OpenEyes - Coding Style

Editors: G W Aylward
Version: 0.92:
Date issued: 3 December 2010

Target Audience

General Interest	
Healthcare managers	
Ophthalmologists	✔
Developers	✔

Amendment Record

Issue	Description	Author	Date
0.9	Draft	G W Aylward	9 July 2010
0.91	Draft	G W Aylward	9 October 2010
0.92	Typos corrected	G W Aylward	9 October 2010

Table of Contents

Introduction

For any software project, large or small, following a consistent programming style can help programmers read and understand source code, and help to avoid errors. This is particularly true for collaborative projects such as OpenEyes with many programmers involved who may be geographically separated. In the same way, having a logical naming convention can save a great deal of time, for example by obviating the need to check variable or function names. This document sets out a programming style and naming convention suggested for OpenEyes developers.

File names

OpenEyes consists of a large body of code written in at least three languages (PHP, javascript, and HTML) with links to back end SQL databases. This means that there are large numbers of files, variables, methods, table and other names which require a consistent naming convention. The following section describe the convention for file names

PHP scripts

These files should be named using camel case, with a descriptive name, and the .php file extensions. For example, a file which allows editing of drugs might be called ‘editDrugs.php’.

PHP classes

These files should be named using Pascal case with the prefix ‘OE’ (This identifies the file as an OpenEyes class as well as avoiding namespace clashes. Class files tend to be included in other scripts, so should all finish with a double file ending (.inc.php). The name of the class should be descriptive and include the word ‘Class’ to indicate its function. For example, putting that all together for a class which handles prescription events, the file would be named ‘OEPrescriptionEventClass.inc.php’. If a class is supplied with an interface in a separate file, this should be named in exactly the same way but with ‘Interface’ instead of ‘Class’. For example, ‘OEPrescriptionEventInterface.inc.php’.

CSS files

These files should be named using camel case, with a descriptive name, and the .css file extension. Where possible those files that are associated with an OpenEyes event class should have the same descriptive name. For example a CSS file associated with the event described in the previous section would be named ‘prescription.css’.

Javascript files

These files should be named using camel case, with a descriptive name, and the .js file extension. Where possible those files that are associated with an OpenEyes event class should have the same descriptive name. For example a javascript file associated with the event described in the previous section would be named ‘prescription.js’.

Other files

All other files should be named using camel case and the appropriate file extension (.gif, .jpg etc).

SQL Table names

Tables should be named in lower case, be descriptive, and be plural. If one word is insufficient, then underscore-separated compounds should be used. However, ideally this should be confined for intermediate tables, so that for example, the table ‘patients_contacts’ describes a many-to-many relationship between the patients table and the contacts table.

The primary key is always the singular expression of the table name, with the addition of ‘_id’, so the primary key for the patients table would be ‘patient_id’, and for the patients_contacts table would be ‘patient_contact_id’.

Like table names, field names are also written as underscore-separated compounds. This helps to distinguish them from variable names in PHP code.

Code files

Code should be written with sufficient white space and comments to make the code legible and easy to follow. Control and flow should be indicated by indentation of the text, and squiggly brackets surrounding code segments should be alone on a line, which gives a clearer layout. These points are illustrated by the code in the following example;

```
// Defaults are from supplied parameters
if ($lastName)
{
	$defaults = array('last_name' => $lastName);
	if ($type && $type != "NULL")
	{
		$defaults['type'] = $type;
	}
}
else
{
	$defaults = NULL;
}

// Validations
$validations = array('last_name' => "required");
```

HTML layout

The final output of OpenEyes code is HTML read and interpreted by the browser. Examining the raw HTML can be very helpful in the debugging process, so efforts should be made to ensure it is easily read by a human. In practical terms this means that the HTML should usually have one element per line, and that child elements

should be indented to indicate the hierarchical relationships. For example, the following is a section of the output of the login page.

```
<div id="screen">
    <div id="header">
        <img id="icon" src="graphics/OpenEyesIcon.jpg">
        <h1>OpenEyes</h1><h2 id="username">Login Page</h2>
    </div>
    <div id="admin_nav" style="height: 1px">
    </div>
    <div id="messages">
        <p class="alert">Please enter your username and password</p>
    </div>
    <div id="main">
        <form name="oeform" action="login.php" method="post">
            <p class="legend" style="width: 3.75em">Login:</p>
            <fieldset id="editform"><br />
                <label>User name:</label>
                <input type="text" class="text" name="uid" value="" />
                <label>Password:</label>
                <input type="password" class="text" name="password"  />
                <input type="submit" class="formbutton" name="submit" />
                <input type="hidden" name="submitted" value="TRUE" />
            </fieldset>
        </form>
        <script language="JavaScript" type="text/javascript">
            var formValidator = new FormValidator("oeform");
            formValidator.addValidation("uid", "required");
            formValidator.addValidation("uid", "alphanumeric");
        </script>
    </div>
    <div id="footer">
        <p>&copy; 2010 OpenEyes</p>
    </div>
</div>
</body>
</html>
```

Often the HTML is produced by PHP code using echo or print statements, and there is then a potential conflict between having a readable layout of code, and a readable layout of the resulting HTML. This conflict can be resolved by judicious use of white space, as in the following example. An editor which uses colour to distinguish code types can be very useful. In the following diagram, the flow of the PHP code (in blue) can be readily appreciated, along with the indented layout of the HTML which is sent to the browser (in green);

```
617         // Right posterior segment
618         echo '
619                 <div class="leftcolumn">
620                     <h4>Right posterior segment:</h4>
621                     <select class="clinSelect" id="RPSSelect" onchange="addToTextarea(\'RPSSelect\', \'right_ps\')">
622                         <option>Findings</option>';
623         foreach ($phraseArray as $phrase) if ($phrase->part == "Postseg")
624         {
625             echo '
626                         <option>'.$phrase->phrase.'</option>';
627         }
628         echo '
629                     </select>
630                     <br />
631                     <textarea name="right_ps" id="right_ps" onkeydown="adjustHeight(\'right_ps\', 40);">'.$examination-
632         $this->echoEyeDraw('edit', 'RPS', $drawings->RPSDataString, 'right_ps');
633         echo '
634                 </div>';
635
```

File headers

Every OpenEyes file should have a header which consists of the following information. The header should be in the correct format for PHPDocumentor and consist of the following

- Name of the file
- A short description
- The mode (Login, admin, or patient, or a combination)
- Parameters (GET or POST)
- Author name and email
- Version
- Modification date
- Copyright notice
- Package

The following file header shows an example header for a small script file

```
/**
 * viewContacts.php
 *
 * Displays a table of contacts for the current patient.
 *
 * <b>Mode:</b> patient
 *
 * OpenEyes is licensed by Moorfields Eye Hospital NHS Foundation Trust (the “Licen-
sor”) under version 3 of
 * the GNU General Public Licence (GPL), and version 1 of the Open Eyes Proprietary
Licence (OEPL).
 *
 * You can choose the licence that most suits your intended use of OpenEyes. If you
wish to contribute to the OpenEyes open source project
 * or incorporate OpenEyes into your own open source project, version 3 of the GNU
General Public Licence or any later version shall apply.
 * If you wish to use OpenEyes for commercial purposes, the terms of the OpenEyes
```

```
Proprietary Licence shall apply;
 *
 * A plain text version of the OpenEyes Proprietary Licence is distributed with this soft-
ware. The Licensor reserves the right to publish
 * revised and/or new versions of the OpenEyes Proprietary Licence from time to time.
Each version will be given a distinguishing version number.
 *
 * When using OpenEyes in your commercial application, or open source application
you are required to distribute your chosen
 * licence (GPLv3 or OEPLv1) with your application and ensure the following acknow-
ledgement is contained within both the program
 * and in any user's manual.
 *
 *            "This software uses elements of OpenEyes open source software (see
http://www.openeyes.org.uk) Open Eyes is used and may only be
 *            used under the [GPL/OEPL] version [insert version number]"
 *
 * @author Bill Aylward <bill.aylward@mac.com>
 * @license http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html LGPLv2.1
 * @license http://www.openeyes.org.uk/licenses/oepl-1.0.html OEPLv1.0

 * @version 1.0
 * Modification date: 9th October 2010
 * @copyright Copyright (c) 2010 OpenEyes
 * @package Clinical
 */
```

PHPDocumentor does not seem to provide tags for GET or POST variables. However, the following syntax produced acceptable results in the documentation output.

```
/**
 * removeContact.php
 *
 * Called by viewContacts to remove the relationship
 *
 * <b>GET Parameters:</b><br>
 * string <var>id</var> ID (mandatory)<br>
 *
 * <b>POST Parameters:</b><br>
 * string <var>id</var> ID (mandatory)<br>
 *
 * <b>Mode:</b> admin, patient
```

Variable names

Variables should be descriptive, and written in camel case, for example '$errorMessage'. Variables received as arguments to a function should begin with an underscore, for example '$_permission'

PHPDocumentor

PHPDocumentor is the current standard auto-documentation tool for the PHP language. It is a free software tool that can create documentation in a variety of formats for both developers and users.

Currently, OpenEyes makes use of the software to produce documentation directly from the source files in the htdocs directory. A directory called documentation contains the output of the software, and because this is dependent entirely on changes in the source code, the directory is excluded from Git updates by the .gitignore file.

Installation instructions are included in Appendix 1, and more information is available on the website.

In order to update your documentation after a commit or after a Git pull, the following should be run from the command line;

```
/usr/bin/phpdoc -o HTML:Smarty:HandS -d path_to_served/OpenEyes/
htdocs -t path_to_served/docs -dn OpenEyes
```

The documentation can then be read by pointing a browser to documentation/index.html. Any errors are listed in documentation/errors.html.

OpenEyes files are categorised into ‘packages’ which are indicated by PHPDocumentor tags as described in the following table;

Package name	Description
Base	Base classes and fundamental files
Enumeration	Classes that simulate enums
HTML	Classes that produce HTML objects
Links	Classes that enable links to other systems
Clinical - Clinical events	Clinical pages or events
Admin - Admin events	Administration pages or events

Glossary

The following is a list of definitions of terms and concepts used throughout this document

Term	Definition
Camel case	This is the practice of writing compound words without spaces, but with the initial letter of each word (apart from the first) starting with a capital letter. For example “camelCase”
Pascal Case	Same as Camel case, but with the first word also starting with a capital. For example “PascalCase”
Underscore-separated compounds	The use of the underscore character to separate words in compound names, for example “underscore_separated”

Appendix 1

Installation and configuration of PHPDocumentor

1. Make sure you have PHP and pear installed

Pear should have been installed along with the PHP installation.

2. Set the installation destination

Put the software in a directory served by your webserver software. The following commands replace 'path_to_served' with the actual path to your served directory.

```
pear config-set data_dir path_to_served/pear
```

3. Install

```
sudo pear upgrade PhpDocumentor
```

4. Change logo (optional)

To change the logo at the top left of the template to an OpenEyes logo, do the following;

```
cd path_to_served/pear/PhpDocumentor/phpDocumentor/Converters/HTML/Smarty/templates/HandS/templates/media
sudo mv logo.png OLDlogo.png
sudo cp path_to_served/openeyes/htdocs/graphics/logo.png logo.png
```