**ColdFusion Coding Guidelines**

**Authors**

Abhinav

Ashish

Arun

Jyoshna

Manisha

Rajashree

**1. Naming Conventions:**

**1.1. Variable Naming:**

* While naming variables containing more than one word, a case structure like *camelCase* or *PascalCase* should be followed, you can also use underscores. Whatever you use, stick to it throughout the application.
* A proper scope always has to be used with a variable, the scope of the variable should be written in uppercase( If an existing application, it should be consistent throughout project. Don’t change the existing convention in the middle).
* Initialize variables inside a function using the LOCAL scope instead of the var scope.
* Abbreviation of variables names should be avoided. For Example, if your variable is a flag for customer service check, a better name would be ‘customerServiceCheck’ instead of csCheck.

**1.2. File and Folder Naming:**

* File and folder names should also follow a consistent naming convention like PascalCase, camelCase , or underscored\_case.
* The name of a file should symbolize the functionality being done by the code in the file. Like a file containing the code for displaying a login form to the user, should be named as ‘login.cfm’ and the file containing the functionality to be performed when the login form is submitted should be named as ‘login\_action.cfm’.
* Folder names similarly should also symbolize.
* Every section of an application should be kept in separate folder. Like normally we have admin and user sections in a project, so files related to user section should be kept in a folder ‘User’ and files related to admin section should be kept inside ‘Admin’ folder.
* The common functionality or common files related to both sections should be kept inside a ‘Common’ folder.

**1.3. Function and Query Naming.**

* Functions should be named as per their functionality like suppose we have a function for retrieving data from a table called ‘Employees’, then the function name should be ‘getEmployeeDetails’.
* Function names should be verbs if the function changes the state of the program, and nouns if they're used to return a certain value.
* Query names should define what they are doing. A query getting user info should have a name like “*fetchUserInfo*”.
* All queries should have a proper scope if they are not in the VARIABLES scope. For example each query inside a function should be part of the LOCAL scope.
* Query name can be avoided for queries which do an update or an insert as there will not be any result set created for these queries.

**2. Code Formatting:**

* Tabs should be preferred over spaces to indent, if a project already all it’s indentation in spaces than to maintain consistency all new added code should follow spaces to indent.
* There must be even whitespace before and after logical operators, syntax should go as it is.

<cfset VARIABLES.myVar = “Mindfire”> (**good**)

<cfset VARIABLES.myVar=”Mindfire”> (**bad**)

* While writing SQL queries in cfquery, it is advisable to write SQL keywords in uppercase. (If it is an existing application, stick to the existing case and make it consistent throughout the application).
* Longer lines of code should be broken into multiple lines wherever possible.
* Tags and their attributes should be written in lowercase letters.
* The order of attributes in tags should be consistent throughout the application.
* The use of quotations( double or single) should be consistent throughout the application.
* While using variables in functions or any cfm page, normally we should declare them at the starting of the page.
* While declaring variables, logical blocking of variables should be kept in mind. Like suppose, we are declaring four numeric variables and two string variables, then we should be having variables declared in groups – first integer variables, and then strings or vice-versa.

**3. Code Commenting:**

* Every file in an application should have header that contains a block of comments at the starting containing the filename, description about what functionality is being accomplished in the code of this file, and the date when the file was created.
* Each function defined in a component, must have a header with the name, description, arguments description and return type for the function.
* Every critical part of code should be properly commented. It’s better to even give inline comments at the start of a block of code accomplishing a particular functionality.
* Any file should not be having unused piece of code as commented. If needed then we can comment with proper reasoning.
* The hint parameter for cfcomponent, cffunction and cfargument should be used to provide proper hint.
* It is important that comments be kept up-to-date.
* Make comments meaningful. Focus on what is not immediately visible. Don't put the comment for statement that is self explaining, for example:-

i = 0; // Set i to zero.

* When a comment will not fit on a single line, it may be necessary to break it.

**4. Error handling:**

* Every critical part of code should be kept inside a try-catch block like queries,http requests, etc.
* Any code that goes to catch block, should be handled properly to display user-friendly error messages to the user . Also, don’t show the CF or DB error to user.
* Define the specific error type inside catch block and accordingly log the message. (Example, when you are catching a SQL specific exception log the Query Error).
* In catch block, we should be calling a function that should contain the functionality of writing the error to the error log file.
* Avoid empty catch blocks because it hides the errors and exception.
* Use cfthrow to log the user specific exception.
* Use the onError event in Application.cfc to handle the errors that are not handled by try/catch code on the application pages.
* Use the cferror tag to specify ColdFusion pages to handle specific types of errors.
* We should use the 'Missing Template Handler' & 'Site-wide Error Handler', it's present in CF Administrator under Server Settings/Settings.

**5. Project Development guideline:**

**5.1. Project Structure:**

* The project root folder should have the same name as the application.
* The root folder should keep the application.cfc, the module specific folders and the dedicated JavaScript, CSS & image folders.
* Create Configuration or Config file to keep the constants used in the application Like the DSN name, Encryption key or any server path. You could keep this file outside the root folder so that It shouldn’t accessible by browsing, due to security concern. You could keep those in a DB table also, if it’s not a overhead to the application due to DB calls.
* Use separate modules for each section and the related files to a specific module should be stored in dedicated folder.
* There should be module specific component files. There should be one base component file which will keep common functions used by different CFCs like error handling, sending mail, file upload or any server side validation logic. The component files will extend the base component file to use those functions.
* “Use custom tags to encapsulate your application logic so that it can be referenced from any ColdFusion page”. Project folder should have “customTag” folder to keep the customTags used in applications.
* Project folder should have “Include” folder to keep track of all included files.
* Don't add the JS code inside the cfm files. Use separate JS files and include it at the bottom of the CFM page.
* Don't use the inline styles and put all the styles in the separate CSS file for reuse and better code readability.
* All the application global variables can be stored in the configuration file/ application.cfc file as request-scoped variables.
* All the hard-coded data should be stored on the configuration file for easy maintenance and to reduce redundant.
* Presentation layer should be different from the code handling internal things.
* A dedicated Database access layer should always be present in an application.
* An Application Framework like Framework 1, MACH II, ColdBox, Fusebox should be used. You could write your own framework too.
* Good to have the use of the ORM feature to make the application Database independent.
* Use image sprites to include the images.

**5.2. Debug:**

* For coding and debugging purpose, debugging should be enabled from ColdFusion Administrator.
* In Administrator, go to Debugging & Logging > Debug Output Settings and enable debugging according to the needs.
* Use cfsetting to show/suppress debug output.
* You could use CFOUTPUT, CFDUMP & CFABORT for the in-line debug of any functionality.
* You need to remove the debug codes like CFDUMP, CFABORT from the production environment code.
* Debug needs to be enabled for development/QA environment, but it should be disabled for the production environment.

**5.3. Validation:**

* Add server side (ColdFusion) validation for everything irrespective of the client side validation.
* Also add the proper value validation along with the Mandatory validations.
* Suppose for an email field which is mandatory, there should be validation for checking the value and in correct email format and if the field is optional, the validation should only occur when the field has any value.

**5.4. Common files:**

* Store all the generic CF functions in a common component. You could create a common.cfc file.
* Common functions can be like DATE function, string format, object conversion or anything you intend to reuse in application.
* Use inheritance to reuse the common functionalities in CF.
* Similarly, you could create a common js file and include this to the other files, want any functionalities to be used.

**5.5. Session management:**

* Use a session management(ColdFusion or J2EE) in the application.
* Validate your session in each request.
* Define your own session timeout value.
* Never pass the session cookies as the URL parameter.

**6. Application Performance:**

**6.1. Fully scoped variables:**

* Use variables with the proper scope.
* Variables only used in a single function should be scoped locally. This can be done by using the var keyword in a <cfset> or <cfscript> block.
* Scoping locally will create better management of memory, especially in recursive and nested loops and avoid corrupt variables.

**6.2. Evaluations:**

* Complex dynamically constructed expressions will negatively affect performance.

For example: <cfset #foo# = "#bar()#">

The code above should be replaced with the much simpler and more efficient:

<cfset foo = bar()>

**6.3. IIf() and cfif-else:**

* Always use cfif and cfelse instead of IIf().
* The cfif construct is significantly faster and more readable.
* IIF runs about twice as slow as a <cfif>/<cfelse> block.

**6.4. Avoid evaluate():**

* Evaluate slows processing since it is determined at runtime.
* Avoid using it unless there is absolutely no other way to accomplish the task.

**6.5. Use <cfswitch> instead of <cfif>:**

* Using <cfswitch> instead of <cfif> when evaluating a specific expression will perform faster.
* Performance also increases when using <cfswitch> instead of more than three <cfelseif> clauses.
* <cfswitch> is also often easier to read.

**6.6. compare() and compareNoCase():**

* Use compare() or compareNoCase() instead of the "IS NOT" operator to compare two items.
* Using compareNoCase is a true string comparison. Using compareNoCase should be used unless you can guarantee that the data will not contain a leading 0. For example, using IS or EQ will return true when comparing the two strings of “001” and “0001”.
* Using compareNoCase or compare is significantly faster than using IS NOT.

**6.7. listFindNoCase() and listFind():**

* Use listFindNoCase() or listFind() instead of the "IS and OR" operators to compare one item to multiple items.
* Using listFindNoCase or listFind is significantly faster than using IS and OR with five or more options.

**6.8. Query caching:**

* Querying a database is one of the most time consuming parts of a ColdFusion page.
* ColdFusion provides query caching as a way to avoid repeatedly querying the database by caching the query recordset.
* Queries that do not change frequently are the best candidates for caching.
* In ColdFusion Administrator you can specify the maximum number of queries that can be cached at a time.
* There is no limit on the size of a query, so if there are many queries with large recordsets in the cache then it may cause a memory overflow.
* As a good practice avoid caching queries for a long time.
* Use attribute "cachedWithin" in CFQUERY tag.
* To use cached data, the current query must use the same SQL statement, data source, query name, user name, and password.

Ex: cachedwithin="#CreateTimeSpan(0, 0, 6, 0)#".

**6.9. cfqueryparam:**

* Using <cfqueryparam> in inline queries will help to prevent SQL injection making sure to use the CFSQLType to verify the data being passed is of the expected type for the database.
* Using <cfqueryparam> will optimize the query for multiple repetitions. A SQL bind variable is created with some database implementations which will help to optimize database performance.
* It's not necessary to use cfqueryparam for the constant/fixed values.

**6.10.** **BlockFactor :**

* Setting the blockfactor may allow ColdFusion to retrieve data from a database in a more efficient manner.
* Determine the maximum size (in bytes) of each row of returned data and divide that into 32k. That number is your blockFactor.
* The max blockFactor is 100.
* It is not necessary for queries with less than 100 rows returned.
* It might not be supported by some database systems. It’s advisable to check, if it’s supported by your application DB before using this.

**6.11. Caching:**

* Use the <cfcache> tag in pages with contents that are not updated frequently.
* This tag tells ColdFusion server to cache the HTML to a temporary file.
* When ColdFusion gets a request for a cached ColdFusion page, it retrieves the pre generated HTML page without having to process the ColdFusion page, thus improving performance.

**6.12. Cfsavecontent:**

* For ColdFusion pages that contain some dynamic information and some content that changes less frequently, the <cfcache> tag should not be used.
* Instead you can use <cfsavecontent> to cache infrequently changing output in a shared scope variable.
* The <cfsavecontent> tag can also be used for concatenation, for which it is much faster than <cfset>.

Ex:

<cfset result = "">

<cfloop from="1" to="100" step="1" index="i">

<cfset result = result & i>

</cfloop>

The code below using <cfsavecontent> is much better from a performance

perspective:

<cfsavecontent variable="result">

<cfloop from="1" to="100" step="1" index="i">

<cfoutput>#i#</cfoutput>

</cfloop>

</cfsavecontent>

**6.13. cfstoredproc:**

* Wherever possible use stored procedures instead of SQL queries.
* Performance is increased with the use of stored procedures since it is pre-compiled.
* Using stored procedures prevents SQL injection since code is pre-compiled and not able to normally change at runtime providing for a more secure environment.
* When calling stored procedure you need to provide the parameters in the same way as those are defined in SP and need to remove dbVarName while calling.

**6.14. CFHTTP:**

* When using CFHTTP set ResolveUrl=yes only when needed.

**6.15.File functions:**

* Use file functions (e.g FileCopy(), FileRead(),FileWrite()) in place of <cffile> tag for better performance.
* The performance again increases as file size increases.

Ex:

<cffile action="write" file="c:\temp\myfile.txt" output="Some Text">

Instead use:

<cfset FileWrite("c:\temp\myfile.txt","Some Text")>

**6.16. Avoid structFind:**

* Use struct.key or struct[key] instead for better performance.

**6.17. Use arrays instead of lists:**

* Lists suffer from slow Java string processing.
* Do not, however, convert an existing list to an array. The processing time to first convert that list to an array will outweigh the advantages of accessing the data as an array.

**6.18. Avoid <cfmodule>:**

* The preferred method is to use a CFC since it is the fastest and most readable.
* Using cfmodule is slower and less readable than invoking a custom tag with a prefix and a regular custom tag invocation.
* If not using a CFC, use cfimport to invoke a tag with a prefix or simply include the file.

**6.19. Don’t use incrementValue:**

* Using x = x + 1 is typically more readable and slightly faster than incrementValue(x).

**6.20. Avoid large Query of Query objects:**

* Avoid the overhead from a memory and processor utilization aspects of large QoQ objects.
* Determine if it can be better handled using the database.
* Manipulate data at the database level whenever possible. This increases performance, maintainability and enables the separation of business logic from display.

**7. Application Security Checks:**

**7.1. General Application Security Recommendations:**

* Do not store unfiltered user inputs to the database.
* Do not assume that information you get from the HTTP request header is safe. Use safeguards for query strings, cookies, and so on. Be aware that information the browser reports to the server (user agent information) can be spoofed, in case that is important in your application.
* Do not create SQL statements by concatenating strings that involve user input. Instead, create a parameterized query and use user input to set parameter values.

**7.2.Coldfusion Web Application Security Practices:**

1. Secure the CF server with secure profile enabled.
2. Enable sandbox security and disable tags and functions which will not be used by the application.
3. Configuration of application specific properties can be done at the CFAdministrator.
4. Make sure you have Small timeouts for the session and application
5. Cookies are a useful way to keep user-specific information available. However, because cookies are sent to the browser's computer, they are vulnerable to spoofing or other malicious use. Make sure cookies are secure, HttpOnly and timeout are set properly
6. Set expiration dates on cookies to the shortest practical time you can. Avoid permanent cookies if possible. Consider encrypting information in cookies.
7. Configure request limits.
8. Disable directory browsing at server level.
9. Never enable debugging and detailed error pages in the production server. Limit it to local developer's server. (Robust Exception Information)
10. Choose a file extension other than .cfm (configured in web.xml). Use mod\_rewrite (Apache), or ISAPI Rewrite (IIS).
11. Use of <cfparam> tag with strict data-type and scaling will prevent the abuse of URL and Form variables. This tag basically will bind the incoming Form or URL variables to specific types and also create a default values for them.
12. Use of <cfqueryparam> is the common way of stopping the SQL injections and it also increases your db performance (only when the db you use supports it). It will validate dynamic SQL variables against database datatypes.
13. Try always using the custom error page that does not display to the user the entire error information. This hides the internal information and encapsulates your errors. Basically give a customized error message to the user.
14. Do not trust the incoming data and always the data type checking should be maintained.
15. Make sure the authorization and authentication checking is done at server side. It is always the developer's role to protect the database and its contents. Always add codes to get roles from database.
16. Coldfusion includes the support for Regular Expressions and CFML tags which you can use to validate the input.
17. Use <cfldap> for accessing the LDAP servers. Avoid allowing native JNDI (Java Naming and Directory Interface) calls being connected to the LDAP.
18. Use strong cryptographic functions like SHA-256 for hash and AES for symmetric encryption.
19. Clear session and user login with proper use of SessionInvalidate.
20. Use coarse-grained web services. Network latency can be the biggest performance bottleneck. Try to reduce calls to the server. Call a web service once and use a query of queries to return the detailed information for display.
21. Use the XML functions to validate XML input.

Reference URL

http://www.ipaul.com/2009/08/01/coldfusion-best-practices/