLE_ELEMENT
ARRAYVARIABLE->ARRAY_ELEMENT->SIMPLE_ELEMENT
```

Within a template, variables are used like this:

Example 1.4.11 Code using template variables

```
{VAR MY_VAR "Assign a value to a variable from the template"}

You username is: {USER->username}<br/>
The current forum's name is: {NAME}<br/>

{LOOP MESSAGES}
  Subject: {MESSAGES->subject}<br/>
{/LOOP MESSAGES}
```

What variables are available for what template pages is fully determined by Phorum.

1.4.4 Statements

The template language has a number of statements that can be used for executing templating actions and decisions.

- Display a variable
- In line comments
- DEFINE

- VAR
- IF .. ELSEIF .. ELSE ..
- LOOP
- INCLUDE
- HOOK

1.4.4.1 Display a variable

Function This is both the most simple and the most important template statement there is. Using this statement, you can display the contents of a value.

Syntax `{<VALUE>}`

Example 1.4.12 Display a variable

```
The name of the current forum is: {NAME}
```

Example code

1.4.4.2 In line comments

Function Sometimes, it's useful to explain what you are doing when writing complicated templating code. In that case you can use comments to document what you are doing. You can also use comments to add general info to the template (like in the example below).

Syntax `{! <COMMENT TEXT>}`

The `<COMMENT TEXT>` can contain any characters you like, except for "}".

Example 1.4.13 Add in line comments

```
{! This template was created by John Doe and his lovely wife ↩
    Jane }
```

Example code

1.4.4.3 DEFINE

Function Using this statement, you can set definitions that can be used by the Phorum software. These are mainly used for doing settings from the template file "settings.tpl" to tweak Phorum's internal behaviour.

Definitions that have been set using this statement are not available from other template statements.

Syntax `{DEFINE <PHORUM DEFINITION> <VALUE>}`

What you can use for `<PHORUM DEFINITION>` is fully determined by the Phorum software (and possibly modules). The `<VALUE>` can be any of the data types that are supported by the template language (see Section 1.4.3).

Example 1.4.14 DEFINE statement usage

```
{DEFINE list_pages_shown 5}
```

Example code

1.4.4.4 VAR

Function Using this statement, you can set variable definitions that can be used by the Phorum template language.

Syntax `{VAR <TEMPLATE VARIABLE> <VALUE>}`

`<TEMPLATE VARIABLE>` can be an existing or a new variable name (see Section 1.4.3.4). The `<VALUE>` can be any of the data types that are supported by the template language (see Section 1.4.3).

Example 1.4.15 VAR statement usage

```
{VAR MY_VAR "This is my first variable!"}
{VAR MY_VAR OTHER_VAR}
{VAR MY_VAR 1234}

{VAR IS_COOL true}
{IF IS_COOL}
  Yes, this is cool
{/IF}
```

Example code

1.4.4.5 IF .. ELSEIF .. ELSE ..

Function Using these statements, you can control if certain blocks of code in your template are processed or not, based on a given <CONDITION>. This can for example be useful if you want certain parts of the page to be only visible for registered users.

Syntax
```
{IF <CONDITION>}
  .. conditional code ..
[{ELSEIF <CONDITION>}
  .. conditional code ..]
[{ELSE}
  .. conditional code ..]
{/IF}

<CONDITION> Syntax: [NOT] <TEMPLATE VARIABLE> [<VALUE>]
```

The <TEMPLATE VARIABLE> in a <CONDITION> has to be an existing variable name. The <VALUE> can be any of the data types that are supported by the template language (see Section 1.4.3).

If a <VALUE> is used, the <TEMPLATE VARIABLE> will be compared to the <VALUE>. If the <VALUE> is omitted, then the condition will check whether the <TEMPLATE VARIABLE> is set and not empty.

A condition can be negated by prepending the keyword NOT to it.

Multiple conditions can be chained using the keywords AND or OR.

Example 1.4.16 IF .. ELSEIF .. ELSE .. statement usage

```
{IF NOT LOGGEDIN}
   You are currently not logged in.
{ELSEIF USER->username "John"}
   Hey, it's good to see you again, mr. John!
{ELSE}
   Welcome, {USER->username}!
{/IF}

{IF ADMINISTRATOR true OR USER->username "John"}
   You are either an administrator or John.
{/IF}

{IF VARIABLE1 VARIABLE2}
   Variable 1 and 2 have the same value.
{/IF}
```

Example code

1.4.4.6 LOOP

Function The LOOP statement is used for looping through the elements of array based template variables (for example arrays of forums, messages and users).

Syntax
```
{LOOP <ARRAY VARIABLE>}
  {<ARRAY VARIABLE>}
{/LOOP <ARRAY VARIABLE>}
```

The `<ARRAY VARIABLE>` has to be the name of an existing template variable containing an array.

Within the LOOP, the active array element is assigned to a variable that has the same name as the `<ARRAY VARIABLE>` that you are looping over. In our example below, we are looping over `USERS`, which is an array of user data records. Within the loop, `USERS` is no long the array of users itself, but the user data record for a single user instead.

Example 1.4.17 LOOP statement usage

```
<ul>
{LOOP USERS}
  <li>{USERS->username}</li>
{/LOOP USERS}
</ul>
```

Example code

1.4.4.7 INCLUDE

Function Include another template in the template.

Syntax `{INCLUDE [ONCE] <INCLUDE PAGE>}`

The `<INCLUDE PAGE>` can be any of the data types that are supported by the template language (see Section 1.4.3).

By specifiying the keyword `ONCE` before the name of template to include, you can make sure that that template is only included once per page.

Example 1.4.18 INCLUDE statement usage

```
{INCLUDE "paging"}

{VAR include_page "cool_include_page"}
{INCLUDE include_page}

{INCLUDE ONCE "css"}
```

Example code

Limitiation It is not possible to use a dynamic INCLUDE statement (one where the <- INCLUDE PAGE> is set through a template variable) within a LOOP statement, in case the included template needs to have access to the active LOOP element. There is no problem if you use a static INCLUDE statement (one where the <INCLUDE PAGE> is set through a string value).

If you really need this kind of functionality though, you can work around this limitation by assigning the active LOOP element to a new template variable, prior to including the dynamic <INCLUDE PAGE>. Example:

```
{! include_page holds the dynamic page to include }
{VAR include_page "some_page"}

{LOOP loop_variable}
  {! Makes loop_variable available as temp_variable in  ↩
     the include }
  {VAR temp_variable loop_variable}
  {INCLUDE include_page}
{/LOOP loop_variable}
```

This way you can access the active LOOP element from the included template through temp_variable. If you would access loop_variable from there, you'd see that it does not contain the active LOOP element, but the full array that you are looping over instead.

1.4.4.8 HOOK

Function The HOOK statement can be used to run a module hook from a template. By using hooks in the templates, you have an easy way for modules to add data to a page, without having to change the templates too much. Because these hooks need an activated module that acts upon them, creating HOOK statements is certainly for advanced users only.

Syntax {HOOK <HOOK NAME> [<ARG1> <ARG2> .. <ARGn>]}

Both the <HOOK NAME> and the arguments that are used in the HOOK statement can be any of the data types that are supported by the template language (see Section 1.4.3).

How hook functions are called Depending on the number or arguments that are used in the HOOK statement, different type of calls are made to the hook function for the given <HOOK NAME>.

- *No arguments:*
  the hook function is called without any arguments at all:
  hook_function()

- *One argument:*
  The single argument is used directly for calling the hook function:
  `hook_function($ARG1)`
- *Multiple arguments:*
  The arguments are wrapped in an array, which is then used for calling the hook function:
  `hook_function(array($ARG1,$ARG2,..$ARGn))`

Example 1.4.19 HOOK statement usage

```
{HOOK "template_hook"}

{LOOP MESSAGES}
  {HOOK "show_message" MESSAGES}
{/LOOP MESSAGES}

{VAR HOOKNAME "my_magic_hook"}
{HOOK HOOKNAME "my argument"}
```

Example code

1.4.5 Need the power of PHP?

Template writers for whom the template language is too limited can break into PHP at any point in the templates, using the regular `<?php ... ?>` syntax. It is not mandatory at all to use the Phorum template language for your templates.

The biggest drawback here, is that knowledge of the Phorum internals is required if you want to work with the data that has been generated by Phorum.

Most template writers will normally only be using HTML and the Phorum template language.

To prevent confusion between PHP code blocks and template statements (which are both surrounded by "{" and "}" characters), always use a whitespace after an opening "{" character in your PHP code. So instead of writing:

```
<?php if ($this = true) {print "It's true";} ?>
```

you now have to write:

```
<?php if ($this = true) { print "It's true"; } ?>
```

This way you can mix PHP code with template code without running into problems.

Chapter 2

Modules

2.1 Introduction

This section describes Phorum's module system. It is targeted at developers who want to do customization and extend the functionality of Phorum. Modules are the preferred way to archieve this.

For much of this document, we will be talking about an example module "foo". Of course you will not name your module "foo", but something much more appropriate. If you are not familiar with the terms "foo" and "bar", you can visit Wikipedia to see why we chose them.



Be sure to read at least the CAUTIONS AND SECURITY ISSUES section, before making your own modules.

2.2 Terminology

2.2.1 Modules

Modules are self contained pieces of software, that can be added to Phorum to change or extend its functionality. Modules can do this without having to change anything in the standard Phorum distribution files or database structure.

The big advantage of modules this is that upgrading the Phorum code is easy (no file changes to redo after upgrading) and that modules can be easily uninstalled when needed.

Installing a module means: drop the code in the Phorum mods directory, go to the admin "Modules" page, enable the module and enjoy! One additional thing that might be needed, is editing one or more template files to display data that is generated by the module.

2.2.2 Hacks

The moment it is necessary to make changes to the standard Phorum distribution files or database structure to implement some kind of functionality, we are talking about a hack (even if the changes that have to be made are accompanied by a drop in module).

Although there is nothing wrong with writing hacks, the Phorum team wants to urge you to try if you can write a module before resorting to a hack. Especially if you are going to publish your changes to the public. Modules are the preferred way of modifying Phorum functionality, because that will make both upgrading your distribution and having your modification adopted by others easier.

2.2.3 Add-ons

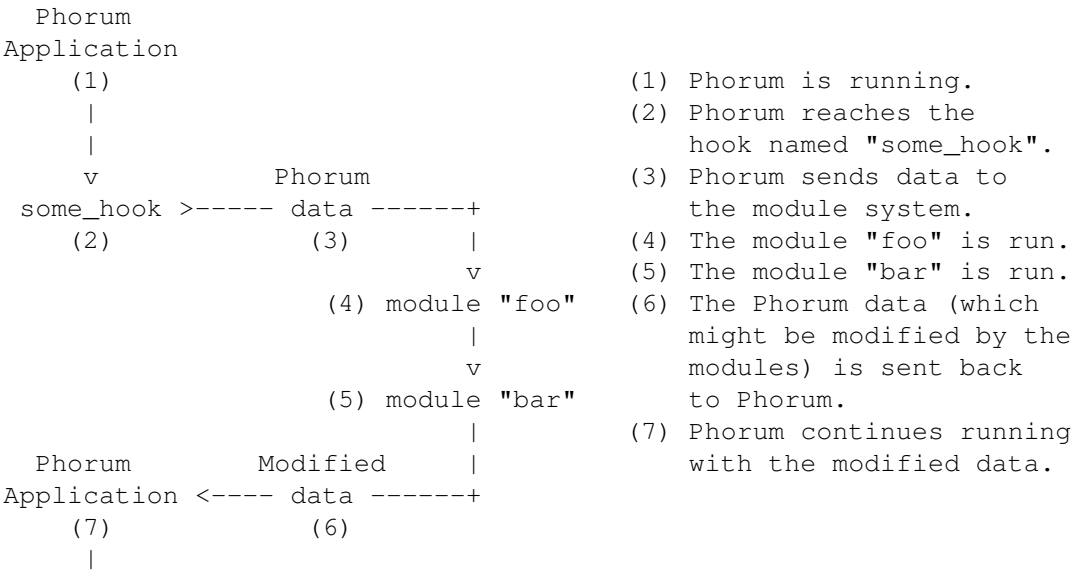
If you add functionality to Phorum by providing extra scripts that go in the Phorum install directory and/or extra templates that need to be added to the main template directory (`templates/templatename/...`), then we talk about an add-on.

For Phorum 5.0, this was a pretty common thing to do. For Phorum 5.1 and up, a special script was added for being able to implement addon code fully through a module. Therefore, there is no real need anymore for writing add-ons: `addon.php`.

2.2.4 Hooks

The Phorum core and Phorum modules are interconnected through hooks. Hooks are points in the application where Phorum stops and runs its data through the modules that are configured to handle the hook. The modules can act upon and change this data.

The following image visualizes what happens when Phorum reaches a hook point in the application, for which two modules ("foo" and "bar") have been configured.



```
    |
    v
```

2.2.5 Hook functions

A module contains PHP functions that act as hook functions. Hook functions will receive some data from Phorum through their arguments and have to return the (possibly modified) data, which will then go either back to Phorum or to the input of another module which also handles the same hook (see [??]). Based on this, the most basic (and useless) hook function you could write would look somewhat like this (see XXX for an explanation of the naming scheme that was used for the function):

```
function phorum_mod_foo_some_hook ($data) {
  return $data;
}
```

The exact nature of the data that is sent to the hook functions depends solely on the hook that is run. See Chapter 3 for a description of all supported hooks, including a specification of the type of data that is sent.

2.3 Writing your own modules

2.3.1 Introduction

This section will explain to you how to roll your own Phorum modules. We will start out by explaining some of the terminology that relates to modules. After that, we will explain a very important part modules: the module information. This contains information for both Phorum (what hooks to run in what order, version dependancies) and module users (title, description and other interesting facts). From there on we will walk you through all the possibilities that modules have.

2.3.2 Module information

Module information is the glue between your module and Phorum. It provides information to Phorum about your module. Before we explain how to add this module information to your module, we will first explain what data can be put in there and how that data is formatted.

Module information is formatted using lines of plain text. Each line contains a piece of information about the module. The general format for each of the lines in the module information is:

```
<key>: <value>
```

Empty lines are allowed between these key/value pairs. Below, you can find a list of the keys and values that can be used in the module information.

It is allowed to use multiple hook lines in your module information, so your module can act upon multiple hooks. When doing this, it is also allowed to use the same

<key>	<value>
title	This is the title for the module that is displayed in the "Modules" page of the admin interface. Example: `title: Foo`
desc	This is the description that is displayed along with the title in the admin interface, to give a little more information about the module. Using HTML in the <value> part is allowed. Example: `desc: This is a very cool module to do stuff.`
hook	This describes which hook functions are called for which Phorum hooks. The value consists of two fields, separated by a pipe "|" symbol. The first field contains the name of the hook that this module is hooking into. The second field contains the name of the hook function that will be called for the hook. Example: `hook: some_hook|phorum_mod_foo_some_hook`

priority	This can be used for changing priorities and dependancies for modules and hooks. Possible values are (in order in which they are processed):
	• `run module before|after *`
	• `run module before|after <other module name>`
	• `run hook <hook name> before|after *`
	• `run hook <hook name> before|after <other module name>`
	Examples:
	Run this module before all other modules:
	`priority: run module before *`
	Run this module before the bbcode module.
	`priority: run module before bbcode`
	Run the "format" hook for this module

hook function for handling different hooks in your module (asuming the hooks are compatible).

Here is an example of what the module information for our example module "foo" might look like:

Example 2.3.1 Module information

```
title: Foo
desc: This is the Foo module for Phorum. Nothing exciting...
version: 1.0.2
release_date: Jan 1st, 2008
url: http://www.phorum.org
author: John Doe <johndoe@example.com>
require_version: 5.2.2
category: user_features

hook: some_hook|phorum_mod_foo_some_hook
hook: some_other_hook|phorum_mod_foo_some_other_hook
hook: yet_another_hook|phorum_mod_foo_some_other_hook

priority: run some_hook before some_other_module
```

What this module info does, is telling Phorum that when it gets to "some_other_hook", it will have to call the function `phorum_mod_foo_some_other_hook()` in your module. It also tells that for "yet_another_hook" the same function has to be called. It will also take care that the hook "some_hook" is run before the same hook in the module "some_other_module".

2.3.3 Module file structure

2.3.3.1 Introduction

This section describes the file structure of Phorum modules. This structure contains things like the module information, hook functions and possibly additional stuff like templates, translations, modules settings, images, scripts, classes, etc.

If your module only needs module information and hook functions to function, it is possible to use the single file structure. If you need more than that, then use the multiple file structure.

2.3.3.2 Single file modules

Single file modules are useful in case case no additional files have to be distributed with your module. Because the module consists of only one single file, it is very easy to distribute. Beware though that the moment that you want to support for example a settings screen, multiple languages or custom images, you will have to switch to the multiple file module structure. Switching does mean some extra work for your

users. So only use this format for modules for which you are sure that you do not need additional files in the future.

Single file modules consist of one single PHP file. The name of this file is not restricted. We advice you to use `mod_<modulename>.php` though for clarity and consitency with other module (e.g. `mod_foo.php`). This file contains both the module information and the hook function definitions. For storing the module informaton, a special PHP comment is used. This comment must look like the following:

```
/* phorum module info
<module information lines go here>
*/
```

Using the example module info from Example 2.3.1, the complete single file module would look like this (see XXX why we use the check on PHORUM at the start of this file):

Example 2.3.2 Single file module

```
{phorum dir}/mods/mod_foo.php
<?php

if(!defined("PHORUM")) return;

/* phorum module info
title: Foo
desc: This is the Foo module for Phorum. Nothing exciting...
version: 1.0.2
release_date: Jan 1st, 2008
url: http://www.phorum.org
author: John Doe <johndoe@example.com>
require_version: 5.2.2
category: user_features

hook: some_hook|phorum_mod_foo_some_hook
hook: some_other_hook|phorum_mod_foo_some_other_hook
hook: yet_another_hook|phorum_mod_foo_some_other_hook

priority: run some_hook before some_other_module
*/

function phorum_mod_foo_some_hook ($data) {
    // Do stuff for "some_hook".
    return $data;
}

function phorum_mod_foo_some_other_hook ($data) {
    // Do stuff for "some_other_hook" and "yet_another_hook".
    return $data;
}

?>
```

Installation of a single file module is done by putting the PHP file (e.g. `mod_foo.php`) directly in the directory `{phorumdir}/mods/` and activating the module from the "Modules" screen in your admin interface.

2.3.3.3 Multiple file modules

These modules are useful in case you need additional files to be stored with your module, for example a settings screen, language files or custom images.

They are stored in their own subdirectory below the directory `{phorumdir}/mods/`. If you have a module named "foo", you will have to create a directory `{phorumdir}/mods/foo/` for storing all module files.

Inside this subdirectory, you will have to create a least two files:

- A file called `info.txt`. This file contains the module information for your module (see Section 2.3.2).
- The PHP file which contains the hook function definitions for your module. The basename of this file should be the same as the name of the module subdirectory. So for our example module "foo", you will have to create a file named `foo.php`.

Using the example module info from Example 2.3.1, the complete multiple file module would look like this (see XXX why we use the check on PHORUM at the start of the PHP file):

Example 2.3.3 Multi file module

```
{phorum dir}/mods/foo/info.txt
title: Foo
desc: This is the Foo module for Phorum. Nothing exciting...
version: 1.0.2
release_date: Jan 1st, 2008
url: http://www.phorum.org
author: John Doe <johndoe@example.com>
require_version: 5.2.2
category: user_features

hook: some_hook|phorum_mod_foo_some_hook
hook: some_other_hook|phorum_mod_foo_some_other_hook
hook: yet_another_hook|phorum_mod_foo_some_other_hook

priority: run some_hook before some_other_module
```

```
{phorum dir}/mods/foo/foo.php
<?php

if(!defined("PHORUM")) return;

function phorum_mod_foo_some_hook ($data) {
    // Do stuff for "some_hook".
    return $data;
}

function phorum_mod_foo_some_other_hook ($data) {
    // Do stuff for "some_other_hook" and "yet_another_hook".
    return $data;
}

?>
```

So far, the module has exactly same functionality as the single file module from Section 2.3.3.2. From here on, the functionality can be extended. Some of the possibilities are:

- Adding custom files to your module tree (images, classes, libs, etc.);
- Letting your module support multiple languages;
- Implementing a settings screen for your module;
- Adding template files for your module; (See XXX about module template files)

2.3.4 Supporting multiple languages

This feature is supported by the multiple file structure.

If your module includes text that will be displayed to end users, you should strongly consider making it support multiple languages. This will allow Phorum installations that use a different language(s) to display output of your module in the same language(s), instead of the language you have written the module in.

For supporting multiple languages, the first thing to do is add the following line to your module information file `info.txt`:

```
hook: lang|
```

There is no actual hook function configured here, because the "lang" hook is only used as a marker for Phorum. It tells Phorum that your module supports multiple languages.

Next, you must provide at least one language file with your module. Language files are stored in a subdirectory name "lang" inside your module directory. In our sample module, the full directory would be `{phorumdir}/mods/foo/lang/`. The language files must be named identical to the main language files that Phorum uses. To include both English and French, your module would require the following file structure:

Example 2.3.4 Tree structure for a module that supports languages

```
{phorum dir}/
     |
     +-- mods/
          |
          +-- foo/
               |
               +-- info.txt
               |
               +-- foo.php
               |
               +-- lang/
                    |
                    +-- english.php
                    |
                    +-- french.php
```

The structure of your language files will be almost identical to that of the main Phorum language files. However, for your own language files it is advisable to add an

extra level in the language variables, to avoid conflicts with language string from other modules or Phorum itself. Here is an example of how you could do that:

Example 2.3.5 Custom language file for a module

```
<?php
$PHORUM["DATA"]["LANG"]["mod_foo"] = array(
    "Hello"   => "Hello!",
    "Bye"     => "Good bye!"
);
?>
```

Here, the extra inserted level is `["mod_foo"]`. To access the "Hello" string from your module code you would use the PHP variable:

```
$PHORUM["DATA"]["LANG"]["mod_foo"]["Hello"]
```

When you want to use the language string from a template file, the you would use the following template variable:

```
{LANG->mod_foo->Hello}
```

In case a Phorum installation is using a language that your module does not support, Phorum will automatically attempt to fallback to English. So it is highly recommend that you include an `english.php` language file in all your modules. If both the current language and English are not found, Phorum will be unable to load a language for your module and will display empty space instead of your language strings.

Always try to reuse strings that are already in the main Phorum language files itself. Only create custom strings when there is no alternative available. Having more text to translate is more work for translators and using core language strings helps in keeping the used terminology consistent.

2.3.5 Module data storage

2.3.5.1 Introduction

Sometimes, modules will have to store some data in the Phorum system. For example an avatar module would have to store what avatar a user want to show and a poll module would have to add the question, answers and voting results for a poll to messages in which a poll is added.

This section description what standard methods are available for letting modules store their data in the Phorum system. Of course, as a module writer, you can divert from this and use any kind of storage that you like. You are in no way limited to only use Phorum specific methods here.

2.3.5.2 Storing data for messages

If your module needs to store data along with a Phorum message, the easiest way is to make use of the meta information array that is attached to each message array (`$me-`

ssage["meta"]). This array is a regular PHP array, which is stored in the database as serialized data (see PHP's serialize manual). Because Phorum and other modules make use of this meta data as well, you should never squash it, neither access the meta data in the database directly. Instead use the methods described in this section.

To prevent name space collissions with other modules or Phorum, it is good practice to create only one key in the meta data array named mod_<yourmodule> (in our example: mod_foo). If your module needs to store only one single value, then put it directly under this key:

```
$message["meta"]["mod_foo"] = "the single value";
```

If multiple values need to be stored, then put an array under the key. This array can be as complicated as you like:

```
$message["meta"]["mod_foo"] = array(
    "key1"    => "value1",
    "key2"    => "value2",
    "complex" => array(
        0 => "what",
        1 => "a",
        2 => "cool",
        3 => "module"
    )
);
```

because the meta data is stored as serialized data in the database, it is not possible to handle the data you store in there through SQL queries.

When storing information in the meta data from a hook function, you can encounter two different situations, which both need a different way of handling: hooks that get an editable message array as their argument and hooks that don't.



2.3.5.2.1 From hooks that get an editable message array as their argument

There are some hooks that send a full message structure to the hook functions. These can change the message structure before returning it to Phorum. Examples are the hooks "hook.before_post" and "before_edit". For these kind of hooks, you can update the meta information in the message structure and be done with it. Here's an example of what this could look like in your hook function:

```
function phorum_mod_foo_before_post ($message)
{
    // Make sure that we have an array for mod_foo in the  ↩
        meta data.
    if (!isset($message["meta"]["mod_foo"]) ||
        !is_array($message["meta"]["mod_foo"])) {
        $message["meta"]["mod_foo"]["foodata"] = array();
    }

    // Add some fields to the mod_foo data.
    $message["meta"]["mod_foo"]["foodata"] = "Some data";
```

```
    $message["meta"]["mod_foo"]["bardata"] = "Some more data ↩
        ";

    // Return the updated message. Phorum will take care of
    // storing the "mod_foo" array in the database.
    return $message;
}
```

2.3.5.2.2 From other hooks For other hooks, the proper way to store information in the meta data is to first retrieve the current message data (including the current meta data) using the `phorum_db_get_message()` function. After this, merge the information for your module with the existing meta data and store the updated data in the database using the `phorum_db_update_message()` function. Here is an example of what this could look like in your hook function:

```
function phorum_mod_foo_some_hook ($data)
{
    // Somehow you get the id for the message. Here we asume
    // that it is stored in the $data hook parameter.
    $message_id = $data["message_id"];

    // Retrieve the message from the database.
    $message = phorum_db_get_message ($message_id);

    // Extract the current meta data.
    $meta = $message['meta'];

    // Make sure that we have an array for mod_foo in the  ↩
        meta data.
    if (!isset($meta["mod_foo"]) || !is_array($meta["mod_foo ↩
        "])) {
        $meta["mod_foo"]["foodata"] = array();
    }

    // Add some fields to the mod_foo data.
    $meta["mod_foo"]["foodata"] = "Some data";
    $meta["mod_foo"]["bardata"] = "Some more data";

    // Store the updated meta data in the database.
    phorum_db_update_message($message_id, array("meta" =>  ↩
        $meta));

    // Return the data that we got as input for this hook  ↩
        function.
    return $data;
}
```

Changing meta data for a message this way will ensure that the existing meta data is kept intact.

2.3.5.3 Storing data for users

2.3.5.3.1 Custom profile fields for users If your module needs to store data along with a Phorum user, you can make use of custom profile fields. These fields will be accessible from within the user data. E.g. if you create a custom profile field named "foobar", the value of that field will be stored in $user["foobar"] (so right next to the standard fields like $user["username"] and $user["email"]).

Creating custom profile fields can be done from the admin interface, under "Custom Profiles". It is also possible to let your module create the custom profile field fully automatical, by using the Custom Profile Fields API. If you choose to let the user of your module create the field by hand, then please include a thorough description of what configuration the user has to do. A lot of problems with modules that require manual configuration come from using wrong options for a custom profile field.

Using a separate field for each piece of data

When using a custom profile field for storing module information, you can use a separate field for each piece of data you want to store. The advantage of doing this, is that you can then use the option "Disable HTML" for the fields that you will be sending to the user's browser. In fields with this option enabled, characters that have a special meaning in HTML will be escaped after loading the user from the database. This prevents the field from being vulnerable to XSS attacks. Recommended settings for storing a single value in a profile field are:

- **Field Name:**
  Name your field `mod_<module name>` if you only need to store one single value. If you need to store more values, then use the format `mod_<module name>_<field name>`. This prevents the risk of clashing with standard Phorum user fields or custom fields that are added for other modules. For example, the "foo" module could use the field names `mod_foo_size` and `mod_foo_name`.

- **Field Length:**
  If you want some field to contain a predefined maximum number of characters, then fill in that number of characters in this field. Before storing the field data to the database, Phorum will trim the data down if it is longer than the defined number of characters. If you need no limit, you can also use `65000` here.

- **Disable HTML:**
  Enable this option, unless you are absolutely sure that the data for this field is either not shown in the browser or escaped by your module before showing it.

- **Show in user admin:**
  If you want the field to be visible along with the user data for a user in the admin interface, then enable this option.

Using a single field for storing a complex data structure

Instead, you can also create a single field for storing a complete array of information. Phorum will automatically take care of storing this information (serialized) in the database. You only should make sure that the custom profile field is large enough to

store all the data and that HTML is allowed for the field, so the special PHP serialization code will not be broken by escaping special characters. When your module needs to store multiple fields, this is the preferred way. Recommended settings for storing a full array in a profile field are:

- **Field Name:**
  Name your field `mod_<module name>`, so you will not risk clashes with standard Phorum user fields or custom fields that are added for other modules. For example, the "foo" module would use the field name `mod_foo`.
- **Field Length:**
  Use `65000` here. Using smaller values will not make the database storage smaller. This value is only used to trim down the data to the provided length. So for storing serialized data, it is best to set this value as high as possible.
- **Disable HTML:**
  Disable this option, so the serialize data will not be broken by escaping special characters.
- **Show in user admin:**
  Disable this option. There is currently no support for showing serialized fields in the user admin pages in a readable way.

2.3.5.3.2 From hooks that get an editable user array as their argument There are some hooks that send a full message structure to the hook functions. These can change the message structure before returning it to Phorum. An example is the hook "[??]". For these kind of hooks, you can update the custom profile field data in the user structure and be done with it. Here's an example of what this could look like in your hook function:

```
function phorum_mod_foo_user_save ($user)
{
    // Some data to store in the "mod_foo" custom field.
    $data = array(
        "user_id" => $user_id,
        "mod_foo" => array (
            "foodata" => "Some user data",
            "bardata" => "Some more user data"
        )
    );

    // Put the data in the user sructure.
    $user["mod_foo"] = $data;

    // Return the updated user. Phorum will take care of
    // storing the "mod_foo" array in the database.
    return $user;
}
```

2.3.5.3.3 From other hooks For storing data in the custom profile field, you can make use of the `phorum_api_user_save()` function. This function needs the user_id of the user and all fields that need to be updated. Below are two pieces of code which show how our example module might store data for a user (asuming $user_id is the id of the user that must be changed).

Example 2.3.6 Filling custom profile fields for a user with data

```
// When using multiple fields "mod_foo_foodata" and " ↩
    mod_foo_bardata".

$userdata = array(
    "user_id"         => $user_id,
    "mod_foo_foodata" => "Some user data",
    "mod_foo_bardata" => "Some more user data"
);
phorum_api_user_save($userdata);

// When using a single custom field "mod_foo" for this  ↩
    module:

$userdata = array(
    "user_id" => $user_id,
    "mod_foo" => array (
        "foodata" => "Some user data",
        "bardata" => "Some more user data"
    )
);
phorum_api_user_save($userdata);
```

2.3.6 Building URLs for Phorum

2.3.6.1 Introduction

You might have noticed that all URLs that are used by Phorum are full absolute URLs. Phorum does not use relative URLs anywhere. and all of the URLs that point to Phorum PHP scripts are generated by the function `phorum_get_url()`. This was done for several reasons. Here are some of them, for the curious developer:

- By generating full URLs, we guarantee ourselves that the user is always opening pages within the same domain. With relative URLs, the user might end up at a different domain because of some webserver redirect (e.g. from `http://-example.com` to `http://www.example.com`), causing possible loss of cookies as a result (since cookies bind to domains). Loss of cookies result in the user being logged out.
- Integrating Phorum in a website is a lot easier when using absolute URLs. When editing the header template for example, it should be okay to add a `<base hr-`

ef="..."/> in there, pointing at the URL where the original site exists. After doing so, the header template can use the same constructions and paths as the main site's header template.

- This becomes even more important when running Phorum in portable or embedded setups. There, Phorum will be run from some script at a random location and no longer from a script in the Phorum directory. By using absolute URLs, the linked resources can still be found.

- Code for generating absolute URLs is needed for generating URLs that can be put in mail messages. It's a logical choice to use the same code for generating the other URLs in Phorum, so only one URL generating function has to be maintained.

- By letting `phorum_get_url()` generate all URLs, we are prepared for future changes and new features. If changes are needed in the URL schema, we only have to update this function and nothing further. All core code and modules that use this function will automatically follow the changes.

2.3.6.2 Build URLs for Phorum PHP scripts: phorum_get_url()

Phorum uses the function `phorum_get_url()` to consistenly build URLs that point to Phorum PHP scripts. It is recommended that you use this function as well when creating URLs to scripts yourself, so special features and future changes will automatically be incorporated in the links you use.

Here is an example of building an URL, which will open the profile page for the user with user_id = 17:

Example 2.3.7 Generating a profile URL using phorum_get_url()

```
$url = phorum_get_url(PHORUM_PROFILE_URL, 17);
```

The argument list that this function takes, depends on the first argument which tells Phorum what URL type has to be built. When building other URLs, other arguments will be used.

About all URL types that `phorum_get_url()` supports are used for building URLs that point to the scripts that are bundled with Phorum. Sometimes, you might want to add an extra script of your own to the Phorum tree (see Section 2.2.3). For those, you can use `phorum_get_url()` as well. The way to go is simple. You need to use `PHORUM_CUSTOM_URL` as the first argument and add the following parameters to it:

- The first parameter needs to be the filename of the file to link to, without the (.php) extension.

- The second parameter needs to be FALSE or TRUE. If it is TRUE, then the current forum_id is added to the URL.

- All other parameters are added directly to the URL.

Here is an example of building a URL which links to the add-on file `myfile.php` in the Phorum installation directory. Lets asume that the URL has to have the forum_id in it and that it needs to contain the additional parameters `foo=bar` and `baz=foo`:

Example 2.3.8 Generating a custom script URL using phorum_get_url()

```
// Build the URL for the add-on script.
$url = phorum_get_url(PHORUM_CUSTOM_URL, "myfile", 1, "foo= ↩
    bar", "baz=foo");

// After this, you could store the URL in the template data,  ↩
    so you
// can use it from the templates to link to the add-on script ↩
    .
// Here an example for filling the template variable {URL-> ↩
    MYFILE}:
$PHORUM["DATA"]["URL"]["MYFILE"] = $url;
```

2.3.6.3 Build URLs to files in the Phorum tree

If you have some non-script files in your module that you need to access through a URL, then make sure that you are generating absolute URLs for these. You can make use of the Phorum setting variable `$PHORUM['http_path']` to build these. This setting relates to the "HTTP Path" option under "General Settings" in the admin interface.

Let's asume you have a file named `foobar.gif` in your "foo" module tree, then you could generate the URL for that file like this:

Example 2.3.9 Generating an absolute URL for a file in a module directory

```
// Since the code probably runs inside a function scope,
// the global $PHORUM variable needs to be imported.
global $PHORUM;

// Build the URL for the foobar.gif.
$url = $PHORUM['http_path'] . "/mods/foo/foobar.gif";

// After this, you could store the URL in the template data,  ↩
    so you
// can use it from the templates. Here an example for filling ↩
     the
// template variable {MOD_FOO->IMAGE_URL}:
$PHORUM["DATA"]["MOD_FOO"]["IMAGE_URL"] = $url;
```

2.3.7 Implementing a settings screen for your module

Note: this feature is only available for modules that use the multiple file module structure

Some modules that you write might need to store settings for later use. For those, you can create a settings page that can be accessed from the "Modules" page in the admin interface.

The settings page must be put in your module's directory by the name of "`settings.php`". So for our example module "foo" the file would go in {phorum dir}/mods/foo/settings.php. In the admin interface under the option "Modules", a link to the settings.php page will automatically be added if the settings.php file is available for your module.

Although you can do anything you want in your settings.php script, it is recommended that you use the tools that are handed to you by Phorum for building pages and storing settings.
If the standard tools are not enough for building your settings page, then it is of course fine to do things differently (e.g. let your module create and use database tables or build the full settings interface using your own form code.)

2.3.7.1 Building input forms

The Phorum PHP object "PhorumInputForm" can be used to build input forms and table displays in the admin interface. The best thing you can do for learning the possibilities of this class is to look at other Phorum modules (like "bbcode" or "replace") or the core admin scripts from the `{phorumdir}/include/admin` directory.

2.3.7.2 Error and success feedback messages

For displaying error and success messages, make use of the functions `phorum_admin_error()` and `phorum_admin_okmsg()`. Both functions take the message to display as their argument. By using those functions, the messages are shown using the standard Phorum admin formatting.

2.3.7.3 Saving module settings to the database

Another tool is the function `phorum_db_update_settings()` that can be used for storing settings in the database. To store settings using this function, you can use code like this:

Example 2.3.10 Storing settings for a module in the database

```
// It is possible to store either scalars or arrays in a  ↩
    settings field.
// Phorum will automatically take care of correct handling.
$foo_settings = array();
$foo_settings["foodata"] = "Some setting data";
$foo_settings["bardata"] = "Some more setting data";

// Store the data in the database.
phorum_db_update_settings(array("mod_foo" => $foo_settings) ↩
    );

// In the next request, the settings data can be found in
// the variable $PHORUM['mod_foot'], for example:
print $PHORUM['mod_foo']['foodata'];
```

2.3.7.4 Prevent settings.php from being loaded directly

To ensure that your settings.php file is only loaded from the admin interface, place this line at the top of your settings.php file (see also XXX: secure against hackers):

```
if (!defined("PHORUM_ADMIN")) return;
```

2.3.7.5 Full module settings page example

Here is a full example settings page, using the tools from above. A real settings page will often be much larger than this, but the basics are the same.

Example 2.3.11 An example module settings.php script

```
<?php

if (!defined("PHORUM_ADMIN")) return;

// If data is posted, then store the posted settings in the ↩
    database.
if (count($_POST))
{
    $PHORUM['mod_foo']['field1'] = empty($_POST['field1'])  ↩
        ? 0 : 1;
    $PHORUM['mod_foo']['field2'] = (int) $_POST['field2'];

    // Do some error checking.
    if ($PHORUM['mod_foo']['field2'] > 1000) {
        phorum_admin_error("The value for field 2 is too  ↩
            high!");
    }
    // The data was okay. Store the settings.
    else {
        phorum_db_update_settings(array("mod_foo" =>  ↩
            $PHORUM["mod_foo"]));
        phorum_admin_okmsg('The settings were saved  ↩
            successfully');
    }
}

// This block is standard for every settings page. The "mod ↩
    " field
// must be set to the name of the module for which the  ↩
    settings
// page is written.
include_once "./include/admin/PhorumInputForm.php";
$frm = new PhorumInputForm ("", "post", "Submit this form") ↩
    ;
$frm->hidden("module", "modsettings");
$frm->hidden("mod", "foo");

// Add a header row to the form.
$frm->addbreak("Foo module settings");

// Add a checkbox to the form.
$row = $frm->addrow(
    "Field 1",
    $frm->checkbox("field1", "1", "Yes", $PHORUM['mod_foo ↩
        ']['field1'])
);

// Add a help balloon to Field 1.
$frm->addhelp(
    $row, "Field 1",
    "This is a help balloon text for Field 1."
);

// Add a text field to the form.
$frm->addrow(
    "Field 2",
    $frm->text_box("field2", $PHORUM['mod_foo']['field2'],  ↩
        50)
);
```

Chapter 3

Module hooks

3.1 Introduction

To satisfy the webmaster that needs every bell and whistle, or those that want to make their web site unique, the Phorum team created a very flexible hook & module system. The hooks allow a webmaster to create modules for doing things like using external authentication, altering message data before it is stored, adding custom information about users or messages, etc. Almost anything you can think of can be implemented through the hook & module system.

This chapter describes all the hooks that are available within the Phorum code. It is mainly targeted at developers that want to write modules.

3.2 Message search

3.2.1 search_redirect

Phorum does not jump to the search results page directly after posting the search form. Instead, it will first do a redirect to a secondary URL. This system is used, so Phorum can show an intermediate "Please wait while searching" page before doing the redirect. This is useful in case searching is taking a while, in which case users might otherwise repeatedly start hitting the search button when results don't show up immediately.

This hook can be used to modify the parameters that are used for building the redirect URL. This can be useful in case a search page is implemented that uses more fields than the standard search page.

Call time:

Right before the primary search redirect (for showing the "Please wait while searching" intermediate page) is done.

Hook input:

An array of phorum_get_url() parameters that will be used for building the redirect URL.

Hook output:
The possibly updated array of parameters.

3.2.2 search_output

This hook can be used to override the standard output for the search page. This can be useful for search modules that implement a different search backend which does not support the same options as Phorum's standard search backend.

Call time:
At the end of the search script, just before it loads the output template.

Hook input:
The name of the template to use for displaying the search page, which is "search" by default.

Hook output:
The possibly updated template name to load or NULL if the module handled the output on its own already.

3.3 Templating

3.3.1 javascript_register

Modules can provide JavaScript code that has to be added to the Phorum pages. Modules that make use of this facility should register the JavaScript code using this hook.

Call time:
At the start of the javascript.php script.

Hook input:
An array of registrations. Modules can register their JavaScript code for inclusion by adding a registration to this array. A registration is an array, containing the following fields:

- **module**
  The name of the module that adds the registration.
- **source**
  Specifies the source of the JavaScript data. This can be one of:
  - **file(<path to filename>)**
    For including a static JavaScript file. The path should be absolute or relative to the Phorum install directory, e.g. "`file(mods/foobar/baz.js)`". Because this file is loaded using a PHP include() call, it is possible to include PHP code in this file. Mind that this code is stored interpreted in the cache.
  - **template(<template name>)**
    For including a Phorum template, e.g. "`template(foobar::baz)`"

- **function(<function name>)**
  For calling a function to retrieve the JavaScript code, e.g. "`function(-mod_foobar_get_js)`"

- **cache_key**
  To make caching of the generated JavaScript code possible, the module should provide a cache key using this field. This cache key needs to change if the module will provide different JavaScript code.

  Note: in case "file" or "template" is used as the source, you are allowed to omit the cache_key. In that case, the modification time of the involved file(s) will be used as the cache key.

  It is okay for the module to provide multiple cache keys for different situations (e.g. if the JavaScript code depends on a group). Keep in mind though that for each different cache key, a separate cache file is generated. If you are generating different JavaScript code per user or so, then it might be better to add the JavaScript code differently (e.g. through a custom JavaScript generating script or by adding the code to the `$PHORUM['DATA']['HEAD_DATA']` variable). Also, do not use this to only add JavaScript code to certain phorum pages. Since the resulting JavaScript data is cached, it is no problem if you add the JavaScript code for your module to the code for every page.

Hook output:
The same array as the one that was used as the hook call argument, possibly extended with one or more registrations.

3.3.2 javascript_filter

This hook can be used to apply a filter to the Phorum JavaScript code. This can for example be used for compressing or cleaning up the JavaScript.

Call time:
Right after the javascript.php script has generated a new JavaScript file and right before storing that file in the cache. The filter hook will not be run for every request to javascript.php, but only in case the JavaScript code has to be refreshed.

Hook input:
The generated JavaScript code.

Hook output:
The filtered JavaScript code.

3.3.3 css_register

Modules can provide extra CSS data for CSS code that is retrieved through the css.php script. Extra CSS definitions can be added to the start and to the end of the base CSS code. Modules that make use of this facility should register the additional CSS code using this hook.

Call time:
At the start of the css.php script.
Hook input:
An array, containing the following fields:

- **css**
  The name of the css file that was requested for the css.php script. Phorum requests either "css" or "css_print". The module can use this parameter to decide whether CSS code has to be registered or not.

- **register**
  An array of registrations, filled by the modules. Modules can register their CSS code for inclusion in the base CSS file by adding a registration to this array. A registration is an array, containing the following fields:

  - **module**
    The name of the module that adds the registration.
  - **where**
    This field determines whether the CSS data is added before or after the base CSS code. The value for this field is either "before" or "after".
  - **source**
    Specifies the source of the CSS data. This can be one of:
    * **file(<path to filename>)**
      For including a static CSS file. The path should be absolute or relative to the Phorum install directory, e.g. "`file(mods/foobar/baz.css)`". Because this file is loaded using a PHP include() call, it is possible to include PHP code in this file. Mind that this code is stored interpreted in the cache.
    * **template(<template name>)**
      For including a Phorum template, e.g. "`template(foobar::baz)`"
    * **function(<function name>)**
      For calling a function to retrieve the CSS code, e.g. "`function(mod_foobar_get_css)`"
  - **cache_key**
    To make caching of the generated CSS data possible, the module should provide the css.php script a cache key using this field. This cache key needs to change if the module will provide different CSS data.

    Note: in case "file" or "template" is used as the source, you are allowed to omit the cache_key. In that case, the modification time of the involved file(s) will be used as the cache key.

    It is okay for the module to provide multiple cache keys for different situations (e.g. if the CSS code depends on a group or so). Keep in mind though

that for each different cache key, a separate cache file is generated. If you are generating different CSS code per user or so, then it might be better to add the CSS code differently (e.g. through a custom CSS generating script or by adding the CSS code to the $PHORUM['DATA']['HEAD_DATA'] variable. Also, do not use this to only add CSS code to certain phorum pages. Since the resulting CSS data is cached, it is no problem if you add the CSS data for your module to the CSS code for every page.

Hook output:

The same array as the one that was used for the hook call arguments, possibly with the "register" field updated. A module can add multiple registrations to the register array.

3.3.4 css_filter

(Phorum 5 >= 5.2.11)

This hook can be used to apply a filter to the Phorum CSS code. This can for example be used for compressing or cleaning up the CSS.

Call time:

Right after the css.php script has generated a new CSS file and right before storing that file in the cache. The filter hook will not be run for every request to css.php, but only in case the CSS code has to be refreshed.

Hook input:

The generated CSS code.

Hook output:

The filtered CSS code.

3.4 Login/Logout

3.4.1 before_logout

This hook can be used for performing tasks before a user logout. The user data will still be availbale in `$PHORUM["user"]` at this point.

Call time:

In `login.php`, just before destroying the user session.

Hook input:

None

Hook output:

None

3.4.2 after_logout

This hook can be used for performing tasks after a successful user logout and for changing the page to which the user will be redirected (by returning a different redirection URL). The user data will still be availbale in `$PHORUM["user"]` at this point.

Call time:

In `login.php`, after a logout, just before redirecting the user to a Phorum page.

Hook input:

The redirection URL.

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_after_logout($url)
{
    global $PHORUM;

    // Return to the site's main page on logout
    $url = $PHORUM["mod_foo"]["site_url"];

    return $url;
}
```

3.4.3 password_reset

(Phorum 5 >= 5.2.13)

This hook is called after handling a password reset request. Based on whether a user account can be found for the provided email address and what the account status for that user is, different actions are performed by Phorum before calling this hook:

- If no user account can be found for the provided email address, then nothing is done.
- If the account is not yet approved by a moderator, then no new password is generated for the user.
- If the account is active, then a new password is mailed to the user's email address.
- If the account is new and not yet confirmed by email, then a new account confirmation code is generated and sent to the user's email address.

The main purpose of this hook is to log password reset requests.

Call time:

In `login.php`, after handling a password reset request.

Hook input:

An array containing four elements:

- status: the password reset status, which can be: "new_password" (a new password was generated and sent for an active account), "new_verification" (a new account verification code was generated and sent for a new account that was not yet confirmed by email), "unapproved" (in case the account was not yet approved by a moderator, no new password or verification code was generated for the user) or "user_unknown" (when the provided email address cannot be found in the database).

- email: the email address that the user entered in the lost password form.
- user: a user data array. This is the user data for the email address that the user entered in the lost password form. If no matching user could be found (status = "user_unknown"), then this element will be NULL.
- secret: The new password or verification code for respectively the statuses "new_password" and "new_verification". For other statuses, this element will be NULL.

Hook output:
Same as input.
Example code:

```
function phorum_mod_foo_password_reset($data)
{
    $log = NULL;
    switch ($data['status'])
    {
        case 'new_password':
            $log = 'New password generated for ' .
                   $data['user']['username'] . ': ' .
                   $data['secret'];
            break;
        case 'new_verification':
            $log = 'New verification code generated for ' .
                   $data['user']['username'] . ': ' .
                   $data['secret'];
            break;
        case 'user_unknown':
            $log = 'Could not find a user for email ' .
                   $data['email'];
            break;
        case 'unapproved':
            $log = 'No new password generated for ' .
                   'unapproved user ' . $user['username'];
            break;
    }

    if ($log !== NULL) {
        log_the_password_reset($log);
    }

    return $user;
}
```

3.4.4 after_login

This hook can be used for performing tasks after a successful user login and for changing the page to which the user will be redirected (by returning a different redirection

URL). If you need to access the user data, then you can do this through the global `$PHORUM` variable. The user data will be in `$PHORUM["user"]`.

Call time:

In `login.php`, after a successful login, just before redirecting the user to a Phorum page.

Hook input:

The redirection URL.

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_after_login($url)
{
    global $PHORUM;

    // Redirect to the user's chosen page
    $url = $PHORUM["user"]["phorum_mod_foo_user_login_url"];

    return $url;
}
```

3.4.5 failed_login

This hook can be used for tracking failing login attempts. This can be used for things like logging or implementing login failure penalties (like temporary denying access after X login attempts).

Call time:

In `login.php`, when a user login fails.

Hook input:

An array containing three fields (read-only):

- username
- password
- location
  - The location field specifies where the login failure occurred and its value can be either `forum` or `admin`.

Hook output:

None

Example code:

```
function phorum_mod_foo_failed_login($data)
{
    global $PHORUM;

    // Get the current timestamp
```

```
    $curr_time = time();

    // Check for a previous login failure from the current
    // IP address
    if (!empty($PHORUM["mod_foo"]["login_failures"][$_SERVER ↩
        ["REMOTE_ADDR"]])) {
        // If the failures occur within the set time window,
        // increment the login failure count
        if ($curr_time <= ($PHORUM["mod_foo"]["login_failures ↩
            "][$_SERVER["REMOTE_ADDR"]]["timestamp"] + (int) ↩
            $PHORUM["mod_foo"]["login_failures_time_window"]) ↩
            ) {
            $PHORUM["mod_foo"]["login_failures"][$_SERVER[" ↩
                REMOTE_ADDR"]]["login_failure_count"] ++;
            $PHORUM["mod_foo"]["login_failures"][$_SERVER[" ↩
                REMOTE_ADDR"]]["timestamp"] = $curr_time;
        // Otherwise, reset the count.
        } else {
            $PHORUM["mod_foo"]["login_failures"][$_SERVER[" ↩
                REMOTE_ADDR"]]["login_failure_count"] = 1;
            $PHORUM["mod_foo"]["login_failures"][$_SERVER[" ↩
                REMOTE_ADDR"]]["timestamp"] = $curr_time;
    } else {
        // Log the timestamp and IP address of a login  ↩
            failure
        $PHORUM["mod_foo"]["login_failures"][$_SERVER[" ↩
            REMOTE_ADDR"]]["login_failure_count"] = 1;
        $PHORUM["mod_foo"]["login_failures"][$_SERVER[" ↩
            REMOTE_ADDR"]]["timestamp"] = $curr_time;
    }
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));
}
```

3.5 Private message system

3.5.1 pm_delete_folder

(Phorum 5 >= 5.2.13)

This hook can be used for working on deletion of a private message folder. E.g. for deleting messages in the folder before.

Call time:

Right before Phorum deletes the private message folder.

Hook input:

The id of the private message folder going to be deleted.

Hook output:

None A clean module should return its input again to have the same data available to all modules working in this hook.

Example code:

```
function phorum_mod_foo_pm_delete_folder($folder_id)
{
    // do something with the folder going to be deleted

    return $folder_id;
}
```

3.5.2 pm_delete

(Phorum 5 >= 5.2.13)

This hook can be used for working deletion of a private message.

Call time:

Right before Phorum deletes the private message.

Hook input:

The id of the private message going to be deleted.

Hook output:

None A clean module should return its input again to have the same data available to all modules working in this hook.

Example code:

```
function phorum_mod_foo_pm_delete($pm_id)
{
    // do something with the message going to be deleted

    return $pm_id;
}
```

3.5.3 pm_list

(Phorum 5 >= 5.2.7)

This hook can be used for reformatting a list of private messages.

Call time:

Right after Phorum has formatted the private message list. This is primarily done when a list of private messages is shown in the private message system.

Hook input:

An array of private message info arrays.

Hook output:

The same array as was used for the hook call argument, possibly with some updated fields in it.

Example code:

```
function phorum_mod_foo_pm_list($messages)
{
    // Filter out private messages that are sent by
    // evil user X with user_id 666.
    foreach ($messages as $id => $message) {
        if ($message['user_id'] == 666) {
            unset($messages[$id]);
        }
    }
    return $messages;
}
```

3.5.4 pm_read

(Phorum 5 >= 5.2.7)

This hook can be used for reformatting a single private message for reading.

Call time:

Right after Phorum has formatted the private message. This is primarily done when a private message read page is shown in the private message system.

Hook input:

An array, describing a single private message.

Hook output:

The same array as was used for the hook call argument, possibly with some updated fields in it. [example}

```
function phorum_mod_foo_pm_read($message)
{
// Add a notice to messages that were sent by
// evil user X with user_id 666.
if ($message['user_id'] == 666) {
$message['subject'] .= ' <strong>EVIL!</strong>';
}
return $message;
}
```

3.6 Buddies system

3.6.1 buddy_list

(Phorum 5 >= 5.2.7)

This hook can be used for reformatting a list of buddies. Reformatting could mean things like changing the sort order or modifying the fields in the buddy arrays.

Call time:

Right after Phorum has formatted the buddy list. This is primarily done when the list of buddies is shown in the private message system.

Hook input:

An array of buddy info arrays. Each info array contains a couple of fields that describe the budy: user_id, display_name, mutual (0 = not mutual, 1 = mutual), URL->PROFILE, date_last_active (formatted date) and raw_date_last_active (Epoch timestamp).

Hook output:

The same array as was used for the hook call argument, possibly with some updated fields in it.

Example code:

```
function phorum_mod_foo_buddy_list($buddies)
{
    // Add a CSS class around the display names for
    // the mutual buddies (of course this could also
    // easily be implemented as a pure template change,
    // but remember that this is just an example).
    foreach ($buddies as $id => $buddy)
    {
        if ($buddy['mutual'])
        {
            $buddies[$id]['display_name'] =
                '<span class="mutual_buddy">' .
                $buddy['display_name'] .
                '</span>';
        }
    }

    return $buddies;
}
```

3.7 Module hooks

3.7.1 bbcode_register

This hook is implemented by the BBcode module in the file `mods/bbcode/api.php`. It allows modules to provide extra or override existing BBcode tag descriptions.

Warning: do not delete tags from the list, e.g. removing a tag based on the login status for a user. That would throw off and invalidate the caching mechanisms. If you need to have some tag act differently for different users, then override the behavior for the tag using a callback function and implement the logic in the callback function.

Call time:

This hook is called from the function `bbcode_api_initparser()` in the BBcode module file `mods/bbcode/api.php`.

Hook input:

An array of tag description arrays. The keys in this array are tag names. The values are arrays describing the tags. For examples of what these tag descriptions look like, please take a look at the file `mods/bbcode/builtin_tags.php`.

Hook output:

The same array as the one that was used for the hook call arguments, possibly updated with new or updated tags.

3.8 Message handling

3.8.1 posting_init

This hook can be used for doing modifications to the environment of the posting scripts at an early stage. One of the intended purposes of this hook is to give mods a chance to change the configuration of the posting fields in `$PHORUM["post_fields"]`.

Call time:

Right after the `posting.php` script's configuration setup and before starting the posting script processing.

Hook input:

None

Hook output:

None

Example code:

```
function phorum_mod_foo_posting_init ()
{
    global $PHORUM;

    //add the default, descriptive text to the message body
    $PHORUM["post_fields"]["body"][pf_INIT] = $PHORUM["DATA ↩
        "]["LANG"]["mod_foo"]["default_body_text"];

}
```

3.8.2 posting_permissions

This hook can be used for setting up custom abilities and permissions for users, by updating the applicable fields in `$GLOBALS["PHORUM"]["DATA"]` (e.g. for giving certain users the right to make postings sticky, without having to make the full moderator for a forum).

Read the code in `posting.php` before this hook is called to find out what fields can be used.

Beware: Only use this hook if you know what you are doing and understand Phorum's editor permission code. If used wrong, you can open up security holes in your Phorum installation!

Call time:

In `posting.php` right after Phorum has determined all abilities that apply to the logged in user.

Hook input:

None

Hook output:

None

Example code:

```
function phorum_mod_foo_posting_permissions ()
{
    global $PHORUM;

    // get the previously stored id for the "sticky_allowed"  ↩
        group
    $mod_foo_group_id = $PHORUM["mod_foo"][" ↩
        sticky_allowed_group_id"];

    // allow creating sticky posts for users in the " ↩
        sticky_allowed"
    // group, if the option has not already been enabled.
    if (!$PHORUM["DATA"]["OPTION_ALLOWED"]["sticky"])
        $PHORUM["DATA"]["OPTION_ALLOWED"]["sticky"] =  ↩
            phorum_api_user_check_group_access ( ↩
            PHORUM_USER_GROUP_APPROVED, $mod_foo_group_id);
}
```

3.8.3 posting_custom_action

This hook can be used by modules to handle (custom) data coming from the posting form. The module is allowed to change the data that is in the input message. When a module needs to change the meta data for a message, then this is the designated hook for that task.

Call time:

In `posting.php` right after all the initialization tasks are done and just before the posting script starts its own action processing.

Hook input:

Array containing message data.

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_posting_custom_actions ($message)
{
    global $PHORUM;

    // for some reason, create an MD5 signature for the  ↩
        original body
```

```
    if (!empty($message["body"])
        $message["meta"]["mod_foo"]["body_md5"] = md5( ↩
            $message["body"]);

    return $message;
}
```

3.8.4 before_editor

This hook can be used for changing message data, just before the editor is displayed. This is done after escaping message data for XSS prevention is done. So in the hook, the module writer will have to be aware that data is escaped and that he has to escape data himself if needed.

This hook is called every time the editor is displayed. If modifying the message data does not have to be done on every request (for example only on the first request when replying to a message), the module will have to check the state the editor is in. Here's some hints on what you could do to accomplish this:

- Check the editor mode: this can be done by looking at the "mode" field in the message data. This field can be one of "post", "reply" and "edit".
- Check if it's the first request: this can be done by looking at the `$_POST` array. If no field "message_id" can be found in there, the editor is handing the first request.

Beware: this hook function only changes message data before it is displayed in the editor. From the editor, the user can still change the data. Therefore, this hook cannot be used to control the data which will be stored in the database. If you need that functionality, then use the hooks [??] and/or [??] instead.

Call time:
In `posting.php` just before the message editor is displayed.
Hook input:
Array containing data for the message that will be shown in the editor screen.
Hook output:
Same as input.
Example code:

```
// Using this, an example hook function that appends the  ↩
    string
// "FOO!" to the subject when replying to a message (how  ↩
    useful ;-)
// could look like this:
function phorum_mod_foo_before_editor ($data)
{
```

```
    if ($data["mode"] == "reply" && ! isset($_POST[" ↩
        message_id])) {
        $data["reply"] = $data["reply"] . " FOO!";
    }

    return $data;
}
```

3.8.5 check_post

This hook can be used for modifying the message data and for running additional checks on the data. If an error is put in `$error`, Phorum will stop posting the message and show the error to the user in the post-form.

Beware that `$error` can already contain an error on input, in case multiple modules are run for this hook. Therefore you might want to return immediately in your hook function in case `$error` is already set.

Below is an example of how a function for this hook could look. This example will disallow the use of the word "bar" in the message body.

Call time:

In the `include/posting/check_integrity.php` file, right after performing preliminary posting checks, unless these checks have returned something bad.

Hook input:

An array containing:

- An array of the message data.
- `$error`, used to return an error message

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_check_post ($args) {
    list ($message, $error) = $args;
    if (!empty($error)) return $args;

    if (stristr($message["body"], "bar") !== false) {
        return array($message, "The body may not contain 'bar ↩
            '");
    }

    return $args;
}
```

3.8.6 before_edit

This hook can be used to change the edited message before it is stored in the database.

Call time:

In `include/posting/action_edit.php`, right before storing an edited message in the database.

Hook input:

An array containing message data.

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_before_edit($dbmessage)
{
    global $PHORUM;

    // If the message body does not contain the disclaimer,  ↩
        add it
    if (strpos($dbmessage["body"], $PHORUM["DATA"]["LANG"][" ↩
        mod_foo"]["Disclaimer"]) === false) {
        $dbmessage["body"] .= "\n".$PHORUM["DATA"]["LANG"][" ↩
            mod_foo"]["Disclaimer"];
    }

    return $dbmessage;
}
```

3.8.7 after_edit

This hook can be used for sending notifications or for making log entries in the database when editing takes place.

Call time:

In `include/posting/action_edit.php`, right after storing an edited message in the database.

Hook input:

An array containing message data (read-only).

Hook output:

None

Example code:

```
function phorum_mod_foo_after_edit($dbmessage)
{
    global $PHORUM;

    // If the message editor is not the same as the message  ↩
        author, alert
    // the message author that their message has been edited
```

```
    if ($PHORUM["user"]["user_id"] != $dbmessage["user_id"])  ↩
        {
        $pm_message = preg_replace(
            "/%message_subject%/",
            $dbmessage["subject"],
            $PHORUM["DATA"]["LANG"]["mod_foo"][" ↩
                MessageEditedBody"]
            );
        phorum_db_pm_send(
            $PHORUM["DATA"]["LANG"]["mod_foo"][" ↩
                MessageEditedSubject"],
            $pm_message,
            $dbmessage["user_id"]
            );
    }
}
```

3.8.8 before_post

This hook can be used to change the new message data before storing it in the database.

Call time:

In `include/posting/action_post.php`, right before storing a new message in the database.

Hook input:

An array containing message data.

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_before_post($message)
{
    global $PHORUM;

    // Add the disclaimer to the new message body
    $message["body"] .= "\n".$PHORUM["DATA"]["LANG"]["mod_foo ↩
        "]["Disclaimer"];

    return $message;
}
```

3.8.9 after_message_save

This hook can be used for performing actions based on what the message contained or altering it before it is emailed to the subscribed users. It is also useful for adding or removing subscriptions.

Call time:

In `include/posting/action_post.php`, right after storing a new message and all database updates are done.

Hook input:
An array containing message data.

Hook output:
Same as input.

Example code:

```
function phorum_mod_foo_after_message_save($message)
{
    global $PHORUM;

    // If the message was posted in a monitored forum, log  ↩
        the id
    if (in_array($message["forum_id"], $PHORUM["mod_foo"][" ↩
        monitored_forums"])) {
        $PHORUM["mod_foo"]["monitored_messages"][$message[" ↩
            forum_id"]][] = $message["message_id"];
    }

    return $message;
}
```

3.8.10 after_post

This hook can be used for performing actions based on what the message contained. It is specifically useful for altering the redirect behavior.

Call time:
In `include/posting/action_post.php`, after all the posting work is done and just before the user is redirected back to the message list page.

Hook input:
An array containing message data.

Hook output:
Same as input.

Example code:

```
function phorum_mod_foo_after_post($message)
{
    global $PHORUM;

    // remove the post count increment for the user in select ↩
         forums
    if (in_array($message["forum_id"], $PHORUM["mod_foo"][" ↩
        forums_to_ignore"])) {
        phorum_api_user_save (
            array (
                "user_id"    => $PHORUM["user"]["user_id"],
                "posts"      => $PHORUM["user"]["posts"]
```

```
                )
            );
    }

    return $message;
}
```

3.9 Miscellaneous

3.9.1 ajax_<call>

(Phorum 5 >= 5.2.8)
This hook allows module writers to implement calls for the Phorum Ajax layer.

The "call" argument from the Ajax argument array is used to construct the name of the hook that will be called. For example for the call "sayhello" the called hook will be `call_sayhello`

A call implementation should always be using the provided functions `phorum_ajax_return()` and `phorum_ajax_error()` to return data to the client. Because these functions will call `exit` after they are done, hook functions that implement an Ajax call stop page execution and do not return like other hook functions. Only if the hook function decides for some reason that the Ajax call is not to be handled by the module, it can return the Ajax argument array.

Call time:

Just before ajax.php tries to find a built-in handler script for an Ajax call. Therefore, this hook can also be used to override core Ajax call implementations. We strongly discourage doing so though.

Hook input:

The Ajax argument array

Hook output:

The same array as the one that was used for the hook call argument.

Example code:

```
function phorum_mod_foo_ajax_sayhello($ajax_args)
{
    // An optional name=.... argument can be used in the  ↩
        request.
    $name = phorum_ajax_getarg('name', 'string', 'Anonymous  ↩
        Person');

    // This will return a JSON encoded string to the client.
    phorum_ajax_return("Hello, $name");
}
```

For this hook implementation, a GET based URL to fire this Ajax call could look like `http://example.com/ajax.php?call=sayhello,name=JohnDoe`.

3.9.2 database_error

Give modules a chance to handle or process database errors. This can be useful to implement addional logging backends and/or alerting mechanisms. Another option is to fully override Phorum's default database error handling by handling the error and then calling exit() from the hook to prevent the default Phorum code from running.

Note: If you decide to use the full override scenario, then it is best to make your module run the database_error hook last, so other modules can still run their hook handling before the script exits. To accomplish this, add this to your module info:

```
priority: run hook database_error after *
```

Call time:

At the start of the function `phorum_database_error` (which you can find in `common.php`). This function is called from the database layer when some database error occurs.

Hook input:

The error message that was returned from the database layer. This error is not HTML escaped, so if you send it to the browser, be sure to preprocess it using html-specialchars().

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_database_error($error)
{
    // Log database errors to syslog facility "LOCAL0".
    openlog("Phorum", LOG_PID | LOG_PERROR, LOG_LOCAL0);
    syslog(LOG_ERR, $error);

    return $error;
}
```

3.9.3 external

The external hook functions are never called from any of the standard Phorum pages. These functions are called by invoking `script.php` on the command line with the `--module` parameter. This can be used to pipe output from some arbitrary command to a specific module, which can do something with that input. If your module does not need any command line input and is meant to be run on a regular basis, you should consider using the [??] hook.

Mind that for using an [??] hook, the module in which it is handled must be enabled in your admin interface. So if an [??] hook is not running, the containing module might be disabled.

To run this hook from the command line, you have to be in the Phorum installation directory. So running the [??] hook of a module named `external_foo` would be done like this on a UNIX system prompt:

```
# cd /your/phorum/dir
# php ./script.php --module=external_foo
```

For easy use, you can of course put these commands in a script file.

Call time:
In the `script.php` when called from the command prompt or a script file.

Hook input:
Any array of arguments. (Optional)

Hook output:
None

3.9.4 scheduled

[??] hook functions are similar to [??] ones, except these functions do not require any input from the command line. The modules containing this hook are invoked by running `script.php` with the `--scheduled` argument (no module name is taken; this argument will run all scheduled hooks for all available modules).

Like the name of the hook already suggests, this hook can be used for creating tasks which have to be executed on a regular basis. To archieve this, you can let `script.php` run from a scheduling service (like a cron job on a UNIX system).

In general, [??] hooks are used for automating tasks you want to execute without having to perform any manual action. Practical uses for a scheduled hook could be:

- housekeeping (cleanup of stale/old data)
- daily content generation (like sending daily digests containing all posted messages for that day)
- forum statistics generation

Keep in mind that for using this hook, the module in which it is handled must be enabled in your admin interface. So if this hook is not running, the containing module might be disabled.

To run this hook from the command line or from a scheduling service, you have to be in the Phorum installation directory. So running this hook for your Phorum installation would be done like this on a UNIX system prompt:

```
# cd /your/phorum/dir
# php ./script.php --scheduled
```

When creating a scheduling service entry for running this automatically, remember to change the directory as well. You might also have to use the full path to your PHP

binary (`/usr/bin/php` or whatever it is on your system), because the scheduling service might not know the path to it. An entry for the cron system on UNIX could look like this:

```
0 0 * * * cd /your/phorum/dir && /usr/bin/php ./script.php -- ↩
    scheduled
```

Please refer to your system's documentation to see how to use your system's scheduling service.

Call time:

In the `script.php` when called from the command prompt or a script file with the `--scheduled` argument.

Hook input:

None

Hook output:

None

3.10 Request initialization

3.10.1 parse_request

This hook gives modules a chance to tweak the request environment, before Phorum parses and handles the request data. For tweaking the request environment, some of the options are:

- Changing the value of `$_REQUEST["forum_id"]` to override the used forum_id.
- Changing the value of `$_SERVER["QUERY_STRING"]` or setting the global override variable `$PHORUM_CUSTOM_QUERY_STRING` to feed Phorum a different query string than the one provided by the webserver.

Tweaking the request data should result in data that Phorum can handle.

Call time:

Right before Phorum runs the request parsing code in `common.php`.

Hook input:

No input.

Hook output:

No output.

Example code:

```
function phorum_mod_foo_parse_request()
{
    // Override the query string.
    global $PHORUM_CUSTOM_QUERY_STRING
    $PHORUM_CUSTOM_QUERY_STRING = "1,some,phorum,query=string ↩
        ";
```

```
    // Override the forum_id.
    $_SERVER['forum_id'] = "1234";
}
```

3.10.2 common_pre

This hook can be used for overriding settings that were loaded and setup at the start of the `common.php` script. If you want to dynamically assign and tweak certain settings, then this is the designated hook to use for that.

Because the hook was put after the request parsing phase, you can make use of the request data that is stored in the global variables `$PHORUM['forum_id']` and `$PHORUM['args']`.

Call time:

Right after loading the settings from the database and parsing the request, but before making descisions on user, language and template.

Hook input:

No input.

Hook output:

No output.

Example code:

```
function phorum_mod_foo_common_pre()
{
    global $PHORUM;

    // If we are in the forum with id = 10, we set the  ↩
        administrator
    // email information to a different value than the one  ↩
        configured
    // in the general settings.
    if ($PHORUM["forum_id"] == 10)
    {
        $PHORUM["system_email_from_name"] = "John Doe";
        $PHORUM["system_email_from_address"] = "John. ↩
            Doe@example.com";
    }
}
```

3.10.3 common_no_forum

This hook is called in case a forum_id is requested for an unknown or inaccessible forum. It can be used for doing things like logging the bad requests or fully overriding Phorum's default behavior for these cases (which is redirecting the user back to the index page).

Call time:

In `common.php`, right after detecting that a requested forum does not exist or is inaccessible and right before redirecting the user back to the Phorum index page.

Hook input:

No input.

Hook output:

No output.

Example code:

```
function phorum_mod_foo_common_no_forum()
{
    // Return a 404 Not found error instead of redirecting
    // the user back to the index.
    header("HTTP/1.0 404 Not Found");
    print "<html><head>\n";
    print "  <title>404 - Not Found</title>\n";
    print "</head><body>";
    print "  <h1>404 - Forum Not Found</h1>";
    print "</body></html>";
    exit();
}
```

3.10.4 common_post_user

This hook gives modules a chance to override Phorum variables and settings, after the active user has been loaded. The settings for the active forum are also loaded before this hook is called, therefore this hook can be used for overriding general settings, forum settings and user settings.

Call time:

Right after loading the data for the active user in `common.php`, but before deciding on the language and template to use.

Hook input:

No input.

Hook output:

No output.

Example code:

```
function phorum_mod_foo_common_post_user()
{
    global $PHORUM;

    // Switch the read mode for admin users to threaded.
    if ($PHORUM['user']['user_id'] && $PHORUM['user']['admin ↩
        ']) {
        $PHORUM['threaded_read'] = PHORUM_THREADED_ON;
    }

    // Disable "float_to_top" for anonymous users.
    if (!$PHORUM['user']['user_id']) {
```

```
        $PHORUM['float_to_top'] = 0;
    }
}
```

3.10.5 common

This hook gives modules a chance to override Phorum variables and settings near the end of the `common.php` script. This can be used to override the Phorum (settings) variables that are setup during this script.

Call time:
At the end of `common.php`.

Hook input:
No input.

Hook output:
No output.

Example code:

```
function phorum_mod_foo_common()
{
    global $PHORUM;

    // Override the admin email address.
    $PHORUM["system_email_from_name"] = "John Doe";
    $PHORUM["system_email_from_address"] = "John.Doe@example. ↩
        com";
}
```

3.10.6 page_<phorum_page>

(Phorum 5 >= 5.2.7)

This hook gives modules a chance to run hook code for a specific Phorum page near the end of the the `common.php` script.

It gives modules a chance to override Phorum variables and settings near the end of the `common.php` script. This can be used to override the Phorum (settings) variables that are setup during this script.
The `phorum_page` definition that is set for each script is used to construct the name of the hook that will be called. For example the `index.php` script uses phorum_page `index`, which means that the called hook will be `page_index`.

Call time:
At the end of `common.php`, right after the [??] hook is called.

You can look at this as if the hook is called at the start of the called script, since including `common.php` is about the first thing that a Phorum script does.

Hook input:
No input.

Hook output:
No output.
Example code:

```
function phorum_mod_foo_page_list()
{
    global $PHORUM;

    // Set the type of list page to use, based on a cookie.
    if (empty($_COOKIE['list_style'])) {
        $PHORUM['threaded_list'] = PHORUM_THREADED_DEFAULT;
    } elseif ($_COOKIE['list_style'] == 'threaded') {
        $PHORUM['threaded_list'] = PHORUM_THREADED_ON;
    } elseif ($_COOKIE['list_style'] == 'flat') {
        $PHORUM['threaded_list'] = PHORUM_THREADED_OFF;
    } elseif ($_COOKIE['list_style'] == 'hybrid') {
        $PHORUM['threaded_list'] = PHORUM_THREADED_HYBRID;
    }
}
```

3.11 Page output

3.11.1 phorum_shutdown

This hook gives modules a chance to easily hook into PHP's register_shutdown_function() functionality.

Code that you put in a phorum_shutdown hook will be run after running a Phorum script finishes. This hook can be considered an expert hook. Only use it if you really need it and if you are aware of implementation details of PHP's shutdown functionality.

Call time:
After running a Phorum script finishes.
Hook input:
No input.
Hook output:
No output.

3.11.2 get_template_file

(Phorum 5 >= 5.2.11)

Allow modules to have influence on the results of the phorum_get_template_file() function. This function translates a page name (e.g. `list`) into a filename to use as the template source for that page (e.g. `/path/to/phorum/templates/emerald/list.tpl`).

Call time:
At the start of the phorum_get_template_file() function from `common.php`.

Hook input:
An array containing two elements:

- page: The page that was requested.
- source: The file that has to be used as the source for the page. This one is initialized as NULL.

Hook output:
Same as input. Modules can override either or both of the array elements. When the "source" element is set after running the hook, then the file named in this element is directly used as the template source. It must end in either ".php" or ".tpl" to be accepted as a template source. Phorum does not do any additional checking on this source file name. It is the module's duty to provide a correct source file name.
Otherwise, the template source file is determined based on the value of the "page" element, following the standard Phorum template resolving rules.

Example code:

```
function phorum_mod_foo_get_template_file($data)
{
    // Override the index template with a custom template
    // from the "foo" module.
    if ($data['page'] == 'index_new') {
        $data['page'] = 'foo::index_new';
    }

    // Point the "pm" template directly at a custom PHP  ↩
        script.
    if ($data['page'] == 'pm') {
        $data['source'] = './mods/foo/pm_output_handler.php';
    }

    return $data;
}
```

3.11.3 start_output

This hook gives modules a chance to apply some last minute changes to the Phorum data. You can also use this hook to call ob_start() if you need to buffer Phorum's full output (e.g. to do some post processing on the data from the [??] hook.

Note: this hook is only called for standard pages (the ones that are constructed using a header, body and footer) and not for output from scripts that do raw output like `file.php`, `javascript.php`, `css.php` and `rss.php`.

Call time:
After setting up all Phorum data, right before sending the page header template.

Hook input:
No input.

Hook output:
No output.
Example code:

```
function phorum_mod_foo_start_output()
{
    global $PHORUM;

    // Add some custom data to the page title.
    $title = $PHORUM['DATA']['HTML_TITLE'];
    $PHORUM['DATA']['HTML_TITLE'] = "-=| Phorum Rocks! |=-  ↩
        $title";
}
```

3.11.4 after_header

This hook can be used for adding content to the pages that is displayed after the page header template, but before the main page content.

Call time:
After sending the page header template, but before sending the main page content.
Hook input:
No input.
Hook output:
No output.
Example code:

```
function phorum_mod_foo_after_header()
{
    // Only add data after the header for the index and list  ↩
        pages.
    if (phorum_page != 'index' && phorum_page != 'list')  ↩
        return;

    // Add some static notification after the header.
    print '<div style="border:1px solid orange; padding: 1em ↩
        ">';
    print 'Welcome to our forums!';
    print '</div>';
}
```

3.11.5 before_footer

This hook can be used for adding content to the pages that is displayed after the main page content, but before the page footer.

Call time:
After sending the main page content, but before sending the page footer template.
Hook input:

No input.

Hook output:

No output.

Example code:

```
function phorum_mod_foo_before_footer()
{
    // Add some static notification before the footer.
    print '<div style="font-size: 90%">';
    print '  For technical support, please send a mail to ';
    print '  <a href="mailto:tech@example.com">the webmaster ↩
        </a>.';
    print '</div>';
}
```

3.11.6 end_output

This hook can be used for performing post output tasks. One of the things that you could use this for, is for reading in buffered output using ob_get_contents() in case you started buffering using ob_start() from the [??] hook.

Call time:

After sending the page footer template.

Hook input:

No input.

Hook output:

No output.

Example code:

```
function phorum_mod_foo_end_output()
{
    // Some made up call to some fake statistics package.
    include("/usr/share/lib/footracker.php");
    footracker_register_request();
}
```

3.12 File storage

3.12.1 after_detach

The primary use of this hook would be for creating an alternate storage system for attachments. Using this hook, you can delete the file from your alternate storage.

Call time:

In `include/posting/action_attachments.php`, right after a file attachment is deleted from the database.

Hook input:

Two part array where the first element is the message array and the second element is a file array that contains the name, size, and `file_id` of the deleted file.

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_reopen_after_detach($data)
{
    global $PHORUM;

    // Remove the attachment from the log of messages with
    // attachments
    unset($PHORUM["mod_foo"]["messages_with_attachments"][ ↩
        $data[0]["message_id"]][$data[1]["file_id"]]);

    // If there are now no attachments on the current
    // message, remove the message from the log
    if (empty($PHORUM["mod_foo"]["messages_with_attachments ↩
        "][$data[0]["message_id"]]))
        unset($PHORUM["mod_foo"]["messages_with_attachments ↩
            "][$data[0]["message_id"]]);
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));

    return $data;
}
```

3.12.2 before_attach

The primary use of this hook would be for creating an alternate storage system for attachments. You would need to use the [??] hook to complete the process as you do not yet have the `file_id` for the file. You will need to use the [??] hook to retreive the file data later.

Call time:

In `include/posting/action_attachments.php`, right before a file attachment is saved in the database.

Hook input:

Two part array where the first element is the message array and the second element is a file array that contains the name, size, and file data.

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_reopen_before_attach($data)
{
    // Save the file with the amazing alternate_file_storage
    // function I haven't yet created
    alternate_file_storage($data[1]);
```

```
    // Remove the file data saved with the  ↩
        alterante_file_storage
    // function
    $data[1]["file_data"] = "";

    return $data;
}
```

3.12.3 after_attach

The primary use of this hook would be for creating an alternate storage system for attachments. You would need to use the [??] hook to remove the file data and in this hook it could be saved properly. You will need to use the [??] hook to retreive the file data later.

Call time:

In `include/posting/action_attachments.php`, right after a file attachment is saved in the database.

Hook input:

Two part array where the first element is the message array and the second element is a file array that contains the name, size, and `file_id` of the newly saved file.

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_reopen_after_attach($data)
{
    global $PHORUM;

    // Log the messages with attachments, including the
    // attachment names
    $PHORUM["mod_foo"]["messages_with_attachments"][$data ↩
        [0]["message_id"]][$data[1]["file_id"]] = $data[1][" ↩
        name"];
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));

    return $data;
}
```

3.12.4 system_max_upload

This hook allows a module to control the maximum file size for a file upload. Most notable would be file system storage. It could ignore the db_limit.

Call time:

In `include/upload_functions.php`, in the function phorum_get_system_max_upload.

Hook input:

An array containing the default limit, the data layer limit and the PHP limit

Hook output:

A 3 part array with the limits adjusted as you wish. The first element in the array would be the most important.

Example code:

```
function phorum_mod_foo_system_max_upload($data)
{
    // ignore the db_limit
    $data[0] = $data[2];
    return $data;
}
```

3.12.5 file_retrieve

This hook allows modules to handle the file data retrieval. The hook can use `phorum_api_error_set()` to return an error. Hooks should be aware that their input might not be `$file`, but `FALSE` instead, in which case they should immediately return `FALSE` themselves.

Call time:

In `include/api/file_storage.php`, right before a file attachment is retrieved from the database.

Hook input:

Two part array where the first element is an empty file array and the second element is the flags variable.

Hook output:

Same as input with file_data filled in.

3.12.6 file_purge_stale

This hook can be used to feed the file storage API function phorum_api_file_purge_stale() extra stale files. This can be useful for modules that handle their own files, using a custom link type.

Call time:

Right after Phorum created its own list of stale files.

Hook input:

An array containing stale files, indexed by file_id. Each item in this array is an array on its own, containing the following fields:

- file_id: the file id of the stale file
- filename: the name of the stale file
- filesize: the size of the file in bytes
- add_datetime: the time (epoch) at which the file was added

- reason: the reason why it's a stale file

Hook output:
The same array as the one that was used for the hook call argument, possibly extended with extra files that are considered to be stale.

3.13 Admin interface

3.13.1 admin_general

This hook can be used for adding items to the form on the "General Settings" page of the admin interface.

Call time:
Right before the `PhorumInputForm` object is shown.

Hook input:
The `PhorumInputForm` object.

Hook output:
Same as input.

Example code:

```
function phorum_mod_foo_admin_general ($frm)
{
    // Add a section for the foo settings
    $frm->addbreak( "Foo Module Settings" );

    // Add the option to cache the bar
    $row=$frm->addrow( "Enable Bar Caching:", $frm-> ↩
        select_tag( "mod_foo[enable_bar_caching]", array( "No ↩
        ", "Yes" ), $PHORUM["mod_foo"]["enable_bar_caching"]  ↩
        ) );
    $frm->addhelp($row, "Enable Bar Caching", "If you select  ↩
        yes for this option, then the bar will be cached." );

    // Return the modified PhorumInputForm
    return $frm;

}
```

3.13.2 admin_css_file

This hook can be used to pull in an alternate css file for the admin screens. That's pretty much all it's useful for. This hook is allowed to change the path to the admin css files because if we didn't allow it, someone would request it.

Call time:
Just before output begins on the admin page.

Hook input:
A string containing the path to the css file which will be used for the admin page.

Hook output:
The path to the actual css file to use.
Example code:

```
function phorum_mod_foo_admin_css_file($cssfile)
{
    // Force admin screens to use the "bar.css" style sheet.
    $pieces = explode('/', $cssfile);
    $pieces[count($pieces)-1] = 'bar.css';
    $cssfile = implode('/', $pieces);
    return $cssfile;
}
```

3.14 Moderation

3.14.1 email_user_start

This hook is put at the very beginning of `phorum_email_user()` and is therefore called for *every* email that is sent from Phorum. It is put before every replacement done in that function so that all data which is sent to that function can be replaced/changed at will.

Call time:
In the file `email_functions.php` at the start of `phorum_email_user()`, before any modification of data.

Hook input:
An array containing:

- An array of addresses.
- An array containing the message data.

Hook output:
Same as input.
Example code:

```
function phorum_mod_foo_email_user_start (list($addresses, ↩
    $data))
{
    global $PHORUM;

    // Add our disclaimer to the end of every email message.
    $data["mailmessage"] = $PHORUM["mod_foo"][" ↩
        email_disclaimer"];

    return array($addresses, $data);
}
```

3.14.2 send_mail

This hook can be used for implementing an alternative mail sending system. The hook should return true if Phorum should still send the mails. If you do not want to have Phorum send the mails also, return false.

The SMTP module is a good example of using this hook to replace Phorum's default mail sending system.

Call time:

In the file `email_functions.php` in `phorum_email_user()`, right before email is sent using mail().

Hook input:

Array with mail data (read-only) containing:

- `addresses`, an array of e-mail addresses
- `from`, the sender address
- `subject`, the mail subject
- `body`, the mail body
- `bcc`, whether to use Bcc for mailing multiple recipients

Hook output:

true or false - see description.

3.14.3 moderation

This hook can be used for logging moderator actions. You can use the `$PHORUM` array to retrieve additional info like the moderating user's id and similar.

The moderation step id is the variable `$mod_step` that is used in `moderation.php`. Please read that script to see what moderation steps are available and for what moderation actions they stand.

When checking the moderation step id for a certain step, always use the contstants that are defined for this in `include/constants.php`. The numerical value of this id can change between Phorum releases.

Call time:

At the start of `moderation.php`

Hook input:

The id of the moderation step which is run (read-only).

Hook output:

None

Example code:

```
function phorum_mod_foo_moderation ($mod_step)
{
    global $PHORUM;

    // Update the last timestamp for the moderation step
    $PHORUM["mod_foo"]["moderation_step_timestamps"][ ↩
        $mod_step] = time();
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));
}
```

3.14.4 before_delete

This hook allows modules to implement extra or different delete functionality.

The primary use of this hook would be for moving the messages to some archive-area instead of really deleting them.

Call time:

In `moderation.php`, just before deleting the message(s)

Hook input:

An array containing the following 5 parameters:

- `$delete_handled`: default = `false`, set it to true to avoid the real delete afterwards
- `$msg_ids`: an array containing all deleted message ids
- `$msgthd_id`: the msg-id or thread-id to be deleted
- `$message`: an array of the data for the message retrieved with `$msgthd_id`
- `$delete_mode`: mode of deletion, either `PHORUM_DELETE_MESSAGE` or `PHORUM_DELETE_TREE`

Hook output:

Same as input.
`$delete_handled` and `$msg_ids` are used as return data for the hook.

Example code:

```
function phorum_mod_foo_before_delete($data)
{
    global $PHORUM;

    // Store the message data in the module's settings for
    // future use.
    $PHORUM["mod_foo"]["deleted_messages"][$msgthd_id] =  ↩
        $message;
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));
```

```
    return $data;
}
```

3.14.5 delete

This hook can be used for cleaning up anything you may have created with the post_post hook or any other hook that stored data tied to messages.

Call time:
In `moderation.php`, right after deleting a message from the database.

Hook input:
An array of ids for messages that have been deleted (read-only).

Hook output:
None

Example code:

```
function phorum_mod_foo_delete($msgthd_ids)
{
    global $PHORUM;

    // Log the deleted message ids
    foreach ($msgthd_ids as $msgthd_id) {
        $PHORUM["mod_foo"]["deleted_messages"][] = $msgthd_id ↩
            ;
    }
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));
}
```

3.14.6 move_thread

This hook can be used for performing actions like sending notifications or for making log entries after moving a thread.

Call time:
In `moderation.php`, right after a thread has been moved by a moderator.

Hook input:
The id of the thread that has been moved (read-only).

Hook output:
None

Example code:

```
function phorum_mod_foo_move_thread($msgthd_id)
{
    global $PHORUM;

    // Log the moved thread id
    $PHORUM["mod_foo"]["moved_threads"][] = $msgthd_id;
```

```
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));
}
```

3.14.7 close_thread

This hook can be used for performing actions like sending notifications or making log entries after closing threads.

Call time:
In `moderation.php`, right after a thread has been closed by a moderator.

Hook input:
The id of the thread that has been closed (read-only).

Hook output:
None

Example code:

```
function phorum_mod_foo_close_thread($msgthd_id)
{
    global $PHORUM;

    // Log the closed thread id
    $PHORUM["mod_foo"]["closed_threads"][] = $msgthd_id;
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));
}
```

3.14.8 reopen_thread

This hook can be used for performing actions like sending notifications or making log entries after reopening threads.

Call time:
In `moderation.php`, right after a thread has been reopened by a moderator.

Hook input:
The id of the thread that has been reopened (read-only).

Hook output:
None

Example code:

```
function phorum_mod_foo_reopen_thread($msgthd_id)
{
    global $PHORUM;

    // Log the reopened thread id
    $PHORUM["mod_foo"]["reopened_threads"][] = $msgthd_id;
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));
}
```

3.14.9 after_approve

This hook can be used for performing extra actions after a message has been approved.

Call time:

In `moderation.php`, right approving a message and possibly its replies.

Hook input:

An array containing two elements:

- The message data
- The type of approval (either `PHORUM_APPROVE_MESSAGE` or `PHORUM_APPROVE_MESSAGE_TREE`)

Hook output:

Same as input.

Example code:

```
function phorum_mod_foo_after_approve($data)
{
    global $PHORUM;

    // alert the message author that their message has been
    // approved
    $pm_message = preg_replace(
        "%message_subject%",
        $data[0]["subject"],
        $PHORUM["DATA"]["LANG"]["mod_foo"][" ↩
            MessageApprovedBody"]
        );
    phorum_db_pm_send(
        $PHORUM["DATA"]["LANG"]["mod_foo"][" ↩
            MessageApprovedSubject"],
        $pm_message,
        $data[0]["user_id"]
        );

    return $data;

}
```

3.14.10 hide_thread

This hook can be used for performing actions like sending notifications or making log entries after hiding a message.

Call time:

In `moderation.php`, right after a message has been hidden by a moderator.

Hook input:

The id of the thread that has been hidden (read-only).

Hook output:

None

Example code:

```
function phorum_mod_foo_hide_thread($msgthd_id)
{
    global $PHORUM;

    // Log the hidden thread id
    $PHORUM["mod_foo"]["hidden_threads"][] = $msgthd_id;
    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));
}
```

3.14.11 after_merge

This hook can be used for performing actions on merging threads

Call time:

In `moderation.php`, right after two threads have been merged by a moderator.

Hook input:

An array with the translated message-ids; old-message_id -> new-message_id

Hook output:

None

3.14.12 after_split

This hook can be used for performing actions on splitting threads

Call time:

In `moderation.php`, right after a thread has been split by a moderator.

Hook input:

The id of the newly created thread

Hook output:

None

3.15 Page data handling

3.15.1 index

This hook can be used to modify the data for folders and forums that are shown on the index page.

Call time:

Just before the index page is shown.

Hook input:

An array containing all the forums and folders that will be shown on the index page.

Hook output:

The same array as the one that was used for the hook call argument, possibly with some updated fields in it.

Example code:

```
function phorum_mod_foo_index($data)
{
    global $PHORUM;

    foreach ($data as $id => $item)
    {
        if (!$item['folder_flag'])
        {
            $data[$id]['description'] .= '<sbr/>Blah foo bar  ↩
                baz';
        }
    }

    return $data;
}
```

3.16 User data handling

3.16.1 user_save

This hook can be used to handle the data that is going to be stored in the database for a user. Modules can do some last minute change on the data or keep some external system in sync with the Phorum user data.

In combination with the [??] hook, this hook could also be used to store and retrieve some of the Phorum user fields using some external system.

Call time:
Just before user data is stored in the database.

Hook input:
An array containing user data that will be sent to the database.

Hook output:
The same array as the one that was used for the hook call argument, possibly with some updated fields in it.

Example code:

```
function phorum_mod_foo_user_save($user)
{
    // Add "[A]" in front of admin user real_name fields.
    $A = $user["admin"] ? "[A]" : "";
    $real_name = preg_replace('/^\[A\]/', $A, $user[" ↩
        real_name"]);
    $user['real_name'] = $real_name;

    // Some fictional external system to keep in sync.
    include("../coolsys.php");
    coolsys_save($user);
```

```
    return $user;
}
```

3.16.2 user_register

This hook is called when a user registration is completed by setting the status for the user to PHORUM_USER_ACTIVE. This hook will not be called right after filling in the registration form (unless of course, the registration has been setup to require no verification at all in which case the user becomes active right away).

Call time:

Right after a new user registration becomes active.

Hook input:

An array containing user data for the registered user.

Hook output:

The same array as the one that was used for the hook call argument, possibly with some updated fields in it.

Example code:

```
function phorum_mod_foo_user_register($user)
{
    // Log user registrations through syslog.
    openlog("Phorum", LOG_PID | LOG_PERROR, LOG_LOCAL0);
    syslog(LOG_NOTICE, "New user registration: $user[username ↩
        ]");

    return $user;
}
```

3.16.3 user_get

This hook can be used to handle the data that was retrieved from the database for a user. Modules can add and modify the user data.

In combination with the [??] hook, this hook could also be used to store and retrieve some of the Phorum user fields in some external system

Call time:

Just after user data has been retrieved from the database.

Hook input:

This hook receives two arguments.
The first argument contains an array of users. Each item in this array is an array containing data for a single user, which can be updated.
The second argument contains a boolean that indicates whether detailed information (i.e. including group info) is retrieved.

Hook output:

The array that was used as the first argument for the hook call, possibly with some updated users in it.

Example code:

```
function phorum_mod_foo_user_get($user, $detailed)
{
    // Let's asume that our usernames are based on the
    // system users on a UNIX system. We could merge some
    // info from the password file with the Phorum info here.

    // First try to lookup the password file entry.
    // Return if this lookup fails.
    $pw = posix_getpwnam($user['username']);
    if (empty($pw)) return $user;

    // On a lot of systems, the "gecos" field contains
    // the real name for the user.
    $user['real_name'] = $pw["gecos"] != ''
                       ? $pw["gecos"]
                       : $user["real_name"];

    // If a custom profile field "shell" was created, then
    // we could also put the user's shell in the data.
    $user['shell'] = $pw['shell'];

    return $user;
}
```

3.16.4 user_list

This hook can be used for reformatting the list of users that is returned by the phorum_api_user_list() function. Reformatting could mean things like changing the sort order or modifying the fields in the user arrays.

Call time:

Each time the phorum_api_user_list() function is called. The core Phorum code calls the function for creating user drop down lists (if those are enabled in the Phorum general settings) for the group moderation interface in the control center and for sending private messages.

Hook input:

An array of user info arrays. Each user info array contains the fields "user_id", "username" and "display_name". The hook function is allowed to update the "username" and "display_name" fields.

Hook output:

The same array as was used for the hook call argument, possibly with some updated fields in it.

Example code:

```
function phorum_mod_foo_user_list($users)
```

```
{
    // Only run this hook code for authenticated users.
    if (empty($PHORUM["user"]["user_id"])) return $users;

    // Retrieve a list of buddies for the active user.
    // If there are no buddies, then no work is needed.
    $buddies = phorum_db_pm_buddy_list();
    if (empty($buddies)) return $users;

    // Flag buddies in the user list.
    $langstr = $GLOBALS["PHORUM"]["DATA"]["LANG"]["Buddy"];
    foreach ($buddies as $user_id => $info) {
        $users[$user_id]["display_name"] .= " ($langstr)";
    }

    return $users;
}
```

3.16.5 user_delete

Modules can use this hook to run some additional user cleanup tasks or or to keep some external system in sync with the Phorum user data.

Call time:
Just before a user is deleted.

Hook input:
The user_id of the user that will be deleted.

Hook output:
The same user_id as the one that was used for the hook call argument.

Example code:

```
function phorum_mod_foo_user_delete($user_id)
{
    // Get user info
    $user = phorum_api_user_get($user_id);

    // Log user delete through syslog.
    openlog("Phorum", LOG_PID | LOG_PERROR, LOG_LOCAL0);
    syslog(LOG_NOTICE, "Delete user registration: $user[ ↩
        username]");

    return $user_id;
}
```

3.16.6 before_register

This hook can be used for performing tasks before user registration. This hook is useful if you want to add some data to or change some data in the user data and to check if

the user data is correct.

When checking the registration data, the hook can set the "error" field in the returned user data array. When this field is set after running the hook, the registration processed will be halted and the error will be displayed. If you created a custom form field "foo" and you require that field to be filled in, you could create a hook function like the one in the example below.

The error must be safely HTML escaped, so if you use untrusted data in your error, then make sure that it is escaped using htmlspecialchars() to prevent XSS (see also paragraph 3.6: Secure your pages from XSS).

Call time:
In `register.php`, right before a new user is stored in the database.
Hook input:
An array containing the user data of the soon-to-be-registered user.
Hook output:
Same as input.
Example code:

```
function phorum_mod_foo_before_register ($data)
{
    $myfield = trim($data['foo']);
    if (empty($myfield)) {
        $data['error'] = 'You need to fill in the foo field';
    }

    return $data;
}
```

3.16.7 after_register

This hook can be used for performing tasks (like logging and notification) after a successful user registration.

Call time:
In `register.php`, right after a successful registration of a new user is done and all confirmation mails are sent.
Hook input:
An array containing the user data of the newly registered user (read-only).
Hook output:
None
Example code:

```
function phorum_mod_foo_after_register($data)
{
    global $PHORUM;

    // Keep a log of user registrations by user id with
```

```
    // the IP address of the computer they used to
    // register
    $PHORUM["mod_foo"]["user_registrations"][$userdata[" ↩
        user_id"]] = $_SERVER["REMOTE_ADDR"];

    phorum_db_update_settings(array("mod_foo" => $PHORUM[" ↩
        mod_foo"]));
}
```

3.17 User authentication and session handling

3.17.1 user_authenticate

This hooks gives modules a chance to handle the user authentication (for example to authenticate against an external source like an LDAP server).

Call time:

Just before Phorum runs its own user authentication.

Hook input:

An array containing the following fields:

- type: either PHORUM_FORUM_SESSION or PHORUM_ADMIN_SESSION;
- username: the username of the user to authenticate;
- password: the password of the user to authenticate;
- user_id: Always NULL on input. This field implements the authentication state.

Hook output:

The same array as the one that was used for the hook call argument, possibly with the user_id field updated. This field can be set to one of the following values by a module:

- NULL: let Phorum handle the authentication
- FALSE: the authentication credentials are rejected
- 1234: the numerical user_id of the authenticated user

Example code:

```
function phorum_mod_foo_user_authenticate($auth)
{
    // Only trust admin logins from IP addresses in  ↩
        10.1.2.0/24.
    if ($auth["type"] == PHORUM_ADMIN_SESSION) {
        if (substr($_SERVER['REMOTE_ADDR'],0,7) != '10.1.2.') ↩
            {
            $auth["user_id"] = FALSE;
            return $auth;
```

```
        }
    }

    // Let Phorum handle autentication for all users that
    // have a username starting with "bar" (not a really
    // useful feature, but it shows the use of the NULL
    // return value ;-).
    if (substr($auth["username"], 0, 3) == "bar") {
        $auth["user_id"] = NULL;
        return $auth;
    }

    // Authenticate other logins against an external source.  ↩
        Here
    // we call some made up function for checking the  ↩
        password,
    // which returns the user_id for the authenticated user.
    $user_id = some_func_that_checks_pw(
        $auth["username"],
        $auth["password"]
    );
    $auth["user_id"] = empty($user_id) ? FALSE : $user_id;
    return $auth;
}
```

3.17.2 user_session_create

Allow modules to override Phorum's session create management or to even fully omit creating a session (for example useful if the hook [??] is used to inherit an external session from some 3rd party application).

Call time:

Just before Phorum runs its own session initialization code in the user API function `phorum_api_user_session_create()`.

Hook input:

The session type for which a session must be created. This can be either `PHORUM_FORUM_SESSION` or `PHORUM_ADMIN_SESSION`.

Hook output:

Same as input if Phorum has to run its standard session initialization code or NULL if that code should be fully skipped.

Example code:

```
function phorum_mod_foo_user_session_create($type)
{
    // Let Phorum handle admin sessions on its own.
    if ($type == PHORUM_ADMIN_SESSION) return $type;

    // Override the session handling for front end forum  ↩
        sessions.
```

```
    // We could for example put the session in a standard PHP
    // session by first starting a PHP session if that was
    // not done yet...
    if (!session_id()) session_start();

    // ...and then storing the user_id of the current user in ↩
         the
    // PHP session data. The user_id is really the only thing
    // that needs to be remembered for a Phorum session,  ↩
        because
    // all other data for the user is stored in the database.
    $phorum_user_id = $GLOBALS["PHORUM"]["user"]["user_id"];
    $_SESSION['phorum_user_id'] = $phorum_user_id;

    // Tell Phorum not to run its own session initialization  ↩
        code.
    return NULL;
}
```

See the [??] hook for an example of how to let Phorum pick up this PHP based session.

3.17.3 user_session_restore

Allow modules to override Phorum's session restore management. This hook is the designated hook if you need to let Phorum inherit an authenticated session from some external system.

The array that is passed to this hook, contains a key for each of the Phorum session types:

- PHORUM_SESSION_LONG_TERM
- PHORUM_SESSION_SHORT_TERM
- PHORUM_SESSION_ADMIN

What the module has to do, is fill the values for each of these keys with the user_id of the Phorum user for which the session that the key represents should be considered active. Other options are FALSE to indicate that no session is active and NULL to tell Phorum to handle session restore on its own.

Note that the user for which a user_id is provided through this hook must exist in the Phorum system before returning from this hook. One option to take care of that constraint is letting this hook create the user on-the-fly if needed. A cleaner way would be to synchronize the user data from the main system at those times when the user data changes (create, update and delete user). Of course it is highly dependent on the other system whether you can implement that kind of Phorum user management in the main application.

Hint: Creating users can be done using the `phorum_api_user_save()` user API function.

Call time:

Just before Phorum runs its own session restore code in the user API function `phorum_api_user_session_restore()`.

Hook input:

An array containing three keys:

- PHORUM_SESSION_LONG_TERM
- PHORUM_SESSION_SHORT_TERM
- PHORUM_SESSION_ADMIN

By default, all values for these keys are NULL.

Hook output:

Same as input, possibly with updated array values.

Example code:

See the [??] hook for an example of how to let Phorum setup the PHP session that is picked up in this example hook.

```
function phorum_mod_foo_user_session_restore($sessions)
{
    // Override the session handling for front end forum sessions.
    // We could for example retrieve a session from a standard PHP
    // session by first starting a PHP session if that was
    // not done yet...
    if (!session_id()) session_start();

    // ...and then retrieving the user_id of the current user
    // from the PHP session data. The user_id is really the
    // only thing that needs to be remembered for a Phorum
    // session, because all other data for the user is stored
    // in the database. If no user id was set in the session,
    // then use FALSE to flag this to Phorum.
    $phorum_user_id = empty($_SESSION['phorum_user_id'])
                    ? FALSE : $_SESSION['phorum_user_id'];

    // If we only use session inheritance for the front end
    // forum session (highly recommended for security), then
    // We keep PHORUM_SESSION_ADMIN at NULL (default value).
    // The other two need to be updated. If the main system does
    // not use the concept of one long and one short term cookie
    // (named "tight security" by Phorum), then simply assign
    // the user_id to both PHORUM_SESSION_LONG_TERM and
    // PHORUM_SESSION_SHORT_TERM.
```

```
    $sessions[PHORUM_SESSION_SHORT_TERM] = $phorum_user_id;
    $sessions[PHORUM_SESSION_LONG_TERM] = $phorum_user_id;

    return $sessions;
}
```

3.17.4 user_session_destroy

Allow modules to override Phorum's session destroy management or to even fully omit destroying a session (for example useful if the hook [??] is used to inherit an external session from some 3rd party application).

Call time:

Just before Phorum runs its own session destroy code in the user API function `phorum_api_user_session_destroy()`.

Hook input:

The session type for which a session must be destroyed. This can be either `PHORUM_FORUM_SESSION` or `PHORUM_ADMIN_SESSION`.

Hook output:

Same as input if Phorum has to run its standard session destroy code or NULL if that code should be fully skipped.

Example code:

See the [??] hook for an example of how to let Phorum setup the PHP session that is destroyed in this example hook.

```
function phorum_mod_foo_user_session_destroy($type)
{
    // Let Phorum handle destroying of admin sessions on its own.
    if ($type == PHORUM_ADMIN_SESSION) return $type;

    // Override the session handling for front end forum sessions.
    // We could for example have stored the session in a standard
    // PHP session. First, we start a PHP session if that was
    // not done yet.
    if (!session_id()) session_start();

    // After starting the PHP session, we can clear the session
    // data for the Phorum user. In the user_session_create hook
    // example code, we stored the user_id for the active user
    // in the session. Here we clear that data. We could also
    // have destroyed the full PHP session, but in that case we
    // would risk destroying session data that was setup by
```

```
    // other PHP scripts.
    unset($_SESSION['phorum_user_id']);

    // Tell Phorum not to run its own session destroy code.
    return NULL;
}
```

3.18 Control center

3.18.1 cc_panel

This hook can be used to implement an extra control center panel or to override an existing panel if you like.

Call time:

Right before loading a standard panel's include file.

Hook input:

An array containing the following fields:

- panel: the name of the panel that has to be loaded. The module will have to check this field to see if it should handle the panel or not.
- template: the name of the template that has to be loaded. This field should be filled by the module if it wants to load a specific template.
- handled: if a module does handle the panel, then it can set this field to a true value, to prevent Phorum from running the standard panel code.
- error: modules can fill this field with an error message to show.
- okmsg: modules can fill this field with an ok message to show.

Hook output:

The same array as the one that was used for the hook call argument, possibly with the "template", "handled", "error" and "okmsg" fields updated in it.

3.18.2 cc_save_user

This hook works the same way as the [??] hook, so you can also use it for changing and checking the user data that will be saved in the database. There's one difference. If you want to check a custom field, you'll also need to check the panel which you are on, because this hook is called from multiple panels. The panel that you are on will be stored in the `panel` field of the user data.

The example hook belows demonstrates code which could be used if you have added a custom field to the template for the option `Edit My Profile` in the control panel.

Call time:

In `control.php`, right before data for a user is saved in the control panel.

Hook input:
An array containing the user data to save.

- error: modules can fill this field with an error message to show.

Hook output:
The same array as the one that was used for the hook call argument, possibly with the "error" field updated in it.

Example code:

```
function phorum_mod_foo_cc_save_user ($data)
{
    // Only check data for the panel "user".
    if ($data['panel'] != "user") return $data;

    $myfield = trim($data['your_custom_field']);
    if (empty($myfield)) {
        $data['error'] = 'You need to fill in my custom field ↩
            ';
    }

    return $data;
}
```