How to contribute

There are several ways to contribute to the FusionForge project.

• The first one is to give us a feedback about the software: what you like, what you don't like, features you like to be added in new versions, comments on documentation, etc.

• You are willing to help develop new features. Give a look at the PM/Task Manager. The three subprojects (Todo, Documentation, Localization) lists the open tasks. You can pick up one of the open tasks and start working on it.

• If you find a bug, submit a bug to the Bug tracker at http://fusionforge.org/tracker/?group_id=6. If you already solved the bug and are willing to share the fix with us, please add a patch to the bug.

• If you have developed some new feature, submit a patch to the patch manager of fusionforge. The submitted patch should follow the Coding and Templating standards described in this document.

• Documentation: there are some parts of FusionForge that are not well documented. Refer to the Task Manager, Documentation subproject for a list of open tasks. You can pick up one of the open tasks and start working on it.

How to obtain XHTML compliance for FusionForge
• Localization: The user interface of fusionforge is now nearly completely localized. We need translators for the different languages.

If you like to translate fusionforge to a new, not listed language, please contact us.

FusionForge repository

Anonymous access

You can check out the FusionForge repository from fusionforge.org.

The module you wish to check out must be specified as the <modulename>.

TO BE COMPLETED

Modules in repository

The following modules are available in FusionForge repository.

gforge Main module. Contains FusionForge software.
tools Contains tools which may be useful for FusionForge developers, translators and administrators.

PHP Coding Standards

Introduction

Coding Standards. Live them, love them.

Then come up with a new introduction...

Comments

Guidelines

Non-documentation comments are strongly encouraged. A general rule of thumb is that if you look at a section of code and think "Wow, I don't want to try and describe that", you need to comment it before you forget how it works.

• C++ style comments (/* */) and standard C comments (//) are both acceptable.
• Use of perl/shell style comments (#) is prohibited.

PHPdoc Tags

Inline documentation for classes should follow the PHPDoc convention, similar to Javadoc. More information about PHPDoc can be found here: http://www.phpdoc.de/

File comments

Every file should start with a comment block describing its purpose, version, author and a copyright message. The comment block should be a block comment in standard JavaDoc format along with a CVS Id tag. While all JavaDoc tags are allowed, only the tags in the examples below will be parsed by PHPdoc.
FusionForge contains a mixed copyright. For files that have been changed since the FusionForge fork, the following header should be used:

```c
/**
 * brief description.
 * long description. more long description.
 * Portions Copyright 1999-2001 (c) VA Linux Systems
 * The rest Copyright 2002 (c) their respective authors
 */
```

**Function and Class Comments**

Similarly, every function should have a block comment specifying name, parameters, return values, and last change date.

```c
/**
 * brief description.
 * long description. more long description.
 * @author    firstname lastname email
 * @param     variable  description
 * @return    value     description
 * @date      YYYY-MM-DD
 * @deprecated
 * @see
 */
```

**Note**

The placement of periods in the short and long descriptions is important to the PHPdoc parser. The first period always ends the short description. All future periods are part of the long description, ending with a blank comment line. The long comment is optional.

**Formatting**

**Indenting**

All indenting is done with TABS. Before committing any file to Subversion, make sure you first replace spaces with tabs and verify the formatting.

**PHP Tags**

The use of `<?php ?>` to delimit PHP code is required. Using `<? ?>` is not valid. This is the most portable way to include PHP code on differing operating systems and webserver setups. Also, XML parsers are confused by the shorthand syntax.
Templating

In the FusionForge system, PHP itself is used as the template language. To make the templating clearer, template files should be separated out and included once objects and database results are established. Detailed examples are in the docs repository and here.

Variables in the templates are presented surrounded by `<?php ?>` tags instead of the `{ }` tags that some other template libraries would use. The end result is the same, with less bloat and more efficient code.

Expressions

- Use parentheses liberally to resolve ambiguity.
- Using parentheses can force an order of evaluation. This saves the time a reader may spend remembering precedence of operators.
- Don't sacrifice clarity for cleverness.
- Write conditional expressions so that they read naturally aloud.
- Sometimes eliminating a not operator (!) will make an expression more understandable.
- Keep each line simple.
- The ternary operator (`x ? 1 : 2`) usually indicates too much code on one line. `if... else if... else` is usually more readable.

Functions

Function Calls

Functions shall be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

```php
$var = foo($bar, $baz, $quux);
```

As displayed above, there should be one space on either side of an equals sign used to assign the return value of a function to a variable. In the case of a block of related assignments, more space may be inserted to promote readability:

```php
$short  = foo($bar);
$long_variable = foo($baz);
```

Function Definitions

Function declarations follow the unix convention:

```php
function fooFunction($arg1, $arg2 = '') {
    if (condition) {
        statement;
```
Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate. Here is a slightly longer example:

```php
function connect(&$dsn, $persistent = false) {
    if (is_array($dsn)) {
        $dsninfo = &$dsn;
    } else {
        $dsninfo = DB::parseDSN($dsn);
    }

    if (!$dsninfo || !$dsninfo['phptype']) {
        return $this->raiseError();
    }

    return true;
}
```

**Objects**

Objects should generally be normalized similar to a database so they contain only the attributes that make sense.

Each object should have `Error` as the abstract parent object unless the object or its subclasses will never produce errors.

Each object should also have a `create()` method which does the work of inserting a new row into the database table that this object represents.

An `update()` method is also required for any objects that can be changed. Individual `set()` methods are generally not a good idea as doing separate updates to each field in the database is a performance bottleneck.

`fetchData()` and `getId()` are also standard in most objects. See the tracker codebase for specific examples.

Common sense about performance should be used when designing objects.

**Naming**

- Constants should always be uppercase, with underscores to separate words. Prefix constant names with the name of the class/package they are used in. For example, the constants used by the `DB::` package all begin with “DB_”.

- True and false are built in to the php language and behave like constants, but should be written in lowercase to distinguish them from user-defined constants.

- Function names should suggest an action or verb: `updateAddress, makeStateSelector`

- Variable names should suggest a property or noun: `UserName, Width`
• Use pronounceable names. Common abbreviations are acceptable as long as they are used the same way throughout the project.

• Be consistent, use parallelism. If you are abbreviating “number” as “num”, always use that abbreviation. Don’t switch to using “no” or “nmb”.

• Use descriptive names for variables used globally, use short names for variables used locally.

$AddressInfo = array(...);
for($i=0; $i < count($list); $i++)

**Control Structures**

These include if, for, while, switch, etc. Here is an example if statement, since it is the most complicated form:

```php
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    defaultaction;
}
```

Control statements shall have one space between the control keyword and opening parenthesis, to distinguish them from function calls.

You should use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

For switch statements:

```php
switch (condition) {
    case 1: {
        action1;
        break;
    }
    case 2: {
        action2;
        break;
    }
    default: {
        defaultaction;
        break;
    }
}
```

**Including PHP Files**

Anywhere you are unconditionally including a class file, use require_once. Anywhere you are conditionally including a class file (for example, factory methods), use include_once. Either of these will
ensure that class files are included only once. They share the same file list, so you don't need to worry
about mixing them - a file included with require_once will not be included again by include_once.

Note

Note: include_once and require_once are statements, not functions. You don't need parentheses
around the filename to be included, however you should do it anyway and use ' (apostrophes)
not " (quotes):

include('pre.php');

Templating Standards

Coding Example

The following code examples demonstrate how all coding on FusionForge is going to be done in the future.
The first example shows the “switchbox” page (taken from www/tracker/index.php) - where the
various objects are included, instantiated and checked for errors every step of the way.

Once the objects are instantiated, the template file can be included. In this example, the template file is
detail.php (example2).

Template page

```php
<?php
/**
 * FusionForge Tracker Facility
 *
 * Portions Copyright 1999-2000 (c) The SourceForge Crew
 * Copyright 2002-2004 (c) The FusionForge Team
 * http://fusionforge.org/
 *
 * This file is part of FusionForge. FusionForge is free software;
 * you can redistribute it and/or modify it under the terms of the
 * GNU General Public License as published by the Free Software
 * Foundation; either version 2 of the Licence, or (at your option)
 * any later version.
 *
 * FusionForge is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * with FusionForge; if not, write to the Free Software Foundation,
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
 */

echo $ath->header(array
    ('title'=>>'Detail: '.$ah->getID(). ' '.$ah->getSummary()));

?>

```
<h3>DO NOT enter passwords or confidential information in your message!</h3>

```php
if (!session_loggedin()) {
    Email: &nbsp;
    <input type="text" name="user_email" size="20" maxlength="40">
}
```

```
<input type="submit" name="SUBMIT" value="Monitor">
```

```
Date:<br>
<?php echo date( $sys_datefmt, $ah->getOpenDate() ); ?>
```

```
Submitted By:<br>
<?php echo $ah->getSubmittedRealName(); ?>
```

```
Assigned To:<br>
<?php echo $ah->getAssignedRealName(); ?>
```

```
Category:<br>
<?php echo $ah->getCategoryName(); ?>
```

```
Status:<br>
<?php echo $ah->getStatusName(); ?>
```

```php
if (!session_loggedin()) {
    Email: &nbsp;
    <input type="text" name="user_email" size="20" maxlength="40">
}
```

```
<input type="submit" name="SUBMIT" value="Monitor">
```
Documentation

We now use XML Docbook to write documentation. You can read the Docbook Definitive Guide [http://www.docbook.org/tdg/index.html] online if you want more information about XML Docbook.

You can use the Serna XML editor [http://www.syntext.com/downloads/serna-free/] to edit the documents.

Documentation is generated by Docbook XSL stylesheets [http://sourceforge.net/projects/docbook/] (html output) and DB2Latex XSL stylesheets [http://db2latex.sourceforge.net/] (PDF output).

How to obtain XHTML compliance for FusionForge

The complete XHTML specification is available at XHTML specification [http://www.w3.org/TR/xhtml1/] at www.w3c.org [http://www.w3.org].

Here is listed a summary of what is needed to be XHTML compliant:

XML declaration All pages should have the following xml declaration:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

Document must be well-formed All open tags must be closed, all tags must stay between `<` and `>`. All tags must be lowercase `<html>` must be converted to `<html>`. Empty tags must be closed No standalone `<br>` tag is allowed; `<br/>` must be used. Attributes must always be quoted `<td rowspan=3>` must be converted to `<rowspan="3">`. The differences listed here are the most significant differences between HTML and XHTML, there are other, minor differences. For a complete list, refer to the XHTML specification.