Intro to CFC's

Presented by Pete Freitag.
Foundeo Inc.

What's a CFC?

- ColdFusion Component
- Similar to a Class in OOP
- A Logical Grouping of data and functions (or methods)

OOP, OMG!

inheritance constructors polymorphism

interfaces overriding **beans** dao factories

gateways design patterns oop getters

setters **super** this class object orm

singleton abstraction coupling

void arguments encapsulation

What's all that fancy talk?

Words I may interchange

- Component = Object
- Function = Method

Why use CFC's

- Maintainability
- Reusability
- Quality
- Readability
- Separation of Responsibility

Functions

- We need to know how to write a function before we can write a CFC.
- We must use the CFFUNCTION tag for CFCs.

<CFFUNCTION>

Common Tag Attributes

Attribute	Description
Name	The name of the function.
ReturnType	The data type (eg "string") of the data that the function returns. Or void if it doesn't return anything.
Output	Yes/No this function outputs stuff to browser.
Hint	Documentation for the function.
Access	Only Applies to CFC's. What code can call this function? Public, Private, Package, Remote
Roles	For integration with CFLOGIN. Who can call this function?

A Simple Function

```
<cffunction name="todayIs" returntype="string" output="false">
   <cfreturn DateFormat(Now(), "dddd mmmm d, yyyy")>
</cffunction>
```

Data is returned from the function with the <cfreturn> tag.

Passing Data into Functions

- The data that is passed into a function is called an **argument**
- We define arguments with the... you guessed it: <cfargument> tag.

<CFARGUMENT>

Common Tag Attributes

Attribute	Description
Name	The name of the argument.
Type	The data type (eg “string”) of the argument.
Required	Yes/No this argument must be passed in.
Default	If the argument is not passed in, this is the default value.
Hint	Documentation about the argument.

Function with Arguments

```
<cffunction name="myDateFormat" returntype="string" output="false">
   <cfargument name="date" type="date" default="#Now()#">
   <cfreturn DateFormat(arguments.date, "dddd mmmm d, yyyy")>
</cffunction>
```

Arguments values are accessed as: arguments.*name*

Variables in Functions

- Variables that are only used inside the function are called "local" variables. They **MUST** be defined below your arguments using the **VAR** keyword.
  - Otherwise the variable is accessible outside of the function, you could overwrite another variable.
  - Code example.

Creating a CFC

- At their most basic level, a CFC is just a bunch of functions inside a <cfcomponent> tag.
    - When you store data in a CFC, then they become powerful.
    - You can have many instances of a CFC each having different data.

CFC Files

- CFC's are saved using the file extension .cfc the name of the file is also the name of the CFC.

City.cfc

```
<cfcomponent hint="I represent a city" output="false">
  <!--- data contained in the City Component --->
  <cfset variables.cityName = "Syracuse">

  <cffunction name="getCityName" returntype="string" output="false">
     <cfreturn variables.cityName>
  </cffunction>

  <cffunction name="setCityName" returntype="void" output="false">
     <cfargument name="cityName" type="string" required="true">
     <cfset variables.cityName = arguments.cityName>
  </cffunction>

</cfcomponent>
```

Invoking a CFC

- Three options:
  - CreateObject()
  - CFOBJECT
  - CFINVOKE

Invoking a CFC

```
<!--- CreateObject --->
<cfset cityObject = CreateObject("component", "City")>
<cfoutput>#cityObject.getCityName()#</cfoutput>

<!--- OR CFOBJECT --->
<cfobject component="City" name="cityObject">
<cfoutput>#cityObject.getCityName()#</cfoutput>


<!--- OR CFINVOKE --->
<cfinvoke component="City" method="getCityName" returnvariable="name">
<cfoutput>#name#</cfoutput>
```

Packages

- The folder that the .cfc file is located in is called it's "Package".
- If you are in the same folder/package you can reference other CFC's simply by their file name. Otherwise use package notation.
  - "folderA.folderB.MyCFCName" would be used for /folderA/folderB/MyCFCName.cfc

Packages

- You can place CFC's in the custom tag folder to have them accessible to all your sites.
- You can also setup mappings in the ColdFusion administrator.
    - "myMappingName.MyCFCName"

Example 4

Example 5

- Inheritance
  - Defines a relationship between two components. Allows you to "inherit" code from the parent.
- Must be an "IS A" relationship.
  - Book "is a" Product
  - City "is a" State (Not True)
    - City "has a" State

Using Inheritance

- Use **extends** attribute of cfcomponent to specify parent component name.
- Use **super** keyword to refer to parent component.
    - eg: *super.methodName()* to call a method on the parent (typically when overriding).

Interfaces

- New in CF8
- Defines a set of functions that must be defined. If the component defines all those methods it is said that it **implements** the interface.
- Unlike inheritance, you can implement many interfaces (you can only extend one component).
- Great For Building Component API's

Definitions

- Let's go back to the beginning and define some of those terms.

CFC's Vs Other OOP

- Constructors (announced for CF9)
- Destructors (not supported)
- Overloading (not supported)
- Multiple Inheritance (not supported, thankfully)
- Interfaces (now supported in CF8)

Thanks!

Any Questions??

blog: petefreitag.com