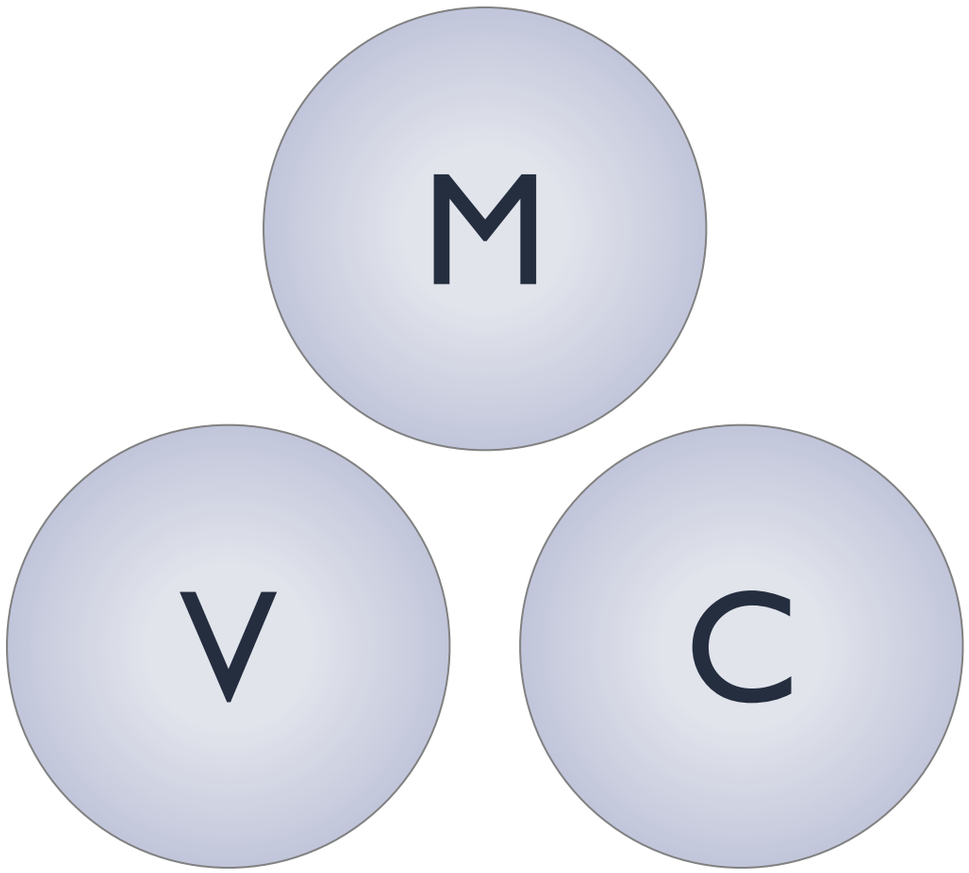
HOTEL DELTA

Introduction to Framework One

info@hoteldelta.net

What is FW/1?



What FW/1 is NOT

ORM/Object Model

Scaffolding or other CRUD Help

Dependency Injector

Remoting Automagicalizer

Event Management, Logging, Caching, . . .

The FW/1 Philosophy

Convention

over

Configuration

No XML!


alias="m" path="m
uit alias="v" path="views
rcuit alias="app" path="contr
its>
lass
<cl ias="tester" classpath="class
</classes
<parameters>
<parameter "defaultFuseaction" valu
<!-- you may to change this to devel
<parameter nam e" value="production"
<!-- change this mething more secure
<parameter name="p d" value="skelet
<!--
These are all defa alues that ca
<parameter name="fuseact riable" va
<parameter name="precedent OrUrl"
<parameter name="scriptFile ter"
<parameter name="maskedFileDe
parameter name="characterEncod
ramater name="strictMode" val
eter name="allowImplici

Why Prefer Convention?

Less irritating to develop

No XML means no need to do something,
then tell the XML that you did it.

Bonus: one fewer place for bugs to creep in.

Why Prefer Convention?

Encourages better decisions

Baked-in conventions driven by community standards encourage consistent & rational code structure, force clean separation of concerns and promote shared, well-vetted practices.

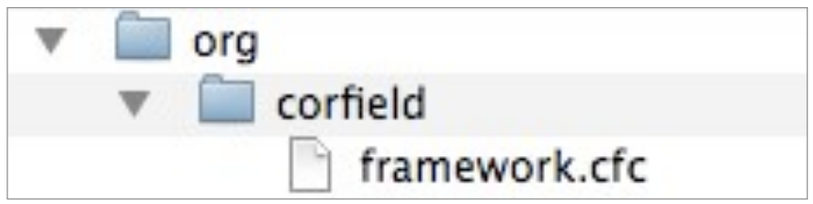
Why Prefer Convention?

Fewer maintenance headaches

Predictable structure lowers the learning curve and generates less code to wade through when diagnosing problems and making changes.

Installing FW/1

1. Put framework.cfc in a sensible place.



(or in the root of your app)

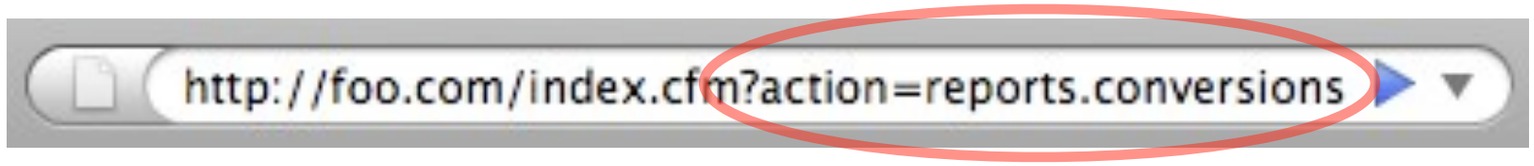
2. Extend framework.cfc in your Application.cfc.

```
<cfcomponent extends="org.corfield.framework" output="false">
```

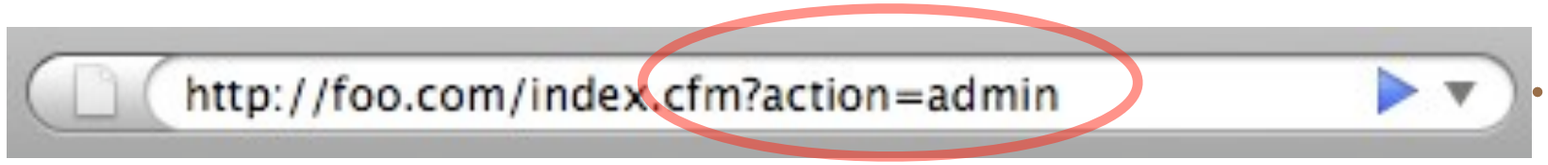
3. Done.

Convention: Sections & Items

"Action" Defines a Section.Item

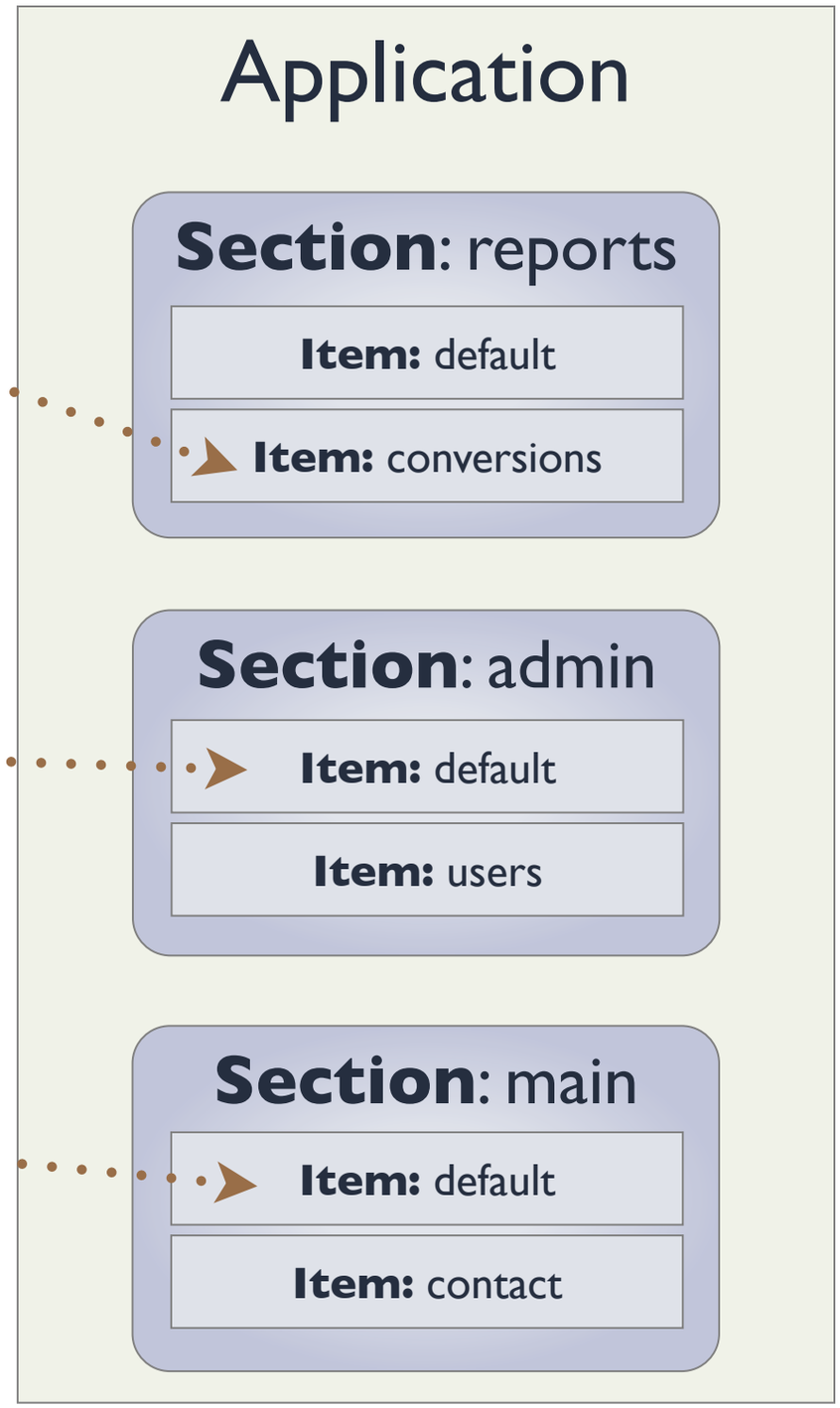


Request for section "*reports*" with item "*conversions*"



Request for section "*admin*" with default item ("*default*")

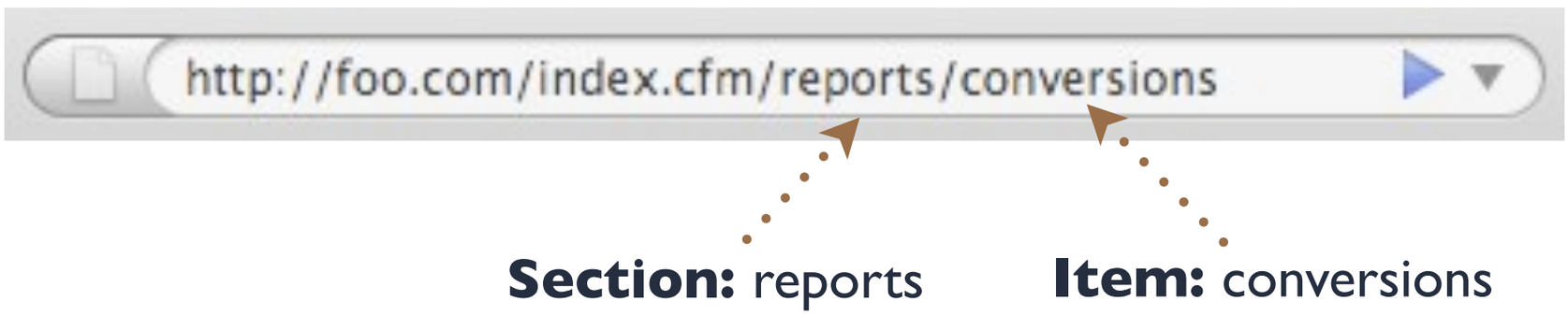
Request for default section & item ("*main.default*")



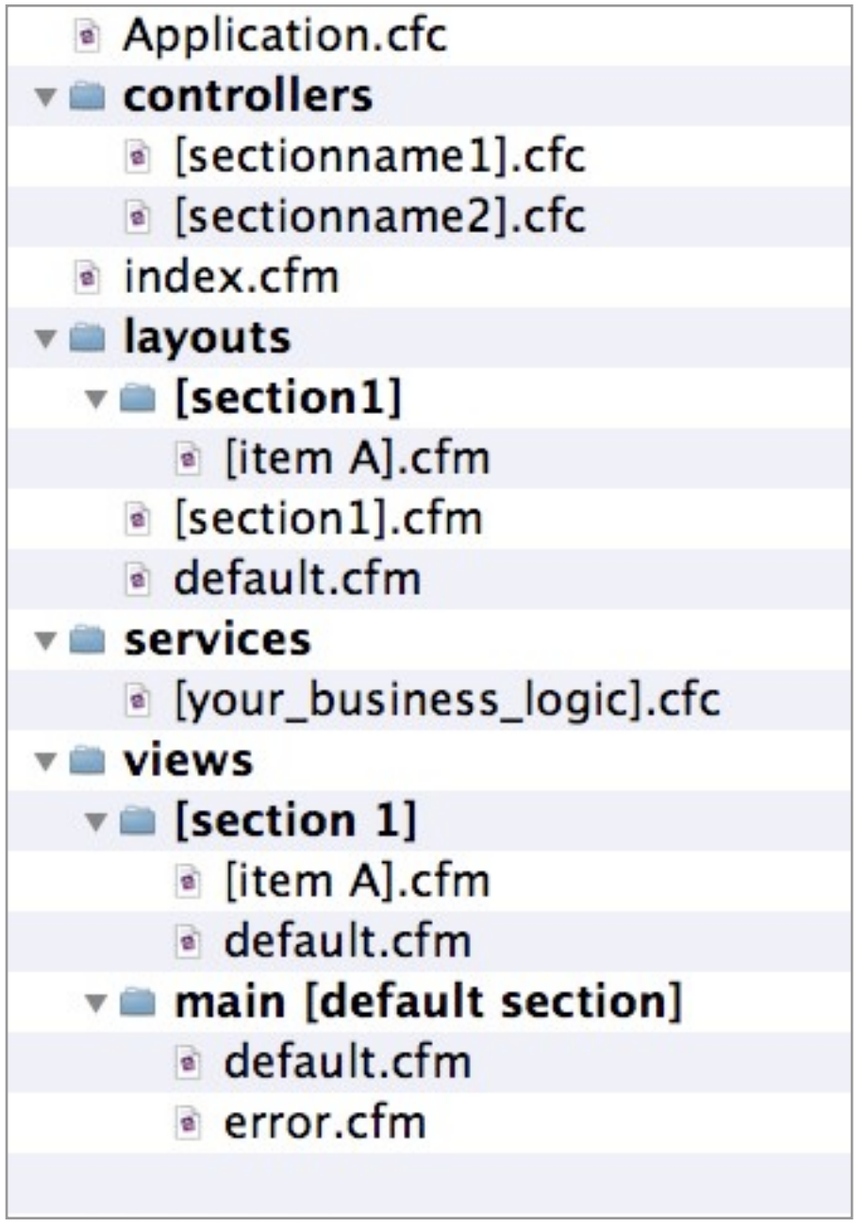
SES URLs

Out-of-the-Box Support for "Search-Engine-Safe" URLs:

(first two are assumed to be SECTION & ITEM - others treated as query string vars)



Convention: File Structure



Controllers

Named per *section* with *item* as method names

Views

Folder per *section* with files named for *item*

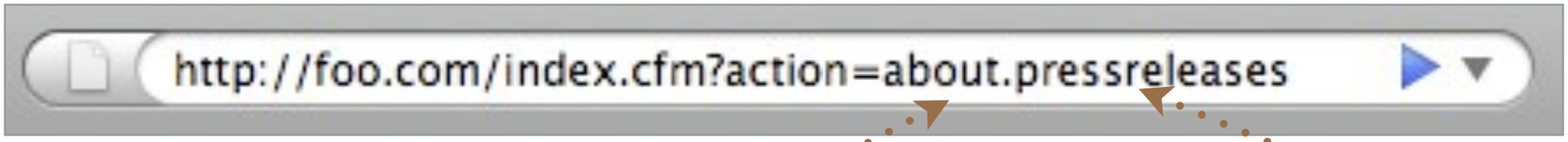
Layouts

Global default layout + *section*-named files + *item*-named files in *section*-named folders (layouts cascade up from most specific to the global).

Services

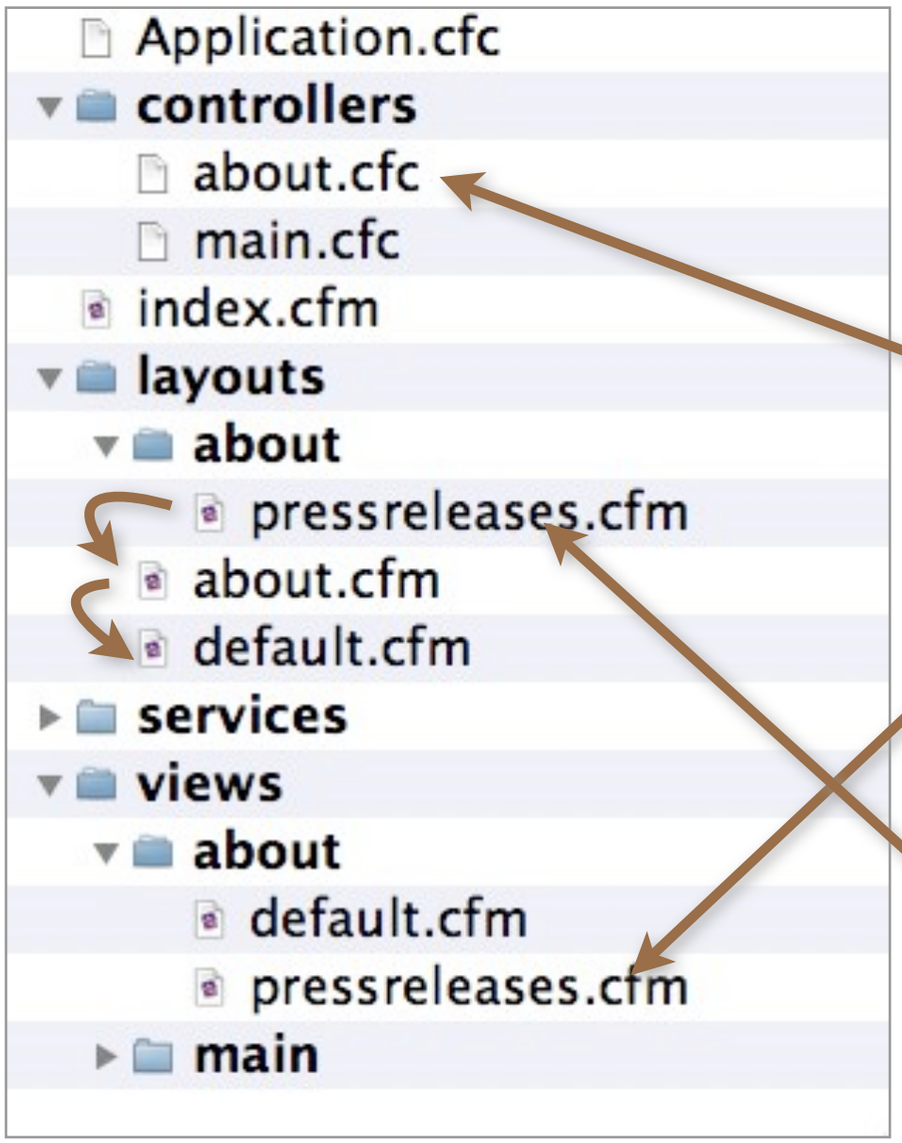
When using implicit service invocation (which has become somewhat unfashionable), naming scheme is same as controllers - otherwise, whatever you want.

File Structure: Simple Case



Section: about

Item: pressreleases



Controller

Method `pressreleases()` called in the controller `about.cfc`

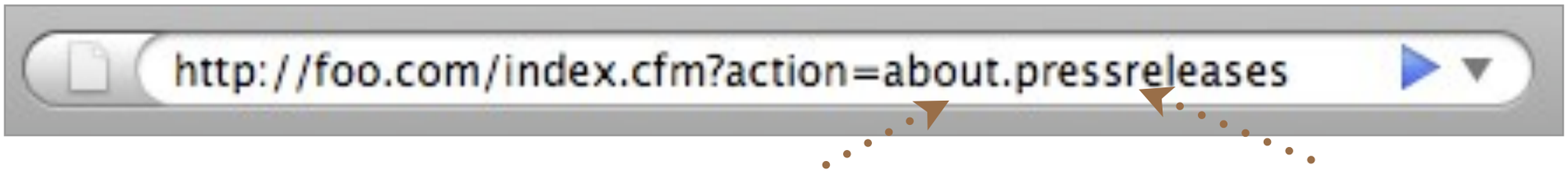
View

`pressreleases.cfm` in the folder `about`

Layout(s)

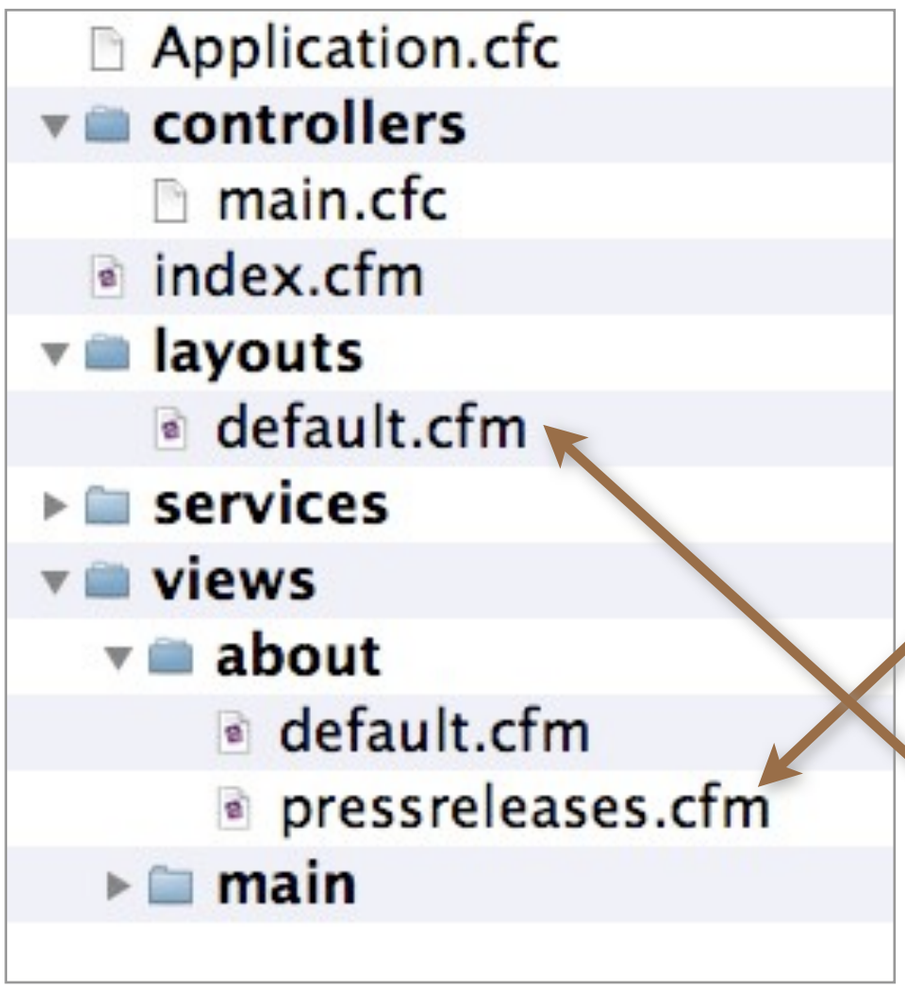
Output of `pressreleases.cfm` in the folder `about` is wrapped in `about.cfm` is wrapped in `default.cfm`

File Structure: Even Simpler Case



Section: about

Item: pressreleases



Controller

None.

View

`pressreleases.cfm` in the folder `about`

Layout(s)

`default.cfm`

Convention: request.context

Main "Data Bus"

Used for passing information from controllers to views to layouts

Incorporates FORM and URL Vars

Automatically grabs these at the start of the request - FORM takes precedence

Shorthand in Most Places: "RC"

All controllers are passed the argument `rc` containing the `request.context`, and all views/layouts have access to `rc` as well.

Convention: Controller Methods

Order of Execution of Controller Methods

(all optional)

0. **init**(fw)*
1. **before**(rc)
2. **start*item***(rc)
3. ***item***(rc)
4. **end*item***(rc)
5. **after**(rc)

By convention, use start*item*/end*item* OR *item* - not both (usually start/end is used in conjunction with implicit services)

* called once on instantiation (instance is then cached)

Simple Case of Request Lifecycle*

action=about.pressreleases

Application.cfc (extends org.corfield.framework)

`[setupApplication(),] [setupSession(),] setupRequest()`

↓

/controllers/about.cfc

`about.before(rc), about.pressreleases(rc), about.after(rc)`

↓

/views/about/pressreleases.cfm

(accesses `rc` and has its own `local` scope)

↓

/layouts/about.cfm

(accesses view content as `body` - can access `rc`)

↓

/layouts/default.cfm

(accesses inner layout as `body` - can access `rc`)

* doesn't include implicit services, subsystems, or manually queued controllers/services

Manual Controller Calls

controller()

The `controller()` method can be used inside Application.cfc to queue calls to controllers other than those specified by the request's action. All of the implicit methods will also be called for that controller item.

For instance, you may want to perform authorization or some other kind of global routine in `setupRequest()`

You are queuing, not "calling"

Keep in mind you are not actually calling a method on a controller when you use the `controller()` method, instead you are adding it to the queue of controllers to execute.

You cannot call controller() in a controller

Once execution of the controller queue has started you cannot call `controller()`.

Services

(the most "controversial" feature of FW/1)

"Unaware" of FW/1

Services are intended to be agnostic of and decoupled from FW/1 conventions.

Implicit vs. Explicit

Version 1.x of FW/1 called services implicitly by default, 2.x will not (fashion seems to lean against implicit service invocation). Implicit calls can be turned off with a true value for the "suppressImplicitService" configuration option.

Using service()

Inside controllers you can use the service() method to manually queue calls to service methods, passing the results back into the request.context.

Manual, synchronous calls to services

Some prefer to manually call service methods in controllers using the beanFactory interface to get at their services. Some prefer such business logic components to exist in a separate "model" folder to prevent confusion with implicit services.

Choosing The View(s)

(other than the default behavior of looking for a file based on the action)

setView(“*section.item*”)

Called inside controllers to manually set the view.
If called, FW/1 will skip looking for the file based on the action.

onMissingView()

Global method in Application.cfc that will handle ALL missing views (warning: also handles an error if no error view has been specified).

view(“*section/item*”[,*args*])

Called inside views and layouts to include other views. If second argument is present it is passed as the “local” scope to the included view.

variables.framework{}

```
variables.framework = {
  action = 'action',
  usingSubsystems = false,
  defaultSubsystem = 'home',
  defaultSection = 'main',
  defaultItem = 'default',
  subsystemDelimiter = ':',
  siteWideLayoutSubsystem = 'common',
  home = 'main.default',
  error = 'main.error',
  reload = 'reload',
  password = 'true',
  reloadApplicationOnEveryRequest=false,
  generateSES = false,
  SESOmitIndex = false,
  baseURL = 'useCgiScriptName',
  suppressImplicitService = false,
  unhandledExtensions = 'cfc',
  unhandledPaths = '/flex2gateway',
  preserveKeyURLKey = 'fw1pk',
  maxNumContextsPreserved = 10,
  cacheFileExists = false,
  applicationKey ='org.corfield.framework'
};
```

Set in Application.cfc

Sets global configuration options for FW/1, allows easy over-riding of default conventions, but be careful doing that unnecessarily.

A Couple Highlights:

reloadApplicationOnEveryRequest

Determines if FW/1 will cache controllers/services - true while developing, false in production (probably want to set dynamically based on deployment config).

suppressImplicitService

Defaults to false in FW/1 1.x, but will default to true in FW/1 2.x. Unless you KNOW you want implicit services, turn it off.

Other Handy Features

redirect()

Acts as a "smart" CFLOCATION, preserving request context if needed.

buildURL()

Easy way to create URLs for links and images (etc.) based on a FW/1 action - sensitive to whether you're using SES or not.

onMissingMethod() in Controllers

Inside controllers you can define this to react to all calls from FW/1 - warning: that includes all calls to before(), after(), start*item()*, end*item()* - so, when using onMissingMethod you should probably define before() and after() even if empty.

request.layout

You can stop the layout cascade by setting `request.layout = false` in any layout or view.

Topics Not Covered Here

Subsystems

Stand-alone FW/1 apps, all running under one roof.

Auto-Wiring/Bean Factories

You can use ColdSpring or other frameworks to auto-wire your Controllers/Services

customizeViewOrLayoutPath()

Create separate, parallel layout/view packages that can be called dynamically (great for "skins")

populate(foo)

Calls all found methods in foo matching set*X*(), where *X* is a variable in the request.context

https://github.com/seancorfield/fw1/wiki/

Photo Credits

- Feather: http://www.flickr.com/photos/n0rthw1nd/4418311590/

HOTEL DELTA

info@hoteldelta.net