I bet you didn’t know you could do that with ***ColdFusion!!***

Elliott Sprehn
elliott@teratech.com
http://www.elliottsprehn.com/blog/

What *is* ColdFusion?

- A set of languages (CFML,CFScript)
- A runtime environment
  - An interpreter for expressions
- Language Model
  - Objects
  - Functions
  - Types

ColdFusion has many very cool “features”

- First class date values
- First class Functions
- Dynamic Object Model
  - Runtime defined object types (no classes)
  - OnMissingMethod (CF8)
  - Not sealed

ColdFusion Date Values

- Most transparent type available.
- Valid Dates (January 1st 2008):
  - “January 2008”
  - “1 2008”
  - “1/1”
  - “Jan 1”
  - “2008 1”
  - "{ts '2008-01-01 00:00:00'}"
  - 39448

Dating Loops

- ColdFusion allows looping over dates.
  - Is there a bug with the second code snippet?

Examples:

```
<cfloop from="January 1st 2005" to="January 31 2005"
index=”date”>
</cfloop>

for( d = now(); d lt dateAdd("w",1,now()); d = d + 1) {
}
```

Why do date loops matter?

- Makes for VERY clean code.
  - *We’ll get to an example a bit later...*

Functions

- First Class Values
- Must have unique names... or do they!?
  - **Trick:** Can structDelete(variables,"func") and cfinclude a template with "func" to replace it.
- Allow arguments
  - Named, positional.
  - Allow Extra Arguments.

Arguments against Arguments

- Passing named arguments is EASY.
  - func(a=1, b=2)
  - func(argumentCollection=struct)
- Passing positional arguments is HARD.
  - func(1, 2)
  - ~~func(argumentCollection=array)~~ ???
  - **Tricks:**
    - <cfinvokeargument name=”1” value=”1”>
    - evaluate(“func(args[1],args[2],args[3])”)

MetaData

- Component MetaData
  - getMetaData(instance)
  - getMetaData(function)
  - getComponentMetaData(name) (CF8)
- Belongs to component or function.
  - Static (In the Java Sense)
  - Disappears if the component is recompiled.
    - Must be initialized at definition.
  - What about <cfproperty> ?

Uses of Component Static Variables

- Component Static Variables
  - Initialize inside the <cfcomponent> body.
    - **Make sure to lock!**
- Shared Dependencies Between Components
  - Can share variables, collections, arrays...
  - “Compute once” values like lists of files inside a package.
    - Removes the need for extraneous factory patterns.
- Implicit Singletons

Implicit Singletons

- Object is transparently a singleton.
  - No refactoring costs.
  - Natural looking code.
- Pattern used in other languages.
  - Ruby / Python / Perl
- How?

```
function init() {
   var static = getMetaData(this).static;
   if( not structKeyExists(static,”instance”) )
      initSingleton(); // locked internally
   return static.instance;
}
```

Uses of Function Static Variables

- Annotation like data. (ex. methods=”POST”)
- getMetaData(func).static
  - Careful not to use the attribute “static” on the function.
- Creating Closures.

Closures!

- No, not Openures. Must be closed.
- Function with associated context.
  - Used frequently in other languages.
- **Can this be done in ColdFusion?**

Ruby:

```
File.open(“myfile.txt”,”) do |f|
  f.puts “Line 1”
  f.puts “Line 2”
end
```

Function Pointers and Contexts

- Functions can be referenced by name.
  - <cffunction> both defines a function and assigns a variable.
- Can be aliased
  - variables.aliasName = func
- Binds to calling context.
  - Components are exception.
- Can we store a different context for invocation to implement Closures?
  - Yes! ***MetaData to the rescue!***

Other Interesting ColdFusion Features

- OnMissingMethod (CF8)
  - Can build prototype Objects like JavaScript.
  - **More on this in a minute...**

- Can impersonate ColdFusion CFC types
  - Use with care, “great power... great responsibility...”

```
getMetaData(this).name = “com.other.component.Name”
```

Prototype Objects

- Build an Object as a chain of Objects.
  - **Dynamic, shared, inheritance.**
  - JavaScript uses this Model.
- Methods are looked up by traversing the Prototype Chain.
- Methods are overridden by be placed lower on chain.
- **Can this be done in ColdFusion?**
  - **YES!**

Powered by OnMissingMethod

- How?
  1. Use OnMissingMethod to trap methods that don't already exist on the target Object.
  2. Travel up the prototype chain to find the right method.
  3. Once found invoke it with the context of the current object.
- **Isn’t this slow?**
  - **Nope!** Only first invocation requires lookups.
  - Make method “real” once found in chain.

Prototype Code Example

Prototype!

Questions

?