Web Development Naming Conventions

Author: Erik Giberti
Date: September 28, 2006
Revision: Draft 1.0

Summary: A document covering topics regarding variable, filename, database and directory naming and style conventions in web application development. Includes a brief discussion of commenting styles and common practices for creating maintainable code.

Table of Contents

Background and Overview

This document grew out of increased frustration in the realm of advice on how to go about effectively create websites using assorted naming and style conventions. Each of the documents reviewed failed to consider alternative methods or address pros and cons of each approach. Ultimately, a blend of approaches will likely yield the most productive use of a coding practices document. Each company should certainly tailor a document for coding styles based on the needs of the organization and the platform utilized. While this survey of topics is hardly exhaustive, it provides a framework for larger questions as well as provides resources for getting more information on individual topics.

The document is broken down by subject matter to facilitate scanning. When possible multiple viewpoints are included along with which development environments suggest use of a particular methodology. The goal of this document was to create a framework that is usable for ColdFusion and PHP development at AF-Design and may be extended to include other forms of development, however, that was not the focus of this document and it's purpose may end with web development.

This document, while static in its distributable form, provides the information that you can use to create or modify your organizations code practices documentation. The most important part of any code or style document should be consistency, not the minutia of how to place brackets, semi-colons or method names. Hopefully after reading this document you will be able to address concerns within your organizations code style or lack there of.

Lastly, what you will *not* get from this document is a dialog on the strengths of a particular development platform/language/environment vs. another. The author has experience with ColdFusion, Perl, PHP, ActionScript, T-SQL, JavaScript, C++ and C# and so the examples trend in that direction. This is in no way a slight of Visual Basic, Java, Ruby or any other language, but simply a reflection of the author's experience. This is also not a diatribe about the pros or cons of a particular methodology such as Model View Controller, Rails, UML or any other method for developing applications.

Commenting

Ask any developer what they wish their predecessor did better and undoubtedly they will quickly create a list of issues with their coding style and lack of comments. Each language has it's own syntax specific parts to defining a comment block such as `<!-- -->` in HTML, `--` in SQL and `/* */` and `//` in ECMA languages. Code comments can be broken down into three primary types[1] each with specific rolls in your application. The Class/Page Comments provide a context to understand the file that's being worked on. Inline Comments provide context about the code right where other developers need it. The last method, Function Commenting, provides not only background on a particular method, but also creates a nice framework of documentation for future reuse of the code.

[1] http://www.particletree.com/features/successful-strategies-for-commenting-code/

Documentation is a separate discussion, but it should be noted that a few different systems of commenting could actually help you create dynamic documentation of files. Projects including Javadoc[2] for Java, PHPdoc[3] for PHP and Sandcastle[4] for .NET create documentation that is either HTML based or flat files that can be distributed to developers to reduce development time by providing documentation for the code in readable format. Each of these tools have specific requirements on how comments must be structured which I will not go into here. It suffices to say that adoption of a framework from any vendor requires training and dedication to that framework. The suggestions that follow are not specific to any framework, but will provide information that will speed future revisions and modifications.

Class/Page Comments

This is perhaps the quickest to implement since it can be added to a default document template easily in most authoring environments and provides much of the detail a later developer (or possibly even you) will need when this document is opened years later. Some essential elements of this initial comment block are included here as ColdFusion code – note the similarity to an engineering drawing's title block:

```
<!---
      Title: A Sample Comment Block
      Project: AF-Design Code Samples
      Author: Erik Giberti

      Description: A sample page comment / title block

      Created: September 23, 2006
      Revision History:
            Sept. 26, 2006 : Erik Giberti – Changed font face
             in document for code samples
--->
```

Some additional information you might find in a title block comment include licensing information, any copyright notices that are required, dependent files, file location, project group information and the like. There is really no limit to what can or should be included.

Warning: If the Class/Page Comments are in a true HTML, XML or JavaScript document, it is exposed to anyone who cares to look at the source for it and this could compromise your sites security. Always keep your sites security in mind when adding these comments within documents that are served directly to your users. It could be worth adding a process to remove this information on production servers. Most application environments, PHP, ColdFusion, Perl and .NET included, comments are suppressed during compilation instead of being displayed and so it is preferred to place comments within the application language instead of text that is served to you users.

Inline Commenting

[2] http://java.sun.com/j2se/javadoc/
[3] http://www.phpdoc.de/
[4] http://www.microsoft.com/downloads/details.aspx?FamilyID=E82EA71D-DA89-42EE-A715-696E3A4873B2&displaylang=en

This is often the most common and most underused commenting since it provides help around a particular process or piece of code that might be confusing later. A simple example of this to explain what each variable within a piece of code is holding. Of course your always going to use good variable names but this removes any doubt as to the purpose of a variable. Again, an example in ColdFusion:

```
<cfset AdCounter = 0> <!--- Used for counting ad clicks --->
```

or

```
<!--- For counting the number of unique visitors --->
<cfset VisitorCounter = 0>
```

Inline commenting can also clarify information that is being processed close to where it is actually being done so that other developers can see the logic behind a particular approach. This example, using our variables from above:

```
<table>
<tr>
<td>
<!--- display the click through rate for our ads --->
<cfoutput>#numberFormat(evaluate(AdCounter/VisitorCounter),“0.00”)#%</cfoutput>
</td>
</tr>
</table>
```

As you can see, the comment provides context as to what this number actually is because it is unclear from the surrounding code.

Function Commenting

Similar in practice to Class/Page Commenting, the comment can provide a quick overview of what a particular method does. In the following example, a JavaScript function is commented only using Function Commenting, notice it is still possible to discern what the file does without digging into the code – which clearly seems to be missing some of the functionality to actually add these vectors.

```
/*
      Summary: Adds to Vector values together, also see Vector class
      Parameters: VectorA and VectorB (both Vector objects)
      Return: Vector object
*/
function AddVectors(VectorA, VectorB)
{
      var VectorResult = new Vector();
      VectorResult.Direction = VectorA.Direction + VectorA.Direction;
      VectorResult.Magnitude = VectorA.Magnitude + VectorB.Magnitude;
      return VectorResult;
}
```

The resulting Vector object contains the values added together – this would actually result in a wrong value being passed, but someone can look at the code and see that this is the place where it needs to be corrected.

Standardized Comment Hinting

Using some common text in all capital letters can help call out certain pieces of content within your comments that are important to someone reading your code. In addition to the revision history at the top of the document, these can help call out other items inline.

1. KLUGE or HACK: Acknowledges where inelegant code was used as a work around some other problem.
2. BUG: Identifies the location of a problematic piece of code.
3. TODO: Denotes unfinished pieces of work (possibly a method stub that needs tweaking)
4. FIX: Marks where something was corrected. Often the bug # would be included in this line.

Variable Namespace

Namespaces are essentially containers that will hold an applications environment separate from other environments that could potentially conflict with your variables. While this can largely be ignored in most small web application environments, in some environments Java, C++ and .NET for example, it's a good practice and is easy to implement. For instance, AF-Design should, whenever possible, use the namespace AFDesign to eliminate any conflict between my class files (regardless of language) and the core libraries of the language. A few examples of declaring a namespace follow:

C++[5]:

```
namespace AFDesign
{
	int bar;
}

using namespace AFDesign;
```

XML[6]:

```
<html xmlns="http://www.w3.org/1999/xhtml" />
```

A second way to create a unique namespace in languages that do not formally support namespaces is to prefix each variable with a namespace declaration for example in JavaScript AF-Design might use:

```
var AFDesign_MyVariableName = null;
```

Note in this example AF-Design is actually a poor name because the hyphen would be evaluated as an operator and so must be omitted. This could cause confusion with a company without the hyphen in the name.

Variable Names, Capitalization and Conventions

[5] http://en.wikipedia.org/wiki/Namespace_%28computer_science%29
[6] http://en.wikipedia.org/wiki/XML_Namespace

A few primary methods for notation of multiple word files exist as standards currently. Microsoft has provided guidance to .NET developers through a naming conventions document in the MSDN library[7]. Sun provides guidance for developing in Java[8] that also provides a good framework for other ECMA based languages.

1. Camel (or Camelback) [`myVariableName`]
    a. First letter lowercase with each subsequent words first letter capitalized.
    b. Standard method for Java variables
2. Pascal [`MyVariableName`]
    a. Each words first letter is capitalized, including the first word.
    b. Standard method for .NET variables
3. Uppercase [`MYVARIABLENAME`]
    a. All letters are capitalized
    b. Only readable for short 1-3 character variables
    c. Java recommends using all caps for declared constants
4. Lowercase [`myvariablename`]
    a. All letters are lowercase
    b. Only readable for one word variables (like Camel with a singular word)
5. Underscores
    a. Using underscore character like a space [`MY_VARIABLE_NAME`]
    b. Special cases using underscore as first value to protect namespace [`_variable`]

Regardless of the style chosen, it is most important for the variable to describe what it contains above all else. For the purpose of your company, if you are using Microsoft .NET as a platform, you should fully embrace their suggestions surrounding Camel or Pascal cases.

Code Blocks

In a language that supports code blocks using brackets, it is always better to use the brackets for clarity than to eliminate them. There are two ways to use the brackets when writing code blocks:

```
if( some statement)
{
      code block;
}
```

and

```
if ( some statement ){
      code block;
}
```

while the author prefers to use the later – Java and ECMA style, if the objective of a coding style is clarity, the first method is clearer when looking at larger nested blocks of code that might have 3 or

[7] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconNamingGuidelines.asp
[8] http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html

more sets of braces nested within each other. The first method is also the recommended approach for .NET (C#) development.

Typeless/Dynamically Typed Languages

In “typeless” languages such as Perl, ColdFusion and PHP, it is often important to denote the type of variable to be expected as part of the variable name. This method has been referred to as the Hungarian Notation[9] of variable naming. In a simple for loop, we might choose to use the following PHP code instead of a simpler variable name because it becomes clear what each type of variable is being evaluated at all steps of the loop.

```
print “<table><tr>\n”;
for( $iCounter = 0; $iCounter < count($aWords); $iCounter++ )
      {
      print “<td>” . $aWords[$iCounter] . “</td>\n”;
      if($iCounter % 2 == 0)
      {
            print “</tr><tr>\n”;
      }
}
print “</tr></table>\n”;
```

The following table provides a few examples of standard prefix values for languages to help hint at what type of value is stored in a particular variable. While this method has been largely eliminated in most iterations of modern languages, there are still places where it is helpful to know what type if information is in each one. You may choose to use this sparingly throughout your code as needed.

Type	Common Prefixes	Example Variable
Boolean	b, bool	`bFlagName`
Integer	i, int	`iCounter`
String	s, str	`sStreetName`
Array	a, arr	`aCartItems`
Query	q, qry	`qResultSet`
XML	x, xml	`xDocument`
Object	o, obj	`oPerson`
Pointer	p, ptr	`pLetter`

Some languages, notably ActionScript, include class hinting tips in the authoring environment if you use consistent extension values, a full list of those is available from Adobe’s website[10] but some examples are `_xml` for XML documents, `_mc` for Movie Clip and `_str` for String.

Directory Structure

[9] http://www.irritatedvowel.com/Programming/Standards.aspx
[10] http://www.adobe.com/devnet/flash/whitepapers/actionscript_standards.pdf

Depending on the server environment used, web servers may or may not be case sensitive in regards to filenames. A best practice to use when creating directory structures is to be as explicit as possible and to create all lowercase directories, this has emerged as common convention most major English language websites[11] including Amazon.com, BBC, Blogger, Dell, Google, Live.com, Microsoft, MSN, MySpace, Napster, National Wildlife Federation, NPR, Yahoo!, and YouTube.

Unlike with variable names, a clear convention regarding multiple word directories has yet to emerge, however, three primary methods are clearly visible, one where each word becomes a sub-directory (`http://www.domain.com/about/us/`), a second where a unique character separates the sub-directories (`http://www.domain.com/about-us` or `http://www.domain.com/about_us/`) and a third where they are merged together as one word potentially using a Camel notation (`http://www.domain.com/aboutUs` or `http://www.domain.com/aboutus/`).

The best advice regarding structure is to use a directory that accurately describes the content that a user might encounter in that directory. This is helpful in aiding the user in navigation as well as search engine indexing. Google uses filename and directory as part of its index for relevance so accurately described content is helpful in search engine placement[12]. Whenever possible use the hyphen instead of the underscore to ensure the best placement for your content in search engines.

1. Directories should indicate the subject matter of the content contained therein.
   Example: `/downloads/`
2. Directories should use hyphen as spaces when necessary. Example: `/shopping-cart/`
3. Common directories for web based applications:
   `/classes` – in environments that support class files, they would be stored here
   `/components` – in ColdFusion environments, CFC files are stored here
   `/includes` – for application level includes such as configuration files etc
   `/images` – for design images, note: subdirectories are common in this folder
   `/styles` – for CSS include files
   `/javascript` – for JavaScript include files

Filenames

Generally speaking filenames should use all lower case values, much like directories. Filenames should always describe the content contained within them for search engine placement as well as clarity for developers. Since filename length limits have all but been lifted from modern operating systems it is no longer necessary to use cryptic naming conventions. A file that deals with corporate history which might have been saved as corphist.html before, should be named in one of the following ways – of course your file extension may vary depending on your environment:

1. `corporate-history.html` – preferred method for readability and search engines [13,14,15]
2. `corporate_history.html` – readable
3. `corporatehistory.html` – less desirable

[11] Top 14 of 15 unique sites on Alexa.com 9/26/06 http://www.alexa.com/site/ds/top_500
[12] http://www.google.com/search?q=About+Us
[13] http://www.pandecta.com/forum/checklist3.html
[14] http://www.lattimore.id.au/2005/05/28/search-engine-optimisation-seo-dashes-versus-underscores/
[15] http://www.mattcutts.com/blog/dashes-vs-underscores/

The strength to this approach is that the file can be read clearly without opening it to verify its contents. Additionally, search engines evaluate the hyphen as a space in the filename and so would index the content as “corporate history” instead of “corporatehistory” which is how the other method would be indexed. It should be noted a large number of proponents of the underscore will tell you otherwise, but at least for the time being, the “dash” or hyphen is the best bet.

Moving JavaScript and CSS into Include Files

Often while developing websites you will create the same chunk of code over and over again. If you find yourself doing this, you’re not effectively encapsulating your application. For instance, if you are routinely creating a function that validates a string length to determine if a field was properly filled out, you might consider creating a function that handles the process for checking a string length into an include file. This way if a change needs to be made in that logic, it is easy to do. While this seems like a basic development principle, AF-Design often spends 10% to 15% of a projects development time for legacy applications doing just this!

```
<script type=”text/javascript”>
function IsStringBlank(StringIn)
{
// check the character length of the string
if(StringIn.length == 0)
	{
		return true;
	} else {
		return false;
	}
}
</script>
```

can be included in a file using this single line of code

```
<script src=”/javascript/string-functions.js” type=”text/javascript”></script>
```

The side benefits of including functionality in this method is browser caching will then store this chunk of code for all pages locally, saving download time for your website. While this simple function only saves you a few bytes on subsequent loads, with more complex functions or CSS, handling image roll-over effects and the like, that number can increase dramatically.

Contents of a File

This section begins to discuss some topics surrounding code segmentation, there are MANY approaches to doing this correctly and at the expense of being trite, this is included only to clarify the above section. If you are implementing a full Object Oriented MVC type application, this is certainly not going to provide the level of detail you need and should be skipped. If you don’t know that an MVC approach is, read on, it could save you hundreds of hours in application development time.

As mentioned before, file names should be descriptive of the contents in that file, however, we often create files that have more than one purpose. While there are some exceptions to this rule, it is

best to segment files out into individual tasks or processes. For example, with web applications, we often need to create JavaScript form validation to handle checking of field values prior to passing the data back to the server. We can create one giant file that handles all components of this validation. This file might be called `contact-us-validation.js`. This will work for smaller applications, such as a contact us type application. Now what happens the applications' forms have hundreds of fields?

One method for handling this is to segment that file down further. Instead of one large file that handles all aspects of the validation, we could create three files, which can more cleanly handle the applications nuances. The original file, `contact-us-validation.js` would likely still exist as a container for specific functionality that doesn't fit within the other two files, a `form-field-validation.js` file would house any specific checks for values such as `IsStringBlank`, `IsStringEmail`, or `IsStringPhoneNumber` and any other helpers for accessing field information such as `GetSelectBoxValues`. This segments out the reusable code logic for a form. This file can then be utilized in other forms as well since all of the validation functions are generic and not form specific. The last file we would need to create is a display logic file such as `form-field-error-display.js`, one that handles the display type functions. These might include DHTML to change field colors, move keyboard focus, and display JavaScript alert boxes.

Databases

Databases should follow a similar methodology to variables using Pascal case wherever possible. Table names should be clear as to the contents included therein. Whenever possible, err on the side of clarity. The author of this document prefers to use some Hungarian notation when denoting special objects within a database, specifically around advanced functionality supplied by different vendors.

Data normalization is an important consideration in designing web application databases. It is strongly recommended that you read current materials[16] on data normalization when creating web databases so when necessary any departure from good design is deliberate. Occasionally, it is okay to ignore normalization in applications that are only collecting information for later export to some other analysis tool such as Excel. For business-oriented data that will live and be used in the database, consider more thorough normalization practices.

This list should provide a framework to assist in naming conventions and some simple tips to make database creation and design easier for both developers and business users later.

1. Use descriptive names for databases. Example: `ProductCatalog`
2. Use plural nouns for tables. Examples: `Users` not `User`, `Images` not `Image`
3. Normalize data as much as is relevant for the application.
4. Include a primary key value for all tables.
5. Primary key values should be clear indicators to what table they are referencing. Example: `UserID`
6. Index all relevant columns used in lookup queries.
7. Use stored procedures when supported by the database.
8. Prefix database features as follows if they are supported in the environment:

[16] The further reading section has a few good initial resources: http://en.wikipedia.org/wiki/Database_normalization

Feature	Prefix	Example
Stored Procedure	sp_	`sp_AuthenticateUser`
User Defined Functions	f_	`f_StripHtmlEntities`
Views	v_	`v_TransactionHistory`
Triggers	t_	`t_CalculateTax`