



MAX 2006 Beyond Boundaries




ColdFusion Components as Objects


Simon Slooten
Prisma IT, The Netherlands
simon@prisma-it.com
Oct 23-26 2006, Las Vegas


2006 Adobe Systems Incorporated. All Rights Reserved. 

Biography of Simon




- I am from Rotterdam, the Netherlands
(where the pilgrim fathers originally started their journey for America in 1620)
- 24 years experience in software development, consultancy & training
- Started Prisma IT in 1992
- Have been working with CF since 1995
- Adobe Certified Master Instructor




2006 Adobe Systems Incorporated. All Rights Reserved. 

This session




- Is about:
 - How to apply basic Object Oriented ideas and concepts to CFML development
 - Why to do that
- Is not about:
 - Advanced OO stuff
 - How to build the 'ultimate CFC'
 - 'the right' way of doing things
 - Comprehensive or complete

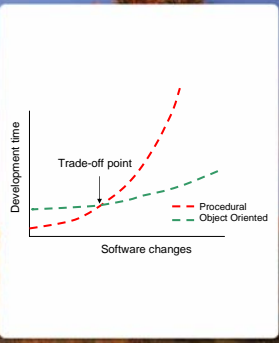
Software development is like economics: Every answer is correct, as long as you can justify your choices


2006 Adobe Systems Incorporated. All Rights Reserved. 

The maintenance problem




- A large proportion of the lifecycle cost of software is spent on maintenance
- The general consensus is that Maintenance Costs will decrease when OO-methods are used




2006 Adobe Systems Incorporated. All Rights Reserved. 

CFML ... Object Oriented?




- CFML is a Procedural language
- ColdFusion is written in JAVA
- JAVA is an Object Oriented language
- CFC's are CFML's implementation of Classes
- CFC's are not 'true - OO' but 'pseudo - OO'


(The majority of web development does not benefit from the differential between pseudo-OO and true-OO)

2006 Adobe Systems Incorporated. All Rights Reserved. 

Object Oriented vs Procedural



- Object orientation is a computer simulation of reality
- In an OO program, elements of the real world (e.g. a customer, an order, a product) are personified in a software abstraction known as an *object*
- An example: *The story of the paper boy.*

2006 Adobe Systems Incorporated. All Rights Reserved. 

The paperboy example

payPaperBoy.cfm

```
<cfquery datasource="myWallet">
  update myWALLET
  set money = money - 20
  where _
</cfquery>

<cfquery datasource="paperboy">
  update hisWALLET
  set money = money + 20
  where _
</cfquery>
```

payPaperBoy.cfm

```
myWallet.getMoney(20)

paperboysWallet.addMoney(20)
```

2008 Adobe Systems Incorporated. All Rights Reserved. 7

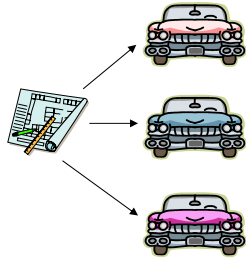
OO terminology (1)

- An object (or instance) is:
 - Collection of data (properties)
 - Color of my car
 - Make of the car
 - Type of engine
 - Etc.
 - Set of actions that can manipulate the data (methods)
 - closeDoor()
 - turnOnLight()
 - startEngine()
 - Etc.

2008 Adobe Systems Incorporated. All Rights Reserved. 8

OO Terminology (2)

- An Object Class is:
 - The 'blueprint' of an object
 - Defines characteristics for an object type
 - All objects in the Class would inherit the characteristics defined by the Class
 - But all objects have their own characteristics
 - My car is black,
 - has an automatic gear
 - And 4 doors



2008 Adobe Systems Incorporated. All Rights Reserved. 9

CFML Components

- A CFC is an 'Object Based Building Block'
- A CFC is the CFML equivalent of an Object Class
- A CFC can have
 - Associated Data → Properties
 - Associated Functions → Methods
- A CFC can not have a constructor (but can have a 'pseudo-constructor')

2008 Adobe Systems Incorporated. All Rights Reserved. 10

More terminology

- Constructor** - code to create an Object from a Class (more about this later)
- Instantiation** - creating an Object from a Class
- Inheritance** - I am what my parent is (a sportscar is a specialized car)
- Polymorphism** - I react differently than my parent (a sportscar is still a car)
- Encapsulation:** - I don't know anything about the outside world but you can ask me everything about me.

2008 Adobe Systems Incorporated. All Rights Reserved. 11


Even more terminology (the last one...I promise)

- Objects can be related in 3 ways:
 - Aggregation** (utilizes...)
 - Example: an object uses another object to validate values
 - Composition** (is a part of...)
 - Example: a car object contains a motor object
 - Inheritance** (is a...)
 - Example: a SportsCar object is a specialized Car object

2008 Adobe Systems Incorporated. All Rights Reserved. 12

A summary

- Procedural coding
 - Variables and code are 'all over the place'
 - All data is public (remember the paper boy story)
 - I (the programmer) need to know a lot about the data used in my app.
- Object Oriented coding
 - Variables and code are 'stored' together in objects
 - Data is private
 - I (the programmer) don't need to know where my data is stored and how to get to it



So much for the theory ... Let's get to the code

A CFML Component

- The simplest component is a library of functions with <cfcomponent> tags around it, saved as a '.cfc' file
- Like this it is not much more than an array of functions
- CFC's can be much more than that, they allow you to store both logic and data

```

<cfcomponent>
  <cffunction name="getMoney"
    returnType="boolean">

    <cfargument name="amount"
      type="numeric"
      access="public"
      required="true" />

    <cfquery ...>
      [SQL]
    </cfquery>
    <cfif ...>
      [check available money]
    </cfif>
    <cfreturn true/false>
  </cffunction>
</cfcomponent>
  
```

Static vs instance based Components

- CFML Components can be used in two ways
 - Static
 - Instance based
- 'Under the hood' CFML always creates an instance from a CFC

- A static component
 - is simply a container for functions
 - Logically group functions together
 - Self-inspection makes them easy to share
 - Can be "consumed" as web services by other clients - Flex, Flash etc.

- An Instance Based Component
 - Can have data associated with them
 - Keeps data logically tied to an object
 - Supports more OO - type of programming using object.property and object.method() notation
 - Methods can share the same data.

Invoking a static component

```

<cfinvoke
  component = "myApp.components.myWallet"
  method = "getMoney"
  returnvariable = "bSuccess">

  <cfinvokeargument
    name = "amount"
    value = "20">
</cfinvoke>
  
```

Creating an Object instance

- an 'object' (instance) is 'constructed' from the CFC (Class)


```

<cfobject component="componentName" name="instanceName">
or
<cfset instanceName = createObject("component", "componentName")>
      
```
- the scope of the object determines how long the object 'lives'


```

<cfset Application.instanceName =
  createObject("component", "componentName")>
      
```
- can have associated data (properties)
 - because you now have an Instantiated Component (that lives in memory), you can add data to that object. In other words: you now have an Object that can have Properties.
 - and...you probably want those properties to be 'hidden' from the outside world

Populating Properties



- We need to 'populate' the properties when the Object is constructed, but..
- A CFC can not have a constructor

- 2 solutions for this 'problem'
 - A CFC has a 'pseudo-constructor'
 - All the code between the <cfcomponent> and the first <cffunction> tags
 - A more general used method is to create an init() method that returns a populated copy of the object



The pseudo constructor



```
<cfcomponent>
  [constructor code]
<cffunction name=...>
</cffunction>
</cfcomponent>
```



The init() method



```
<cfcomponent>
  <cffunction name="init" access="public" returnType="Car" output="false">
    <cfargument name="dsn" type="string" required="yes">

    <cfset variables.datasource = arguments.dsn />
    <cfset variables.make = "" />
    <cfset variables.model = "" />

    <cfreturn this />
  </cffunction>
  [more methods and/or getters & setters]
</cfcomponent>
```



Creating the Object



```
<cfset Session.myCar =
  createObject("component", "myApp.components.Car").init("myDSN") />
```



A brief look at some methods



```
...
<!-- A setter -->
<cffunction name="SetMake" access="public" output="false" >
  <cfargument name="Make" required="true" type="string">
  <cfset variables.Make = arguments.Make />
</cffunction>

<!-- A Getter -->
<cffunction name="GetMake" returnType="string" access="public"
output="false" >
  <cfreturn variables.Make />
</cffunction>
...
```



What have we done so far?



- We have now created an Object 'car', with properties 'Make' and 'Model'
- Because the property is created in the variable-scope of the Object, it is not accessible from the outside world → Private Properties
- To get it or to set it, we need functions (getters & setters).

BUT...

- Why not just make the properties 'public', like this?:
<cfset this.make = arguments.make>
- Which would mean that (from outside the Object) we could do this:
<cfset car.make = "Ferrari">



Why Private properties?



- Public properties can be accessed (and set) from outside the Object.
- Who is in charge of validation and integrity?
- Adding validation would mean we have to search through our code
- With Private Properties we only need to change the setter for that property

2006 Adobe Systems Incorporated. All Rights Reserved.

25



Some words about inheritance



```
<!-- SportsCar.cfc -->  
<cfcomponent extends="Car">
```

- Car.cfc is used as the 'foundation' for the SportsCar Object
 - Properties and Methods can be added
 - Existing Properties and Methods (in Car.cfc) can be overwritten
- Why?
 - It makes life easier
 - Let's say we want to add a navSat property ... when we add that to Car.cfc all objects that extend Car automatically get the property

2006 Adobe Systems Incorporated. All Rights Reserved.

26



What we have achieved



- We now have an **Object** (application.myCar)
- The Object holds all data (**properties**) about my car
- The Object has all functions (**methods**) to maintain the data
- Because the object 'lives' in the Session scope, we can invoke methods and get or set properties from any template in my application
- If something needs to change, I only need to change my car.cfc component
- It would be pretty easy to use Car.CFC in another application too

2006 Adobe Systems Incorporated. All Rights Reserved.

27



Other topics



- The 'var' keyword
- The 'this' scope
- The 'super' keyword
- Methodologies and/or frameworks

2006 Adobe Systems Incorporated. All Rights Reserved.

28



Other topics



- The 'var' keyword
 - Makes variables local to a method (function)
- The 'this' scope
 - The Object's scope. Variables created in the this-scope are public properties
- The 'super' keyword
 - Can be used to access an overloaded method of an extended Class
- Methodologies and/or frameworks
 - Many methodologies and frameworks are available, much discussion about them...ask me afterwards

2006 Adobe Systems Incorporated. All Rights Reserved.

29



Wait..there is more!



- Methods of CFC's can easily be turned into webservices:

```
<cffunction access="public" ..>
```
- CFC's can be invoked from Flex and Flash applications using Remoting
- (FlexBuilder 2 has a VERY cool feature to create CFC's from ActionScript Classes and vice versa)


2006 Adobe Systems Incorporated. All Rights Reserved.

30




MAX

Stuff I found while creating this presentation



- "From a list: 'They [CFC's] are just an array of functions you call from a separate file.' Whoa! CFCs are a lot more than a library of functions - they open ColdFusion to object-oriented thought.' (Joe Rinehart's website)
- 'It seems like some people tend (at times) to go a bit overboard with OO' (Daemon weblog)
- STTCPW

2006 Adobe Systems Incorporated. All Rights Reserved. 33




MAX

Where to go from here

- Adobe Official Curriculum (www.adobe.com/training)
- CFCdev mailing list (www.cfczone.org)
- ColdFusion Developers Journal
- www.corfield.org
- [Weblogs.macromedia.com](http://weblogs.macromedia.com)
- Other MAX 2006 sessions
- <http://java.sun.com/docs/books/tutorial/java/concepts/>


2006 Adobe Systems Incorporated. All Rights Reserved. 32



MAX

Thank you!
simon@prisma-it.com

2006 Adobe Systems Incorporated. All Rights Reserved. 33



MAX

Better by Adobe™

2006 Adobe Systems Incorporated. All Rights Reserved. 34

