

# Intro To Model View Controllers

Nolan Erck  
NCDevCon, September 2011

# What We'll Cover Today

- An introduction to the “Model View Controller” pattern.
- Some pros and cons.
- All information is non-framework specific.
- A few slides to start, and get everyone on the same page, then we'll look at lots of code, I promise!

# About Me

- Independent Consultant ([SouthofShasta.com](http://SouthofShasta.com)).
- SACCFUG Co-Manager.
- Software development for 15 years (CF since 4.5).
- Previous projects: Grim Fandango, SimPark, SimSafari, [Riffage.com](http://Riffage.com), [iConvention.com](http://iConvention.com).
- Working musician.

# Model View Controller

- Not ColdFusion specific.
- Common design pattern used in other OO languages.
- Used within CF frameworks (e.g. Model-Glue, ColdBox, etc).

# Before We Get Started...

- Everyone should know...
- CFComponent
  - How to create and use one.
- Bean
- DAO
- Gateway

# CFComponent

- ColdFusion's OO construct.
- Same as a “Class” or “Object” in Java, C++, etc.
- Other calling conventions (cfscript, cfinvoke, cfobject) but we'll skip those for now.
- *Typically* includes an init() method for setup (though that's not required).

# CFComponent

- Employee.cfc:

```
<cfcomponent>
```

```
    <cfset variables.empName = "" />
```

```
    <cfset variables.employeeID = "" />
```

```
    <cffunction name="doSomething">
```

```
        <!--- code goes here --->
```

```
    </cffunction>
```

```
</cfcomponent>
```

# CFComponent

- And you use it like so:
- `<cfset myBoss = CreateObject( "component", "Employee" ) />`
- `<cfset myBoss.EmployeeID = 557 />`
- `<cfset myBoss.doSomething() />`

# Beans

- The OO version of a “struct”.
- Hold related data.
- And a few functions to get/set that data.
- But not a whole lot else.
  - Validation code might live here.
- Let's look at Customer.cfc...

# DAOs

- Data Access Object.
- Used for, well, accessing the data.
- CRUD operations (1 record).
- Let's look at CustomerDAO.cfc...

# Gateways

- Like DAOs, but for “multiple record” operations.
- GetAll() or search().
- Let's look at CustomerGateway.cfc...

# What is Mode View Controller?

- Design Pattern for building software.
- Not CF-specific.
- Lots of this info transfers to other OO languages just fine.
- Basically a way of organizing and calling your code to “separate the concerns”.
  - Display code, business logic, and data storage

# Definition – Model

- Short for “data model”.
- Where all (yes all) of your SQL code lives.
- DAOs, Gateways, and Beans are part of the Model.
- Doesn't have to be a database.
  - Whatever your storage medium is for data.
    - Log files, XML, etc.
- Some business logic may live here too.

# Definition - View

- The part of the app that users, well, view.
- All HTML, CSS, JavaScript for any visible output.
- Some CF code for display logic, but that's it.
  - e.g. Code to toggle background colors of rows in a table.
- No business logic.
- No SQL code.
- Similar to “dsp\_” files in Fusebox apps.

# Definition - Controller

- Sits between the Model and the View.
- Some business logic lives here.
- No HTML output is done here.
- Just calls to `<cflocation>`, “controlling” where the user goes next.

# Model View Controller

- Model

- DAO
- Gateway
- Bean
- All SQL
- Business Logic

- Controller

- Glues the View and Model together.
- Business Logic.
- “Controls” where the user goes next.

- View

- HTML
- JavaScript
- CSS
- UI related logic, but no business logic.
- No SQL!

# log\_entry\_old.cfm

- Let's look at the log\_entry\_old.cfm sample.
  - Original CFM template.
  - Looks a lot like code we've all seen (and written) at some point.
  - Everything in 1 file.
  - No reuse, no modularity.
  - Only 1 developer can work on it at a time.
  - No ability to swap out anything.
    - What if we want to build a “mobile” version?
    - Have to copy/paste all the queries and logic.

# log\_entry.cfm

- Now let's look at log\_entry.cfm
  - A refactored version.
  - This is the View.
  - Note `<form post="">` posts to a Controller CFC.
  - Also `<input name="method">` tag.
    - Name of the method in our Controller that we're calling.
  - Now only the HTML is in the file.
    - Easy to swap in a new UI without touching SQL or business logic.
  - What if we want to build a Flex UI? Easy!

# TravelLogController.cfc

- Not much going on here.
- saveLogEntry() method:
  - Takes the fields from the form post.
  - Does some minor validation.
  - Passes the fields to the Model for saving.
  - Then “controls” where the user goes next (back to the page w/ a “saved successfully” message).
  - (Could also direct the user back to list.cfm, or a generic “an error has occurred” page).

# Our Model Code

- Nothing new or different here.
- TravelLog.cfc:
  - Bean is the same as a struct (basically).
  - Just “wraps” the data.
  - No moving parts (except maybe validation).
- TravelLogDAO.cfc:
  - Handles saving our log entry to the database.
- TravelLogGateway.cfc:
  - Handles some other SQL functionality for us (things meant for multiple records. DAO is just 1 record).

# Our Model Code (cont)

- Illudium PU36 Code Generator is your friend.
- 99% of the Model code in this app was auto-generated!
  - I made a few hand-edits for simplicity in the demo (but could have modified the Illudium template instead).
- Barely any typing was needed for the Model code.

# Pros

- Promotes code reuse.
- Allows multiple people to work on the same section of the app at the same time:
  - 1 works on the UI (View)
  - 1 works on the business logic (Controller).
- Non-framework, non-language specific.
- Very common pattern/nomenclature.
  - “Model” means the same thing in Java, C++, etc.

# Cons

- A change in style from procedural programming.
  - May take time to “click”.
- “More typing” to get the 3 layers up and running.
  - But the code is more reusable.
  - And as we've seen, PU36 can cut down that typing to where it's really a non-issue.

# Using a Framework

- MVC frameworks all kind of work the same way.
- ...because they're all using the same *design pattern!*
- Model-Glue, ColdBox, Mach-ii, FW/1 all have a place to put “views”, “controllers”, and “models”.
- Only difference is a little syntax and calling convention stuff. Nothing crazy.
- Let's look at [ncdevcon-mg.dev](http://ncdevcon-mg.dev)

# A few last thoughts

- OO is hard!
- That's normal.
- *Nobody* instantly knows this stuff the first time.
- But it does eventually make building large apps simpler.
  - Keeps you organized.
  - Gives you common terminology to talk to other developers about the problem, split up the work, etc.

# Other Resources

- Book: “Head First Design Patterns”
- Book: “Object Oriented Programming in ColdFusion”
- [ColdFusionBloggers.org](http://ColdFusionBloggers.org)
- CFCDev mailing list.
- BACFUG mailing list.

# Thanks!

- Blog: [southofshasta.com/blog](http://southofshasta.com/blog)
- Email: [nolan@southofshasta.com](mailto:nolan@southofshasta.com)
- Twitter: [@southofshasta](https://twitter.com/southofshasta)