

CFC TIPS AND TRICKS



Overview



- CFC Basics
- CFC Best Practices
- Inheritance Examples



CFC Basics



When do you use a CFC?

- CFC
 - ▣ Business Logic
 - ▣ Function Libraries
 - ▣ Part of a Object Oriented Framework
- Custom Tag
 - ▣ Display Elements
- CFInclude
 - ▣ Discourage it use
 - It's only a shortcut
 - It's not black box, you don't control directly what goes in and out
 - Only really appropriate in simple use.

How to Call a CFC

- Many methods
 - `<cfinvoke>`
 - `<cfobject> + <cfinvoke>`
 - `CreateObject + <cfinvoke>`
 - `<cfobject> + object.functionName()`
 - `CreateObject + object.functionName()`
 - `CreateObject.functionName()`
- Be aware that repeated versions of 1st and 6th method are inefficient
- Otherwise, it doesn't matter, but be consistent

Instantiating to a Shared Scope

- **Benefits**
 - Fastest Execution
 - Reduced Resource Use
 - Adds a pseudo compilation which partially tests code
- **Which Scope**
 - Application
 - Function Libraries
 - Object Oriented as Service
 - Session
 - Object Oriented as Persistent Representation of reality
 - User
 - Shopping Cart
 - Variables
 - Object Oriented as Transient Representation of reality
 - Article
 - Merchandise
- **How To**
 - Objects are variables
 - Variables are set to scopes
 - So `session.testObj` instead of `testObj`
 - Locking rules still apply on creation

Variables in CFC

- Unscoped variables are created in the variables scope.
 - ▣ `<cfset test = <cfset variables.test`
- Except that local variables also have no scope
 - ▣ `<cfset var test ≠ <cfset variables.test`



CFC Best Practices

Best Practices

- Strive for “thread safe” CFC’s.
- Fully Scope all Variables use in Functions
- Get the ‘access’ right
- Output = FALSE
- Init Method
- There are good exceptions to most of these.

Strive for “thread safe” CFC’s

- ❑ Thread safe means that they will work correctly across simultaneous requests.
- ❑ Even if you aren’t using in a shared scope, these are useful.
- ❑ This cuts down on very hard to track down bugs.

Var scope all of your local variables

- ❑ Unless specifically scoped all variables exist in variables scope.
- ❑ Variables scope persists for life of CFC.
- ❑ For application scoped CFC's this can be a long time, with many simultaneous hits.
- ❑ A variable scope variable can change before it is returned.
- ❑ Even worse with i counters.

Var Scope Tricks

- Use varscoper tool on the Wharton Developer's Center.
- Scopes are special structures, so:
 - ▣ Create a var scoped 'local' structure
 - ▣ Treat it like a scope.
 - ▣ (Does not work with varscoper)
- Write Functions on own cfm page, with:
`<cfdump var="#variables#" />`

Passing by reference vs passing by value

- ❑ Passing by value means that a copy of the old variable is created as a new variable.
- ❑ Passing by reference means that a pointer to the memory location of the original variable is created.
- ❑ When passing by reference changes made to either “variable” will change both.
- ❑ Most variables in ColdFusion pass by value. Structures do not.

Don't modify arguments directly

- Read it in a Comp Sci text book
- Other experts agree
- Makes debugging easier, if it is necessary
- Concrete example

Don't use 'This' Scope

- ❑ This scope is not thread safe.
- ❑ `This.variableName` inside CFC corresponds to `CFCObject.variableName` outside CFC.
- ❑ Use Getters and Setters instead

Don't Interact With Other Scopes

- Don't interact with scopes outside CFC
 - Application
 - Session
 - Cookie
 - Form
 - Url
 - CGI
 - Server
 - Request
 - Client

Don't we break this rule all the time?

- Yes, yes we do.
- Why?
 - ▣ Point of rule is to ensure that CFC is a portable and reusable as possible.
 - ▣ Business logic tied to application database isn't really portable in the first case.
- However, limit as much as possible.
- Group together as much as possible.

Fully Scope all variables in Functions

- Scopes in typical function
 - ▣ This
 - ▣ Variables
 - ▣ Arguments
 - ▣ Local
- Local and Arguments variable names cannot be the same.
- Scope Search Order
 - ▣ Local
 - ▣ Arguments
 - ▣ Variables

Get the 'Access' Right

- Private
 - ▣ Only accessible by other functions in CFC.
 - ▣ Used for sub functions only relevant to CFC.
- Package
 - ▣ Only accessible by code in same folder.
 - ▣ Any sort of organization makes this almost useless.
- Public
 - ▣ Accessible by any code on the server.
 - ▣ Sounds scary, but relatively safe in our environment.
- Remote
 - ▣ Used for webservices.
 - ▣ If building webservices these should be accompanied by some security

Output = False

- Goal here is separation of concerns
- In today's environment we can go with either Flex 2 or HTML for front end.
- Don't make this choice in the CFC.
- Possible exception, display only CFC.

Init Method

- Every CFC should have an init function that returns “This”
- Makes adding starting parameters very easy.
 - `<cfset testObj = CreateObject("component","test").init() />`
 - `<cfinvoke component="test" method="init" returnvariable="testObj" />`

Exceptions

- Every one of these best practices has exceptions
- Better to know the rules before you break them.

Don't Be This Guy





Inheritance Examples

Inheritance – Relevant Examples

- Wharton Accounts
- Wharton Ldap – Wharton Auth

Wharton Accounts

- Wharton_account
 - FirstName
 - LastName
 - PennID
 - Username
- ▣ Wharton_account_ad
 - DistinguishedName
 - OU
- Wharton_account_ad_exchange
 - MailboxLocation

Wharton Ldap and Wharton Auth

- Variable Sharing