
**Ministry of Community Development**

# Standardized Coding Practices

**Date: June 1st, 2006**
**Prepared By: David Ell**
**Project: CAWS Standards Documentation**
**Harvest Package Name: RFC_000164**
**Harvest Version: 11**

## Table of Contents

# Revision History

| Date | Harvest Version | Section | Description | Author |
|---|---|---|---|---|
| 07-JUL-2004 | 2 | All | Update after group review. RFC_000164 | Colin Bussanich |
| 22-JUL-2004 | 3 | All | Updated with input from DWS group | David Ell |
| 17-AUG-2004 | 4 | All | Update with feedback from DWS group | David Ell |
| 25-AUG-2004 | 5 | 3.3 Additional HTML Client/Side Active Server Pages Component Identification | Added this section, updated TOC, updated history, updated Harvest version | Alan Bullen |
| 27-AUG-2004 | 6 | All | Update after group review. Title page, revision history, Section 3.3 to be updated by Alan. | Colin Bussanich |
| 27-Aug-2004 | 7 | 3.0 Comments<br>3.3 | Added, 'see section 3.3' to last bullet.<br>Added ASP example to comments example, and rewrote part of 3.3 to make it more concise.<br>Added a comment on the need for adequate security. | Alan Bullen |
| 10-Sep-2004 | 8 | All | Sign off authority change, paragraph spacing | Colin Bussanich |
| 11-Nov-2005 | 9, 10 | All | Updated sign-off authority and Ministry name. | David Ell |
| 01-Jun-2006 | 11 | 6 | Added Section 6 | David Ell |
| 22-Oct-2008 | 12 | Main | Headers | Ayano |

# Document Approval

This document has been approved by:

| | |
|---|---|
| Signature | Date |
| *Fredo Vanlierop* | *Senior Technical Architect* |
| Print Name | Title |

# 1.0 Overview

## 1.1 Outline

This document will outline the standardized coding practices for the Ministry of Community Development (CD).

## 1.2 Detailed Description

These coding standards include:

- Variable naming standards
- Commenting standards
- Modularization recommendations
- Readability guidelines
- Required Imports

These standards and guidelines apply to all developers and development languages.

## 1.3 Purpose

These coding practices allow for easier code maintenance.

## 1.4 In Scope

These standards apply to new systems, as well as major enhancements of existing systems. The following are in scope for this document:

- Variable and constant naming
- Commenting standards
- Modularization
- Code format
- Required Imports

## 1.5 Out of Scope

The following are not in scope for this document:

- Ministry standard toolsets
- Software configuration management

# 2.0 Variable Naming

## 2.1 Scope and Usage Prefixes

| | |
|---|---|
| g | Global |
| m | Local to module or form |
| st | Static variable |
| (no prefix) | Non-static variable, prefix local to procedure |
| v | Variable passed by value (local to a routine) |
| r | Variable passed by reference (local to a routine) |
| arr | Array |

## 2.2 Variable Type Prefixes

### 2.2.1 Intrinsic Types

| | |
|---|---|
| bln | Boolean |
| byt | Byte |
| chr | Character |
| col | Collection |
| cur | Currency |
| dat | Date and Time |
| dbl | Double |
| dec | Decimal |
| int | Integer |
| lng | Long |
| obj | Object |
| shr | Short |
| sng | Single |
| str | String |
| udt | User-defined Type |
| var | Variant |

### 2.2.2 Organizational Types

| | |
|---|---|
| cls | Class |
| mdl | Module |

### 2.2.3 User Interface Types

| | |
|---|---|
| cal | Calendar |
| cbo | Combo box, drop-down list box |
| chk | Check box |
| cmd | Command button |
| cmp | Comparison validator |
| dir | Directory list box |
| dlg | Common dialog |
| drp | Drop down list box |
| drv | Drive list box |
| fil | File list box |
| frm | Form |
| grd | Data grid |
| hsb | Horizontal scroll bar |
| hyp | Hyperlink |
| img | Image |
| lbl | Label |
| lin | Line |
| lst | List box |
| mdi | MDI child form |
| mnu | Menu |
| msg | MS Flex grid |
| ole | OLE |
| pic | Picture |
| pro | Progress bar |

| | |
|---|---|
| rad | Radio button |
| rbl | Radio button list |
| req | Required field validator |
| rgx | Regular expression validator |
| rng | Range validator |
| sbr | Status bar |
| tab | Tab control |
| tlb | Toolbar |
| tmr | Timer |
| tre | TreeView |
| txt | Text box |
| vsb | Vertical scroll bar |

### 2.2.4 Database Object Types

| | |
|---|---|
| cnn | Connection |
| dad | DataAdapter |
| db | Database |
| com | DataCommand |
| dr | DataRow |
| ds | DataSet |
| dt | DataTable |
| dbe | DBEngine |
| qry | QueryDef |
| rs | Recordset |
| rel | Relation |
| tbd | TableDef |
| trn | Transaction |

### 2.2.5 Other Object Types

| | |
|---|---|
| con | Container |
| doc | Document |
| fld | Field |
| fs | FileStream |
| grp | Group |
| idx | Index |
| prm | Parameter |
| usr | User |
| wsp | Workspace |

## 2.3 Variable Capitalization

Variables will be capitalized using camelCase with numbers. This is to distinguish between words without using spaces or underscores. Numbers are allowed. After the first character, the first letter of each word (or abbreviation) is capitalized.

## 2.4 Constants

Constants should be in uppercase to distinguish them from variables and other declared objects.

## 2.5 Variable Naming Example

Variables will be named using the following three methods:

1. Scope / Usage Prefix +
2. Variable type Prefix +
3. Descriptive name, using lower camelCase with numbers.

For example, a global variable, of type long, holding a contractor's hourly rate would be named gLngContractorsHourlyRate, or gLngHourlyRate.

# 3.0 Comments

All code must be commented. The following sections show an expected comment block to be included at the top of Modules and Classes, and Functions and Sub-Routines.

Follow the following principles regarding in-code documentation:

- Comments must add value
- The cost of the comment must not outweigh its value
- Comment code blocks that are large or complex
- All components must contain some identification, be dated, and an authors name, however do not comment on the methods used if it is already implicit in easily understood code blocks
- Included code (such as ASP) should indicate code start and end points (see section 3.3)

## 3.1 Modules and Classes

All Modules and Classes should have a comment block at the top of the file, which gives appropriate boilerplate information about the particular Module or Class.

```
'*****************************************************
' MODULE:          FMain
' AUTHOR:          Phil Fresle
' CREATED:         15-Apr-2000
' FILENAME:
' PATH:            ***Path of file relative to root.***
' DESCRIPTION:     ***Description goes here***
'
' MODIFICATION HISTORY:
' 1.0  15-Apr-2000            ***Date***
'      Phil Fresle             ***Author***
'      Initial Version         ***Desc. & Harvest RFC***
'*****************************************************
```

**Example 1: Module or Class Commenting Example.**

## 3.2 Functions and Sub-Routines

All Functions should have a comment block at the top of the declaration which describes the Function, its inputs, its outputs, and lists any global variables which it is dependant on or modifies. Sub-Routines would contain similar information, but would omit the inputs and outputs.

```
'************************************************
' GetUserName (FUNCTION)
'
' PARAMETERS:
' (In) - lUserID - Long -
'
' RETURN VALUE:
' String -
'
' Dependencies:
'   Globals:      ***What objects this code interacts with***
'   Files:        ***Required files for operation***
'   Database:     ***Required database/connection for
'                    operation***
'
' DESCRIPTION:
' ***Description goes here***
'
' MODIFICATION HISTORY:
' 1.0       15-Apr-2000
'           Phil Fresle
'           Initial Version
'************************************************
```

**Example 2: Function and Sub-Routine Commenting Example.**

## 3.3 Additional HTML Client/Side Commenting

Within the stateless environment of a web browser, the ability to view HTML source is an easy way to preserve an applications state within the browser session.

Many web-based programs make use of multiple server- side includes and/or function calls, which can easily hide or obscure where activity begins and ends during a browser session. To allow for easy troubleshooting, all ASP or any other CGI server side scripts, must generate HTML comments showing where they start and end in the web page session. (see Example 3). Since this reveals the relative path to each application module, ensure that application security is adequate.

Error handling within a web-based environment should attempt to deliver to the end user an user-appropriate error message. If available, detailed troubleshooting messages should be delivered to the browser in the form of HTML comments. The form of the messaging must be left to the application designers, as it depends on the nature and complexity of the application. In some cases server-side event logging may be more effective.

```
<%
Option Explicit
Response.Buffer = True
%>

<!--START:INC\common\include.asp-->
<!-- #include file = "../common/errorfns.asp" -->
<!-- #include file = "../common/constant.asp" -->
<!-- #include file = "../common/securityfns.asp" -->
<!-- #include file = "../common/utility.asp" -->
<!-- #include file = "../common/DebugLogFileFns.asp" -->
<!--END:INC\common\include.asp-->
```

IInclude.asp code above generate the first comment in the comment below, and subsequent comment are generated by each of the module Include.asp includes.

```
<!--START:INC\common\include.asp-->
<!--START:INC\common\errorfns.asp-->
<!--END:INC\common\errorfns.asp-->
<!--START:INC\common\constant.asp-->
<!--END:INC\common\constant.asp-->
<!--START:INC\common\securityfns.asp-->
<!--END:INC\common\securityfns.asp-->
<!--START:INC\admin\activitysecurity\Activityfns.asp-->
<!--END:INC\admin\activitysecurity\Activityfns.asp-->
<!--START:INC\sso\ssoutility.asp-->
<!--END:INC\sso\ssoutility.asp-->
<!--START:INC\common\utility.asp-->
<!--END:INC\common\utility.asp-->
<!--START:INC\common\DebugLogFileFns.asp-->
<!--END:INC\common\DebugLogFileFns.asp-->
<!--END:INC\common\include.asp-->
```

**Example 3: Website HTML labels in source generated by included ASP modules**

# 4.0 Modularization Best Practices

In order to make development most efficient in a team-development environment, we have developed a number of best practices around modularization.

1. All classes should be declared in separate class files.
2. Classes and modules should only contain a single set of functionality. (i.e. Don't mix math and database connection functions in the same module.)

These best practices will ensure that you have an appropriate level of modularization to work effectively with the Ministry's Harvest software configuration management system.

# 5.0 Readability

Code should be indented to aid in understanding. Use any indenting method[1] (using either spaces or tabs) as long as it makes it easier for the reader to visually track the scope of control constructs.

```
Public Function nextLine() As String()
  Dim strInLine As String
  strInLine = fwReader.ReadLine
  strInLine = Replace(strInLine, Chr(34), "")
  If Not strInLine = Nothing Then
    nextLine = strInLine.Split(",")
  Else
    nextLine = Nothing
  End If
End Function
```

**Example 4: Example of indenting code to help readability.**

# 6.0 Required Imports

For all .NET application, all required libraries must be imported at the top of each .vb file. This is to ensure that the application can be compiled by the Ministry. In order to determine which imports are required, simply view the project References in the Solution Explorer.

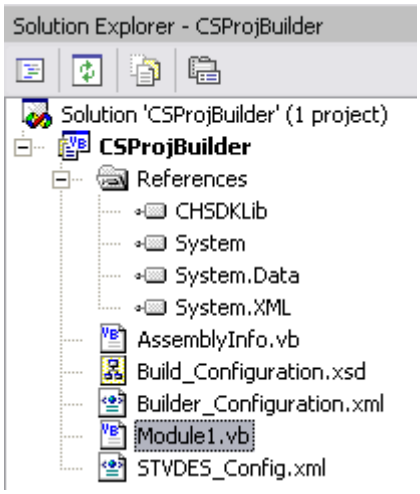


Figure 1: The References list in Visual Studio .NET 2003

[1] Such as Allman or K&R