|  |  |
| --- | --- |
| Logo_FPT_University_doc |  |

|  |
| --- |
| **FPT UNIVERSITY** |
| Capstone Project Document |

USEFUL JAPANESE DICTIONARY FOR VIETNAMESE

Coding convention

|  |  |
| --- | --- |
| USEFUL JAPANESE DICTIONARY FOR VIETNAMESE | |
| **Group Members** | |  |  | | --- | --- | | Lê Đình Nam | SE02316 | | Phạm Thị Minh | SE02300 | | Nguyễn Ngọc Tuấn | SE02189 | | Phạm Tiến Đạt | SE02336 | |
| **Supervisor** | Mr. Nguyễn Văn Sang |
| **Project code** | UJD\_VN |

- Hanoi, 06/2014 –

**Code Source**

Do not use any libraries or code from external sources without prior approval:

1) Similar code may already exist

2) Licensing may be an issue

3) Legibility and reusability can be issued

4) Performance concerns

**Conventions**

Indenting:

Use an indent of 4 spaces, with no tabs. Please indent your code.

**Case**

* Use "lowerCamelCase" style (lowercase lettering on initial words and capitalization on subsequent words) to name functions, methods, and variables

**Variables**

* Case should never be used to differentiate between variable names. Every variable name in the current scope should be absolutely unique. Variable names should describe the content that they (will) contain, using either complete words or understandable abbreviations.

**Class Definitions**

Class declarations have their opening brace on a new line:

<?php  
class Foo\_Bar  
{  
  
    //... code goes here  
  
}  
?>

# Function Definitions

# Function declarations follow the "K&R style":

<?php  
function fooFunction($arg1, $arg2 = '')  
{  
    if (condition) {  
        statement;  
    }  
    return $val;  
}  
?>

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;  
}  
?>

**Function Calls**

Functions should 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);  
?>

To support readability, parameters in subsequent calls to the same function/method may be aligned by parameter name:

<?php  
  
$this->callSomeFunction('param1',     'second',        true);  
$this->callSomeFunction('parameter2', 'third',         false);  
$this->callSomeFunction('3',          'verrrrrrylong', true);  
?>

**Arrays**

Assignments in arrays may be aligned. When splitting array definitions onto several lines, the last value may also have a trailing comma. This is valid PHP syntax and helps to keep code diffs minimal:

<?php  
  
$some\_array = array(  
    'foo'  => 'bar',  
    'spam' => 'ham',  
);  
?>

**PHP Code Tags**

**Always** use <?php ?> to delimit PHP code, not the <? ?> shorthand. This is required for PEAR compliance and is also the most portable way to include PHP code on differing operating systems and setups.

**HTML**

HTML Code should not be found anywhere in our code, other than the smarty templates. Mark places where you find it with a TODO, as well as any plain text outputted to screen. We will convert these to a multiple language format in next version.

**Control Structures**

* Place a single space between the control keyword (if, for, while, switch, etc.) and opening parenthesis to distinguish control statements from function calls
* Always use curly braces—even when technically optional (i.e., avoid PHP's alternative syntax for control structures, except the ternary operator noted below)
* Include break; after all switch case statements
* Use the Allman/BSD style for indentation and layout (braces appear alone and surrounding the indented code)

*Example if statement:*

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

*Example switch statement:*

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

*Or :*

<?php  
  
if (($condition1  
    || $condition2)  
    && $condition3  
    && $condition4  
) {  
    //code here  
}  
?>

**Ternary Operators**

Can be used for simple true/false/action checks.

($result)? echo "<result>success</result>":echo "<result>failure</result>" ;

**Database Tables and Fields**

* Lowercase words separated by underscores:
* Items
* item\_specifications
* Do NOT make calls directly to native database functions (ie: mysql\_insert\_id, mysql\_result,etc). Use ONLY adodb functions.

**Smarty plugins**

* function.smarty\_plugin.php

**Directories, File Structure**

* Do not create new directories or class, library files without first discussing

with team. Most functions,classes belong in existing files.

* New plugins, and files should follow existing layout and naming.

**Security**

**Passwords**

* Never hardcode passwords into any script
* Never put live data passwords into SVN
* Always get your database passwords from included csp config.

Configuration

* Never hard code paths to any file – use configuration files.

**Database**

* Do not distribute database information. (via email, download or otherwise)
* Do not connect to production server from home (only dev).
* Do not run test or live queries on production data.

**User Data**

* All Credit Card information should be encrypted.
* User data should be obfuscated before being used in development environments.

**Code Documentation**

Follow PHPDOC.org documentation style.

**Subversion**

Include comments for all commits to SVN. If you are fixing a bug, included the TICKET

ID in the Subversion Comments

**TODO**

Anytime you need to make comments regarding future modifications or requirement put a //TODO: comments. These will show up in PHPEclipse and give us an idea of things left unfinished (but functional).

*Example:*

// TODO: This code should be re-examined for increased performance

**CODE CLEANUP**

Where you see standard rules broken -> MARK CODE WITH A TODO

**PHP Comments**

Use C-style (/\* \*/) comments for multi-line statements and C++ comments (//) for single-line statements.

Use <!-- --> in html mode.

Header file comment :

<?php

/\*\*

\* Short description for file

\*

\* Long description for file (if any)...

\*

\* @category CategoryName

\* @package PackageName

\* @author Original Author <author@example.com>

\* @author Another Author <another@example.com>

\* @copyright 1997-2005 The PHP Group

\* @license http://www.php.net/license/3\_0.txt PHP License 3.0

\* @version CVS: $Id:$

\* @link http://pear.php.net/package/PackageName

\* @see NetOther, Net\_Sample::Net\_Sample()

\* @since File available since Release 1.2.0

\* @deprecated File deprecated in Release 2.0.0

\*/

/\*

\* Place includes, constant defines and $\_GLOBAL settings here.

\* Make sure they have appropriate docblocks to avoid phpDocumentor

\* construing they are documented by the page-level docblock.

\*/

/\*\*

\* Short description for class

\*

\* Long description for class (if any)...

\*

\* @category CategoryName

\* @package PackageName

\* @author Original Author <author@example.com>

\* @author Another Author <another@example.com>

\* @copyright 1997-2005 The PHP Group

\* @license http://www.php.net/license/3\_0.txt PHP License 3.0

\* @version Release: @package\_version@

\* @link http://pear.php.net/package/PackageName

\* @see NetOther, Net\_Sample::Net\_Sample()

\* @since Class available since Release 1.2.0

\* @deprecated Class deprecated in Release 2.0.0

\*/

class foo

{

}

?>

*Database conventions*

* Tables will always be named singular names for most tables, except when dealing with a clear collection. Thus the foreign key relationship will hold true to table\_id form
* Tables will be grouped by prefixes of primary related tables (itemProperties related to table item)
* Additional prefix grouping will be used for tables related to a particular functionality - "workspace", "sys" - this is only because there isn't a parent table, but the tables are in fact related, but by functionality.
* Primary keys of all tables will be lowercase table\_id or write off name table\_id, except in the case of composite keys (those will depend on the combination)
* Foreign keys will be table\_id (so if primary table is "site", and its primary key is "table\_id", when "table\_id" is referenced in other tables, they will reference "site\_id".
* Attribute of table always set name start with name table\_ or write off name table\_ (example :u\_id ( id of user), contact\_id (id of contact)).

● With few exceptions, the standards we are following are documented here:

*http://weblogs.asp.net/jamauss/articles/DatabaseNamingConventions.aspx* Database Naming Conventions