CFC Crash Course

An Introduction to ColdFusion Components

alagad

120 Nelson Lane , Clayton, NC 27527 | p 1 (888) ALAGAD4 | f 1 (888) 248.7836 | www.alagad.com | info@alagad.com

Doug Hughes, President

What You Will Learn

- What a ColdFusion Component (CFC) is
- How to write and use CFCs
- CFC Best Practices
- Overview of Object Oriented Programming with CFCs

Why Use ColdFusion Components? (page 1)

- Code Reuse
  - CFCs are similar to custom tags but with multiple entry points and ability to hold their own data.
- Black boxes
  - Developers shouldn't care about implementation details "under the hood".
  - A developer only needs to know the interface.

Why Use ColdFusion Components? (page 2)

- Organization
  - CFCs can be organized into a hierarchy of packages.
- CFCs bring the power of Object Oriented Programing to ColdFusion
- Remotely expose your business logic.
  - Web Services and Flash Remoting

Why Use ColdFusion Components? (page 3)

- CFCs are the foundation of most modern frameworks.
  - Model-Glue
  - Mach II
  - Fusebox
  - Reactor
  - ColdSpring
  - Etc, etc.

What Does a CFC Do?

- A CFC...
  - Holds Data (like a structure)
  - Provides mechanisms for acting on it's data (functions)

Terminology

- **Class / Component:**
  - This is the CFC file you write.
  - A blueprint for an object.
  - Defines the data a CFC can hold and the actions that can be taken on that data.
- Example: Cat.cfc

Terminology

- **Property:**
  - Describes a characteristic of an object
  - Data held within the CFC.
- Example: color, weight, furLength, name

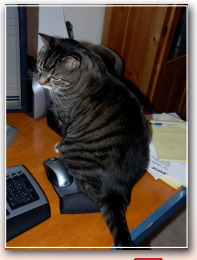
Terminology

- **Methods / Functions:**
  - Defines something the object can do.
  - Used to act upon properties.
- Examples: meow(), nap(duration), eat(food)

alagad

Terminology

- **Object**
  - A particular instance of a class
  - An instantiation of a CFC
- Example:
  - My Cat sam with short gray hair weighs 15 pounds.



alagad

Key Points

- Classes describe objects.
- Objects are one instance of a class.

alagad

The Parts of a CFC

- <cfcomponent>
  Defines a component (a Class)
- <cffunction>
  Defines a function (a Method)
- <cfargument>
  Defines a parameter of a function.
- <cfreturn>
  Returns results from a function.

alagad

<cfcomponent>

- CFCs are wrapped in a <cfcomponent> tag.
- All CFC code is placed inside the <cfcomponent> tag.
- The <cfcomponent> tag and contents are placed in a file with a ".cfc" extension.
- Example: HelloWorld.cfc
- Only one component per ".cfc" file.

<cffunction>

- Defines a method - something our CFC can do.
- Can accept arguments using the <cfargument> tag.
- Returns results using the <cfreturn> tag.

A Very Simple Example

```
<cfcomponent>

  <cffunction name="sayHello">
    <cfreturn "Hello World!" />
  </cffunction>

</cfcomponent>
```


<cffunction> Attributes

Name
- Required, specifies the function's name
- Used to invoke the function from an instance

Access
- Indicates the context from which the function can be invoked
- Options are: public, private, protected, remote

Hint
- Developer provided documentation about the function

Output
- When true, the function behaves like a <cfoutput> tag
- When false, the function behaves as if in a <cfsilent> tag (use this!)

ReturnType
- Indicates the type of data which will be returned by the function

<cfreturn>

- Used to return a value from a function
- Data being returned must be of correct type
- Has no attributes

```
<cfreturn "Hello World!" />
```


A More Complete Simple Example

```
<cfcomponent hint="I am a hello world component">
  <cffunction name="sayHello"
    returntype="string"
    access="public"
    output="false"
    hint="This method returns the string 'Hello World!'">

    <cfreturn "Hello World!" />
  </cffunction>

</cfcomponent>
```


<cfargument>

- Used to define function arguments
- Used within a <cffunction> tag
- Arguments are placed into the arguments scope within the function

```
<cffunction name="echo">
    <cfargument name="string">
    <cfreturn arguments.string />
</cffunction>
```


<cfargument> Attributes

Name
- Required, specifies the argument's name

Hint
- Developer provided information about the argument
- Displayed in generated documentation

Required
- Indicates if the argument is required

Type
- Indicates the required type of variable to pass in

Default
- A default value for the argument
- Used when no value is provided

Now What?

- You now know the basic building blocks of CFCs.
- How do you use them?

Instantiating CFCs and Invoking Methods

Instantiating a CFC

- Creating an instance of a CFC.
- Making a particular cat from the blueprint.

Invoking a Method

- Causing a method to execute.
- Making the Cat meow.

<cfinvoke>

- Invokes a function on a component or an instance.
- Pass arguments using the <cfinvokeargument> tag.
- This is quite verbose.

<cfobject>

- Creates an instance of an object
- Fairly verbose

CreateObject()

- Creates an instance of an object.
- Least verbose

Objects are Stateful

- Objects can store data inside themselves
- Similar to a structure.
- Several scopes into which data can be placed.

Scope Considerations

This scope

- Creates a public property on the component

Variables scope

- Private to a CFC instance

'Var'ing variables

- Declares a variable local to a function

"This" Scope

- Publicly accessible properties
- No data verification
- No control over types
- Behaves somewhat like a structure in that the variables don't need to exist ahead of time.

“Variables” Scope

- Privately accessible properties
- No data verification
- No control over types
- Not identifying a scope inside a CFC uses the CFC’s Variables scope

‘Var’ed Variables

- Used only in functions
- Exist only for the duration of the function execution
- Un vared variables end up in the component’s variables scope
- Varing Helps keep code thread safe and prevent strange bugs
- **You** must var any variable returned by tags
  - <cfquery>
  - <cffile>
  - A lot of tags return variables, whether you realize it or not.
- Must be placed after the last <cfargument> tag and the first line of code in a function

‘Var’ing Example

```
<cffunction name="foo">
    <cfargument name="bar" />
    <cfset var example = “Example value” />
    <!--- code goes here --->
</cffunction>
```


Accessors and Mutators

- A fancy way to say Getters and Setters
- Accessor and Mutator methods are used to set and get the value of a private property
- Can specify a specific data type
- Allows for verifying data values
- Can make read only properties
- Best practice

Constructors

- Constructors are code which is executed when a component is instantiated.
  - Provided by most OO languages
- ColdFusion does not provide constructors!
- Instead use Pseudo Constructors
  - Anything outside a <cffunction> tag in a component
  - Preferable to create a method named init() which accepts configuration settings and returns the configured object.

Pseudo Constructors

- Any code outside of a <cffunction> tag
- Executed when CFC instantiated
- Can be used to initialize variables with default values, etc.

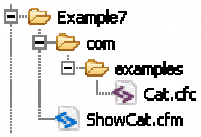
```
<cfcomponent>
    <!--- begin pseudo constructor --->
    <cfset variables.foo = "bar" />

    <cffunction ...>
       <!--- do stuff --->
    </cffunction>
</cfcomponent>
```


Init() method

- Preferable to the pseudo constructor area
- Add a method named "init" which configures and returns the object
- A standard convention in ColdFusion
- A good place to set configuration settings

Packages

- CFCs are organized into packages
- Packages are directory names separated by periods
- Packages are relative to the base template path
- Mappings in the CF administrator can also be used as packages



CFC Lifespan

- A CFC "lives" only as long as the scope it's in.
- You can place a CFC in a persistent scope to persist across multiple page requests
- Data in CFC persists along with the CFC

CFCs are Best used to represent something

- Not just a collection of functions
- Can be used to model real world concepts
- For instance:
  - Cat
  - Vehicle History Report
- ***This is the foundation of Object Oriented programming!***

Other Uses for CFCs

- Web Services
- Flash Remoting
- Event Gateways require CFCs
- Application.cfc
- *And more!*

The Three Tenets of Object Oriented Programming

- **Encapsulation**
- Inheritance
- Polymorphism

Three Tenets: Encapsulation

- **The component Has The Power!**
- Literally means is "Data Hiding"
- Perhaps the biggest advantages of OO
- Components become "black boxes"
- Reduces repeated business logic
- All a developer needs to know is the CFC's interface (it's functions and properties).

Three Tenets: Encapsulation

- Account.cfc

```
<cfcomponent>

   <cffunction name="getBalance">
      <!--- do anything needed to get and return the balance --->
   </cffunction>

   <cffunction name="deposit">
      <!--- do anything needed to deposit funds --->
   </cffunction>

</cfcomponent>
```


Encapsulation Eases Change

- If your logic is encapsulated you can change implementation without side effects.
- So long as the arguments and return value remain the same implementing code will not break.

Three Tenets: Inheritance

- A means by which a CFC can get all of the behavior and data of another CFC.
- An extending CFC can change the behavior of the extended CFC.
- Frequently abused
- **Favor Composition over Inheritance**
  - Hard to understand, but very valuable.

Extending CFCs

- CFCs have the ability to "Extend" another CFC.
  - Get all the functionality of the parent CFC
  - Customize behavior
  - Access to private methods
- The CFC extended is known as the "super class"

The Super Keyword

- Can be used to call methods on a super class.
- Only supports one level of inheritance
  - super.xzy() works
  - super.super.xyz() does not work

Three Tenets: Polymorphism

- Literally means "Many Forms"
- The language can treat two objects as if they are the same type
  - A lion and tiger are both types of cats
  - A Checking Account and a Savings Account are both types of Accounts.

Polymorphism in ColdFusion

- Provided by:
  - Inheritance
  - <cfinterface>
- Allows code to "know" how to interact with components.
  - If I know how to interact with an Account then I know how to interact with a Checking Account and Savings Account.

Interfaces

- <cfinterface> tag
  - New in ColdFusion 8
- A contract that guarantees the functions an implementing CFC will have
- Provides for Polymorphism without Inheritance

Composition

- Composition means to build your component out of other components.
- A fantastic alternative to inheritance.
- Creates more cohesive components.

CFC Metadata

- <cfdump>
  - Dumps a description of the object.
  - Useless unless debugging.
- getMetaData()
  - Returns a structure of information on a component.

Interface Metadata

- getComponentMetaData()
  - Returns a structure of information on a class or interface.

Coding With Responsibilities

- CFCs should only be responsible for one thing
- CFCs should do that one thing well
- Ask yourself...
  - What should an object be responsible for?
  - What shouldn't an object be responsible for?

What You Learned

- What a CFC is
- Why to use CFCs
- How to write and use CFCs
- CFCs are the foundation of Object Oriented Programming

Questions and Answers

-

Exercise

- Create a CFC that will return a fortune and a query of categories
- Create a script that instantiates the CFC and calls the methods on it.
- Use the CFC in your procedural fortune application.
- Let me know when you're done.

Let's Discuss

- What problems did you run into?
- What problems do our foresee with CFCs?
- What are your thoughts about using CFCs so far?
- How did your CFC know which datasource to use?
- How does your CFC make use of encapsulation?
- Is your CFC cohesive? (hint: No, but why not?)