Headstrong

2008

Flex

Cairngorm Framework

Arunkumar K

JP Techno Park, #3/1 Millers Road, Bangalore - 560052

Contents

[1. Getting Started with Cairngorm – Part 1 2](#_Toc210301228)

[2. Getting Started with Cairngorm – Part 2 5](#_Toc210301229)

[3. Getting Started with Cairngorm – Part 3 10](#_Toc210301230)

[4. Getting Started with Cairngorm – Part 4 16](#_Toc210301231)

[5. Getting Started with Cairngorm – Part 5 25](#_Toc210301232)

Ref: <http://www.davidtucker.net/category/cairngorm/>

# [Getting Started with Cairngorm – Part 1](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)

**October 7th, 2007**

I spoke about Cairngorm 2.2 in the Flex Bootcamp at Max this week. Many people were interested in Cairngorm, but I only had about 10 minutes to explain the basics of Cairngorm. I guess the easiest way to assist these people is to do a quick blog series on the benefits of Cairngorm. This series will combine articles with "code-along" videos.

**Disclaimer:** *I do not claim the be "the expert" on Cairngorm - I am far from it. However, I have used Cairngorm on several large projects (both at Georgia Tech and in my own business). I am certainly open to corrections if you see that I have made an error on this project. If you want "the experts" check out:* [*Steven Webster*](http://weblogs.macromedia.com/swebster/)*,* [*Alistair McLeod*](http://weblogs.macromedia.com/amcleod)*,* [*Alex Uhlmann*](http://weblogs.macromedia.com/auhlmann)*, and* [*Peter Martin*](http://weblogs.macromedia.com/pmartin)*.*

**Note:** *The guys at Adobe Consulting (that developed Cairngorm) are currently investigating some new things with the framework as a whole. It is possible (actually probable) that some of these things will change in the future. One of the very specific areas of change is the Model Locator.*

**Part 1 – Getting Started By Using a Model Locator**

The Model Locator pattern is used in Cairngorm, but you don't have to have a full Cairngorm Implementation to use the pattern. First, let's cover what benefits that you get from using a Model Locator.

A Model Locator is a centralized repository for all of the data that is needed across your application. Your data will exist inside of a "singleton class". This "class" can only have one instance of itself. Why is this important? Let me give you an example.

We have great mini-notepads at work that I use to write down data while I work. However, sometimes, I lose a notepad – get a new one, and then find the old one. After both have been heavily used – it is really hard to find out which notepad I used to write down a piece of information from one week ago. That is a simple example – but imagine if I had 20 notepads? This could get crazy.

In the same way you could have a "class" that gets "instantiated" 20 times within your application (even if you don't mean for it to). The easiest way to eliminate this problem is to use a "singleton". A singleton is a class that is never "created" in the traditional way. It has one main rule – no more than one of itself can exist at any point in time. How does it do this? I will show you in the Model Locator example.

[PLAIN TEXT](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)

**Actionscript:**

1. package net.davidtucker.CairngormSample.model {
3. import com.adobe.cairngorm.model.IModelLocator;
5. [Bindable]
6. public **class** ModelLocator implements IModelLocator {
8. *// Single Instance of Our ModelLocator*
9. private static **var** instance:ModelLocator;
11. public **function** ModelLocator(enforcer:SingletonEnforcer) {
13. if (enforcer == **null**) {
14. throw **new** Error( "You Can Only Have One ModelLocator" );
15. }
16. }
18. *// Returns the Single Instance*
19. public static **function** getInstance() : ModelLocator {
21. if (instance == **null**) {
22. instance = **new** ModelLocator( **new** SingletonEnforcer );
23. }
24. return instance;
25. }
27. *//DEFINE YOUR VARIABLES HERE*
29. }
30. }
32. *// Utility Class to Deny Access to Constructor*
33. **class** SingletonEnforcer {}

This code may look a bit daunting in the beginning, but trust me that it is not as difficult as it may appear. First, we have our package definition and we import some classes. Right now we know that we will need the IModelLocator interface. To use this you will need the Cairngorm SWC that can be found here: [Cairngorm](http://labs.adobe.com/wiki/index.php/Cairngorm:Cairngorm2.2:Download). Also note - you could build a model locator without Cairngorm, and I do this frequently on small projects (you just leave out the 'implements IModelLocator' and 'import com.adobe.cairngorm.model.IModelLocator' code from the Model Locator).

[PLAIN TEXT](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)

**Actionscript:**

1. [Bindable]
2. public **class** ModelLocator implements IModelLocator {
4. *// Single Instance of Our ModelLocator*
5. private static **var** instance:ModelLocator;

Next we have our class definition. It is important that you use the Bindable metatag directly above your class definition. This will allow all of our variables that we define inside of the Model Locator to be used for binding. We also will go ahead and create one variable. It will be called "instance" and it will be of type ModelLocator. This will be the variable where we will store our one instance of our class. It will also be denoted as a "static" property. If you are not sure what a "static" property is, it's ok - we will discuss that in our next lesson.

[PLAIN TEXT](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)

**Actionscript:**

1. public **function** ModelLocator(enforcer:SingletonEnforcer) {
2. if (enforcer == **null**) {
3. throw **new** Error( "You Can Only Have One ModelLocator" );
4. }
5. }

This is followed by the constructor. The constructor takes on argument - "enforcer". You will notice that this "enforcer" has a type of "SingletonEnforcer" which is defined directly after our class. Here is the logic behind that:

* When you put a class in an Actionscript file below the main class, it is only available to that class. Many people refer to these as "utility classes" (although many people use that term in a much broader scope).
* If the constructor requires this argument - then only our main class can create an instance of itself, because we do not have access to the "SingletonEnforcer" class - only the main class has this access.
* We will not access our class in the normal way by using the "new" statement because we can't call the constructor (I will show you how we will do it in a bit).

Once we get inside of the constructor, we have a few lines that make sure things work as planned. The "if" statement ensures that we had a valid "enforcer" passed in. If there wasn't it throws an Error stating that "You Can Have Only One ModelLocator".

[PLAIN TEXT](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)

**Actionscript:**

1. *// Returns the Single Instance*
2. public static **function** getInstance() : ModelLocator {
3. if (instance == **null**) {
4. instance = **new** ModelLocator( **new** SingletonEnforcer );
5. }
6. return instance;
7. }

The "getInstance" function is how we will access our ModelLocator from our application. This function simply passes back the "instance" of the class. If it doesn't exist yet, it creates it. We can now get the ModelLocator by using the following code:

[PLAIN TEXT](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)

**Actionscript:**

1. **var** model:ModelLocator = ModelLocator.getInstance();

**Video Example - Getting Started and Building a Contact List**

[[](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)

**Application Code**  
[Download](http://www.davidtucker.net/airtips/ModelLocatorSample.zip) (418 kB)

# [Getting Started with Cairngorm – Part 2](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)

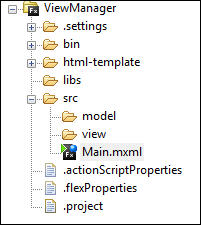
###### October 18th, 2007

**Recap:** *In* [*Part 1*](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)*, I discussed the basic implementation and use of the ModelLocator pattern. This pattern is one of many design patterns contained within the Cairngorm micro-architechture. This design pattern will be used in Part 2 as well, so it is assumed that you are familiar with the concepts in Part 1 of the tutorial. At this point, we still are not working with a "complete" Cairngorm application (that will come in Part 3).*

**Part 2 - Using a ModelLocator to Manage the View**

**Note:** As with all of the tutorials that will come in this series, this lesson has two parts. First, in the article you will learn the theory behind the topic, and then in the video you will do an actual "code-along". The article will give some instructions in how to set up your project for the "code-along".

In the previous tutorial you learned the advantages of using a ModelLocator to manage data within an application, however, the advantage of the ModelLocator pattern extends beyond the management of data. It can manage the "view" of an application as well. To see how view management works in Cairngorm, you will first need to create a new project named "ViewManager" and name the main application file "Main.mxml". For this project, your will also need to add the Cairngorm.swc to your project build path (as described in [Part 1](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)). You will also need to create two new folders inside of the "src" folder: view and model. When you are completed, your project should look similar to Figure 1 below.

  
**Figure 1** - ViewManager Project

Next, you will need to take the ModelLocator code from the previous tutorial and place it inside your application. I have posted the code below for your convenience.

[PLAIN TEXT](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)

**Actionscript:**

1. package model {
2. import com.adobe.cairngorm.model.IModelLocator;
3. [Bindable]
4. public **class** ViewModelLocator implements IModelLocator {
5. *// Single Instance of Our ModelLocator*
6. private static **var** instance:ViewModelLocator;
7. public **function** ViewModelLocator(enforcer:SingletonEnforcer) {
8. if (enforcer == **null**) {
9. throw **new** Error( "You Can Only Have One ViewModelLocator" );
10. }
11. }
12. *// Returns the Single Instance*
13. public static **function** getInstance() : ViewModelLocator {
14. if (instance == **null**) {
15. instance = **new** ViewModelLocator( **new** SingletonEnforcer );
16. }
17. return instance;
18. }
19. *//DEFINE YOUR VARIABLES HERE*
20. }
21. }
22. *// Utility Class to Deny Access to Constructor*
23. **class** SingletonEnforcer {}

**Example 1** - The ModelLocator from [Part 1](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/)

If you need information about the ModelLocator, please return to [Part 1](http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/) of the tutorial.

The only item that must be changed in the ModelLocator is the "package" statement. For this project, you will be placing the ModelLocator in the "model" folder, so the package path simply needs to me "model" (it has already been corrected above). You will also need to add one variable to your ModelLocator initially. This variable will be called "workflowState" and it will be of type "uint". The declaration will look like this:

[PLAIN TEXT](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)

**Actionscript:**

1. public **var** workflowState:uint = 0;

**Example 2** - Defining the workflowState Variable

This variable will be used to "control" the view in your application. The most common way to accomplish this is to use a ViewStack [ [Reference](http://livedocs.adobe.com/labs/flex/3/langref/mx/containers/ViewStack.html" \t "_blank) ]. If you are not familiar with a ViewStack, feel free to read through [this information](http://livedocs.adobe.com/flex/201/html/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Book_Parts&file=navigators_066_03.html). A ViewStack has a property named "selectedIndex". This numeric value defines which "child" is visible in the ViewStack. Consider the following code:

[PLAIN TEXT](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)

**mxml:**

1. **<mx**:ViewStack id="myViewStack"**>**
3. **<mx**:HBox id="box1"**>**
4. **<mx**:Label text="I am Box 1" **/>**
5. **</mx**:HBox**>**
7. **<mx**:HBox id="box2"**>**
8. **<mx**:Label text="I am Box 2" **/>**
9. **</mx**:HBox**>**
11. **<mx**:HBox id="box3"**>**
12. **<mx**:Label text="I am Box 3" **/>**
13. **</mx**:HBox**>**
15. **</mx**:ViewStack**>**

**Example 3** - A Basic ViewStack Example

Initially the value of selectedIndex is 0. With this setting "box1" would be visible. If you issue the following command:

[PLAIN TEXT](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)

**Actionscript:**

1. myViewStack.selectedIndex = 1;

**Example 4** - Manually Setting the selectedIndex

then the box named "box2" would be visible. However, if you apply the ModelLocator to this concept, you could use the workflowState varaible to set the selectedIndex property. By binding the selectedIndex to the workflowState value, you now have complete control over what is displayed in the ViewStack from your ModelLocator.

[PLAIN TEXT](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)

**mxml:**

1. **<mx**:ViewStack id="myViewStack" selectedIndex="{modelLocator.workflowState}"**>**
2. ...
3. **</mx**:ViewStack**>**

**Example 5** - Binding the selectedIndex to the workflowState

**Defining Constants for Better Code**

It would be simple to manipulate the view using this method, however, it could lead to potentially confusing code. For example, assume that you have the following:

* A ViewStack with two children: a Login Screen and a Welcome Screen
* The ViewStack's selectedIndex is bound to the workflowState property
* A login button that performs the action shown in Example 4

This might seem as if it works properly, but it doesn't account for any changes. If another child is added to the ViewStack, it could throw off the order. There needs to be a better way to manually set the selectedIndex property. To accomplish this you just need to define constants inside of the ModelLocator.

[PLAIN TEXT](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)

**Actionscript:**

1. *//DEFINE YOUR VARIABLES HERE*
2. public **var** workflowState:uint = 0;
4. *// DEFINE VIEW CONSTANTS*
5. public static const LOGIN\_SCREEN = 0;
6. public static const WELCOME\_SCREEN = 1;

**Example 6** - Defining View Constants in the ModelLocator

By using this method, you only have to change the value in one location if the number of children or the order of the children changes in the ViewStack. Now, you would assign the login button the following action to the click event:

[PLAIN TEXT](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)

**Actionscript:**

1. myViewStack.selectedIndex = ViewModelLocator.WELCOME\_SCREEN;

**Example 7** - Setting the View with Defined Constants

Not only do you protect against future changes, you also have made your code much more logical. Another developer could easily look at the code and understand the process without having to reference all of the children in the ViewStack.

**NOTE:** The audio was not that great for this video. I will be using a better system for the next tutorial.

**Video Example** - Controlling the View with a ModelLocator

[[](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)](http://www.davidtucker.net/2007/10/18/cairngorm-part-2/)

**Application Code**  
[Download](http://www.davidtucker.net/airtips/ViewManager.zip) (3 kB)

# [Getting Started with Cairngorm – Part 3](http://www.davidtucker.net/2007/10/29/cairngorm-part-3/)

###### October 29th, 2007

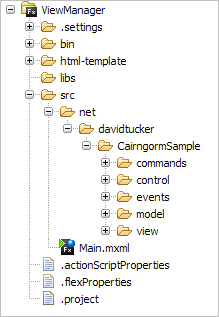
Now that you have isolated two specific elements of the Cairngorm Micro-Architecture, you will now create a more complete Cairngorm application. Up to now the tutorials have covered only one design pattern, the ModelLocator, but now you will be introduced to the most crucial element of Cairngorm, the Service to Worker design pattern. The explanation of this pattern will span two tutorials. This tutorial will cover the basic flow inside of a Cairngorm application, and the next tutorial will expand this flow to include server interaction. However, before you can properly implement this design pattern you need to learn about the organization of a Cairngorm project.

**Organizing a Cairngorm Project**

One of the tasks involved with any project is organization. When working with other developers, this becomes extremely important. Normally a Cairngorm project is organized in the following manner:

* There is a Main.mxml file that is the main application file for the Cairngorm application.
* The project files are contained in a folder that uses "reverse-dns" format. For example, if the project was named CairngormSample, I would use the following folders net/davidtucker/CairngormSample .
* Within the CairngormSample directory, there will be the following folders (there will be additional folders added in the next tutorial).
  + **events/** - This directory holds all of the custom events for the application
  + **control/** - This directory houses the FrontController for the application
  + **commands/** - This directory contains the Commands that are called by the FrontController
  + **model/** - The ModelLocator is contained in this folder (and other classes related to the model)
  + **view/** - The components of the view are contained in this directory

By following this standard, you can know where to find any class that you may need in your Cairngorm application. Figure 1 illustrates this project structure. It also is a good development process to have a standard organizational structure for your projects - even if you are not using Cairngorm.



**Figure 1** - Cairngorm Project Structure

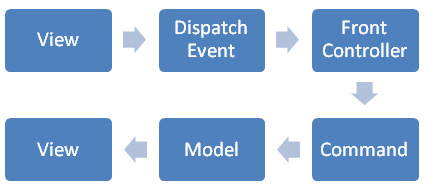
**The Service to Worker Pattern**

The Service to Worker pattern is the hardest for most people to grasp. It encompasses most of the logic for a Cairngorm application. To understand this pattern, you will need to understand some of the classes that are included with Cairngorm and their respective purposes.

* **CairngormEvent** [[Reference](http://www.davidtucker.net/docs/cairngorm/com/adobe/cairngorm/control/CairngormEvent.html)] - CairngormEvent is central in this pattern. It extends from Event. For your event to be handled properly in Cairngorm, it will need to be of type CairngormEvent.
* **CairngormEventDispatcher** [[Reference](http://www.davidtucker.net/docs/cairngorm/com/adobe/cairngorm/control/CairngormEventDispatcher.html)] - CairngormEventDispatcher actually dispatches the CairngormEvents. It is a singleton (just like the ModelLocator). In previous Cairngorm versions, you would call this class regularly, but now CairngormEvents can dispatch themselves (via the method dispatch). This method is simply a wrapper for the CairngormEventDispatcher, so even if you don't actually import and call the class, it is still central to the Service to Worker pattern in Cairngorm.
* **FrontController** [[Reference](http://www.davidtucker.net/docs/cairngorm/com/adobe/cairngorm/control/FrontController.html)] - The FrontController maps a CairngormEvent to a Command.
* **ICommand** [[Reference](http://www.davidtucker.net/docs/cairngorm/com/adobe/cairngorm/commands/ICommand.html)] - For a Command to function properly in Cairngorm, it needs to implement the ICommand interface. This interface requires that a command have a method named execute and that it accept the CairngormEvent that is mapped to it as an argument.

**The Cairngorm Event Flow**

The way that these classes interact is the Cairngorm Event Flow. Figure 2 illustrates this entire process. While this process seems lengthy, it follows a logical order.



**Figure 2** - Cairngorm Basic Event Flow

For example, assume that Figure 2 shows what happens when a user clicks a login button. It would follow the following steps:

1. The user is viewing a component that has a login button.
2. When the button is clicked, that component would dispatch an event named LoginEvent that extends CairngormEvent.
3. The FrontController would receive the LoginEvent and execute the command that is mapped to that event. For this example, it will be named LoginCommand and will implement ICommand.
4. The Command is executed and received the LoginEvent as an argument. The LoginCommand will change the workflowState value in the ModelLocator.
5. The ModelLocator will have a predefined constant for the "Logged in View". The command will change workflowState to this value.
6. Since the view is bound to workflowState, it will automatically update itself to the new view.

Once these items are understood, the next most important thing to understand about Cairngorm is: **Everything Should Be Mapped to an Event**. This is a drastic over-simplification, but it holds true in most situations. When a user clicks a login button - it should dispatch a CairngormEvent. When the user selects an option to change the view - it should dispatch a CairngormEvent. When a user submits a form to be stored on in a database - the form should dispatch a CairngormEvent. The dispatching of a CairngormEvent sets everything in motion.

**Custom CairngormEvents**

The class CairngormEvent can be used inside of your project, but in most situations you will create your own custom events that extend CairngormEvent (as stated previously, for an event to be included in the Cairngorm Event Flow it should extend CairngormEvent). Another reason to create custom events it to create custom properties of the event which contain data that you need to pass to the command. CairngormEvent has a property named data (of type Object) that can contain data, but it is ideal to have a strongly-typed property where you can place the data to pass.

For this example you will create an event that corresponds to the earlier example of a login page. This event will need to meet the following criteria:

* Extend CairngormEvent
* Have a property for the username
* Have a property for the password

Code Example 1 illustrates this completed event. You can see that you will need to import both CairngormEvent and the basic Event class. Also, as with all events, you have to define the Event constant. In this instance, you can use LOGIN. The properties are defined below the constant - and they also are passed into the constructor. The only thing out of the ordinary is that you need to override the public method clone(). This is the the method that the event uses to make a copy of itself. This is key in the Cairngorm Event Flow. Also, for the function to implement ICommand this method will need to have a return type of Event (and not CairngormEvent).

[PLAIN TEXT](http://www.davidtucker.net/2007/10/29/cairngorm-part-3/)

**Actionscript:**

1. package net.davidtucker.CairngormSample.events {
3. import com.adobe.cairngorm.control.CairngormEvent;
4. import flash.events.Event;
6. public **class** LoginEvent extends CairngormEvent {
8. public static const LOGIN:String = "Login";
10. public **var** username:String;
11. public **var** password:String;
13. public **function** LoginEvent(submittedUsername:String,submittedPassword:String) {
14. username = submittedUsername;
15. password = submittedPassword;
16. super(LOGIN);
17. }
19. override public **function** clone():Event {
20. return **new** LoginEvent(username,password);
21. }
23. }
24. }

**Code Example 1** - Custom Event

**Commands**

The Commands within a Cairngorm application actually "command" the application. Even in the next tutorial where you will be interacting with a server, the command still will do a majority of the work. In this example you will create a component that will receive the username and password from the LoginEvent, check the values against hard-coded values, and then change the workflowState on the ModelLocator if the values are correct. The following example performs these steps (but it doesn't have the hard-coded values included - that will be covered in the video).

The first thing to notice about the code below in Code Example 2 is that the LoginCommand implements ICommand. To accomplish this the ICommand class is also imported. In addition, you will notice the boilerplate code that you have used in the past to bring in the ModelLocator. To properly implement ICommand, the method execute() is also created. It received an event of type CairngormEvent (it has to be CairngormEvent and not LoginEvent to properly implement ICommand). To properly use the event, you will need to cast this event to the type LoginEvent (the process of casting will be further explained in the video). The actual logic has been left out of this command, but you can see that the ModelLocator updates the view. Once the logic is implemented this line of code would probably be inside of an if statement.

[PLAIN TEXT](http://www.davidtucker.net/2007/10/29/cairngorm-part-3/)

**Actionscript:**

1. package net.davidtucker.CairngormSample.commands {
3. import com.adobe.cairngorm.commands.ICommand;
4. import com.adobe.cairngorm.control.CairngormEvent;
5. import net.davidtucker.CairngormSample.events.LoginEvent;
6. import net.davidtucker.CairngormSample.model.ViewModelLocator;
8. public **class** LoginCommand implements ICommand {
10. public **var** modelLocator:ViewModelLocator = ViewModelLocator.getInstance();
12. public **function** LoginCommand() {
13. }
15. public **function** execute(event:CairngormEvent):void {
17. **var** loginEvent:LoginEvent = event as LoginEvent;
19. *// COMMAND LOGIC*
21. modelLocator.workflowState = ViewModelLocator.WELCOME\_SCREEN;
23. }
24. }
25. }

**Code Example 2** - Cairngorm Command

**Front Controller**

As stated earlier, the FrontController maps your CairngormEvents to specific commands. Without this, your events would never integrate into the Cairngorm flow. The class extends FrontController and has two methods: the constructor and the initialize method. The initialize method is where you will use the addCommand method to map your events to commands (as you can see with the LoginEvent and LoginCommand).

[PLAIN TEXT](http://www.davidtucker.net/2007/10/29/cairngorm-part-3/)

**Actionscript:**

1. package net.davidtucker.CairngormSample.control {
3. import com.adobe.cairngorm.control.FrontController;
5. import net.davidtucker.CairngormSample.commands.\*;
6. import net.davidtucker.CairngormSample.events.\*;
8. public **class** SampleController extends FrontController {
10. public **function** SampleController() {
11. this.initialize();
12. }
14. public **function** initialize():void {
16. *//ADD COMMANDS*
17. this.addCommand(LoginEvent.LOGIN,LoginCommand);
19. }
21. }
22. }

**Code Example 3** - The Cairngorm FrontController

Simply creating a FrontController is not enough. Like any class it needs to be instantiated inside of your application. Code Example 4 illustrates how to instantiate your FrontController in the Main.mxml file of your application. You simply need to add the control directory as an XML Namespace and then include the FrontController tag in the file.

[PLAIN TEXT](http://www.davidtucker.net/2007/10/29/cairngorm-part-3/)

**mxml:**

1. **<**?xml version="1.0" encoding="utf-8"?**>**
2. **<mx**:Application xmlns:mx="http://www.adobe.com/2006/mxml"
3. xmlns:control="net.davidtucker.CairngormSample.control.\*"
4. layout="absolute"**>**
6. **<control**:FrontController id="controller" **/>**
8. **</mx**:Application**>**

**Code Example 4** - Integrating a FrontController

Now that you have seen the basic elements of a Cairngorm project, you can actually build the working sample with the video. As always, the application code is included below.

[[](http://www.davidtucker.net/2007/10/29/cairngorm-part-3/)](http://www.davidtucker.net/2007/10/29/cairngorm-part-3/)

**Application Code**  
[Download](http://www.davidtucker.net/airtips/Cairngorm3.zip) (7 Kb)

# [Getting Started with Cairngorm – Part 4](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)

###### November 7th, 2007

The basic Cairngorm Event Flow that is handled in [Part 3](http://www.davidtucker.net/2007/10/29/cairngorm-part-3/) is essential to any Cairngorm application, but most applications interact with a server. The Service to Worker pattern that was discussed in the previous tutorial is essential to this process. To learn the expanded Cairngorm Flow, you will need to learn a few new Cairngorm elements.

**The Server Interaction Elements**

There are three new types of classes that you will need to be familiar with to understand the full Service to Worker Pattern.

* **ServiceLocator** [[Reference](http://www.davidtucker.net/docs/cairngorm/com/adobe/cairngorm/business/ServiceLocator.html)] - The ServiceLocator is a singleton that contains references to all of the services that your application will use. It usually includes HTTPService, WebService, or RemoteObject calls.
* **Business Delegate** - A Business Delegate has three functions: to locate the service that is needed, to call a method on the service, and to route the result to the specified responder. There is usually one Business Delegate for each service.
* **Value Object** - Value Objects are classes that by definition do not require any methods, only properties. They allow for the application to transfer strongly-typed complex data objects to and from the server.

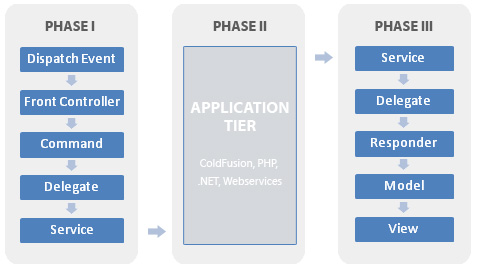
**Organizing Your Cairngorm Project** (With Server Interaction)

In the last tutorial you saw the standard Cairngorm structure for a project. To complete this structure, you will need to add an additional two folders:

* **/business** - This folder typically contains two types of files: the ServiceLocator (named Services.mxml) and Business Delegates.
* **/vo** - This folder contains the value objects that are passed to and from the server.

**Understanding the Process**

Before you begin crafting the server interaction for your project, you need to understand the full Service to Worker pattern. Figure 1 illustrates the entire process.



**Figure 1** - Cairngorm Event Flow with Server Interaction

**Phase I** - Execution Phase

1. The user is viewing a component that has a login button. When the button is clicked, that component would dispatch an event named LoginEvent that extends CairngormEvent.
2. The FrontController would receive the LoginEvent and execute the command that is mapped to that event. For this example, it will be named LoginCommand and will implement ICommand.
3. The Command is executed and received the LoginEvent as an argument. The LoginCommand creates an instance of the Delegate while passing a reference to the Responder. The Command then calls the specified method on the Delegate.
4. The Delegate gets the service from the ServiceLocator and calls the method on the specific service.
5. The service defined in the ServiceLocator receives the information that is passed to it from the call in the Delegate.

**Phase II** - Application Tier Processing

**Phase III** - Response Phase

1. The service that is defined in the ServiceLocator returns the result of the server-side processing.
2. The Delegate receives the result and passes it to the specified responder.
3. The Responder receives the result indicating a successful login.
4. The ModelLocator will have a predefined constant for the "Logged in View". The Responder will change workflowState to this value.
5. Since the view is bound to workflowState, it will automatically update itself to the new view.

**Note:** Traditionally the Command and Responder were the same class in a Cairngorm application. If you are working with other developers, the applications will probably be coded in this manner. However, most developers (including Adobe Consulting) are now separating these classes into two different classes. For a less complex project, it might not make sense to separate these items, but in a large application it could prove advantageous in organization and practice to have two different classes.

**The ServiceLocator**

The ServiceLocator is a singleton that contains references to all of the services that the application will be using. These services can be RemoteObjects, HTTPServices, WebServices, or custom services. Like the FrontController, this class is usually instantiated in your main application file. Unlike many of the other Cairngorm project assets that you have created, this class can be defined in MXML (as well as ActionScript). To properly define this in MXML, there must be a namespace that points to the business folder of the Cairngorm package. In this case the namespace cairngorm is typically used. The ServiceLocator in Code Example 1 has one service defined, loginService.

[PLAIN TEXT](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)

**Actionscript:**

1. <?xml version="1.0" encoding="utf-8"?>
2. <cairngorm:ServiceLocator
3. xmlns:cairngorm="com.adobe.cairngorm.business.\*"
4. xmlns:mx="http://www.adobe.com/2006/mxml">
6. <!-- Login Service -->
7. <mx:RemoteObject id="loginService"
8. showBusyCursor="true"
9. destination="ColdFusion"
10. source="CairngormTest.CairngormLogin">
12. <mx:method name="login" />
14. </mx:RemoteObject>
16. </cairngorm:ServiceLocator>

**Code Example 1** - Sample ServiceLocator

The ServiceLocator is typically named Services.mxml and resides in the business folder of your Cairngorm project.

**Commands with Server Interaction**

In the last tutorial you hard-coded some values into the LoginCommand to check for a specific username and password. Here was the execute method:

[PLAIN TEXT](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)

**Actionscript:**

1. public **function** execute(event:CairngormEvent):void {
2. **var** loginEvent:LoginEvent = event as LoginEvent;
3. if( (loginEvent.username == "david") && (loginEvent.password == "password")) {
4. modelLocator.workflowState = ViewModelLocator.WELCOME\_SCREEN;
5. }
6. }

**Code Example 2** - Command Without Delegate

This methodology will hold true for any Command that doesn't have server interaction, but if you do have server interaction, it will need to be modified to include the delegate. First, you will need to make a decision. As stated earlier, you can have separate Commands and Responders, or they can be the same class. For this example, they will be the same class. The Command will now also need to implement the mx.rpc.IResponder class (please note that com.adobe.cairngorm.business.Responder is deprecated and should no longer be used).

The other initial change is that the execute method now instantiates a class named LoginDelegate (which will be created shortly). The LoginDelegate class required one argument in its constructor, the Responder of the service. In this case, the Command will function both as the command and the responder, so you simply need to insert the this keyword inside the parenthesis. If you chose to have a separate responder, you would insert the reference to it here (instead of the this keyword).

[PLAIN TEXT](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)

**Actionscript:**

1. package net.davidtucker.CairngormSample.commands {
3. import com.adobe.cairngorm.commands.ICommand;
4. import com.adobe.cairngorm.control.CairngormEvent;
5. import mx.controls.Alert;
6. import mx.rpc.IResponder;
7. import net.davidtucker.CairngormSample.business.LoginDelegate;
8. import net.davidtucker.CairngormSample.events.LoginEvent;
9. import net.davidtucker.CairngormSample.model.ViewModelLocator;
11. public **class** LoginCommand implements ICommand,IResponder {
13. public **var** modelLocator:ViewModelLocator = ViewModelLocator.getInstance();
15. public **function** LoginCommand() {
16. }
18. public **function** execute(event:CairngormEvent):void {
19. **var** loginEvent:LoginEvent = event as LoginEvent;
20. **var** delegate:LoginDelegate = **new** LoginDelegate( this );
21. delegate.login(loginEvent.loginAttempt);
22. }
24. public **function** result( event:Object ):void {
25. if(event.result == **true**) {
26. modelLocator.workflowState = ViewModelLocator.WELCOME\_SCREEN;
27. } else {
28. mx.controls.Alert.show("Password Incorrect","ERROR");
29. }
30. }
32. public **function** fault( event:Object ):void {
33. trace("Service Error");
34. }
36. }
37. }

**Code Example 3** - Command with Server Interaction Through a Delegate

**Value Objects**

A Value Object does not extend or implement any Cairngorm class. As stated earlier, it simply is a class that is only required to have properties, but not methods. For example, if you created a Value Object for a Login - it would have a property for username and a property for password.

[PLAIN TEXT](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)

**Actionscript:**

1. package net.davidtucker.CairngormSample.vo {
3. [RemoteClass(alias="CairngormTest.LoginVO")]
4. public **class** LoginVO {
6. public **var** username:String;
7. public **var** password:String;
9. public **function** LoginVO(username:String,password:String) {
10. this.username = username;
11. this.password = password;
12. }
14. }
15. }

**Code Example 4** - Value Object for Login

The RemoteClass metatag is important to note. This will allow the corresponding server-side object (ColdFusion Component, PHP Class, Java Class, etc...) to be mapped to this Value Object. In this case, it is mapped to a ColdFusion component named LoginVO in the CairngormTest folder.

**Business Delegates**

The final design pattern in the Cairngorm Micro-Architecture is the Business Delegate. A Business Delegate essentially is the abstraction layer between your services and the rest of your application. As stated earlier, it has three functions. First, the Business Delegate will locate the service that is needed in the ServiceLocator. Second, it will call a method on that service. Finally, it will route the response back to the specified responder (usually either a command or separate responder).

A Delegate class doesn't extend or implement and Cairngorm classes, but it generally follows the following guidelines.

* The Delegate has at least two properties: one named service which is a reference to a service in the ServiceLocator and one named responder which is the responder for the service calls.
* There is a method for each server-side method that you will be calling.
* Both the responder and the service variables are set in the constructor.

[PLAIN TEXT](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)

**Actionscript:**

1. package net.davidtucker.CairngormSample.business {
3. import mx.rpc.IResponder;
4. import net.davidtucker.CairngormSample.vo.LoginVO;
5. import com.adobe.cairngorm.business.ServiceLocator;
7. public **class** LoginDelegate {
8. private **var** responder : IResponder;
9. private **var** service : Object;
11. public **function** LoginDelegate( responder:IResponder ) {
12. this.responder = responder;
13. this.service = ServiceLocator.getInstance().getRemoteObject("loginService");
14. }
16. public **function** login(login:LoginVO):void {
17. **var** call:Object = service.login( login );
18. call.addResponder( responder );
19. }
21. }
22. }

**Code Example 5** - Service Delegate

There are many benefits to having this layer. If coded correctly, you should be able to change out the server interaction (going from PHP to ColdFusion for example) and only have to change the code in your ServiceLocator and your Delegate. You also can easily insert "stub code" to simulate the actual server interaction during the early stages of development.

**The Application Tier**

In this example, the Application Tier will be handled by a ColdFusion 8 installation. It will contain two Coldfusion components. These components are purposefully simple.

1. **LoginVO.cfc** - This component will correspond to the ActionScript class LoginVO that you created earlier.
2. **CairngormLogin.cfc** - This component will perform the actual login processing.

In this example, the Flex application will pass a LoginVO ActionScript object to the CairngormLogin.cfc's login method through a RemoteObject call. This will be mapped to a LoginVO.cfc object. If this LoginVO.cfc object has the username "david" and the password "password" the method will return true. If not, it will return false.

[PLAIN TEXT](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)

**Coldfusion:**

1. **<cfcomponent** displayname="LoginVO" hint="Login VO For CairngormTest" output="false"**>**
3. **<cfset** this.username = "" **/>**
4. **<cfset** this.password = "" **/>**
6. **</cfcomponent>**

**Code Example 6** - LoginVO.cfc

[PLAIN TEXT](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)

**Coldfusion:**

1. **<cfcomponent** displayname="CairngormLogin" hint="CFC to Test Cairngorm Service Interaction" output="false"**>**
3. **<cffunction** name="login" displayname="login" access="remote" output="false" returntype="boolean"**>**
4. **<cfargument** name="loginAttempt" type="LoginVO" required="true" **/>**
6. **<cfif** (loginAttempt.username EQ "david") AND (loginAttempt.password EQ "password")**>**
7. **<cfreturn** true **/>**
8. **<cfelse>**
9. **<cfreturn** false **/>**
10. **</cfif>**
12. **</cffunction>**
14. **</cfcomponent>**

**Code Example 7** - CairngormLogin.cfc

[[](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/)

**Application Code**  
[Download](http://www.davidtucker.net/airtips/Cairngorm4.zip) (11 Kb)

**Looking Ahead**

There are only two tutorials remaining in the Cairngorm series. In the next tutorials you will learn about code generation options for Cairngorm, and you will also build an actual application using Cairngorm.

**Reference**

The following resouces should assist you in getting Flex connected to your application server.

* [ColdFusion](http://flex.org/coldfusion/)
* [Java](http://flex.org/java/)
* [.NET](http://flex.org/dotnet/)
* [ColdFusion](http://flex.org/coldfusion/)
* [PHP](http://flex.org/php/)
* [Ruby](http://flex.org/ruby/)

# [Getting Started with Cairngorm – Part 5](http://www.davidtucker.net/2007/11/30/getting-started-with-cairngorm-%e2%80%93-part-5/)

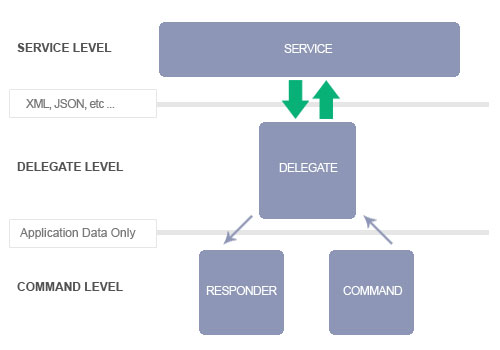
###### November 30th, 2007

In [Part 4](http://www.davidtucker.net/2007/11/07/cairngorm-part-4/) you saw the full Service to Worker pattern demonstrated. However, the method discussed in the last tutorial doesn't fit every situation. In this tutorial you will learn a few "best-practices" for Cairngorm projects as well as an extension to the Service to Worker pattern for more complex cases.

**Delegates, Commands, and Responders**

These classes (Delegates, Commands, and Responders) have specific responsibilities. Many developers run into problems when they do not adhere to these responsibilities. The following tips are guidelines for your code.

* **Responders** - Responders should deal only in strongly-typed application objects. Responders may set values on the model.
* **Commands** - Commands never deal directly with server interaction. Commands may set values on the model.
* **Delegates** - Delegates should pass strongly typed strongly-typed application objects to the Responder. If the Delegate doesn't receive strongly-typed application objects, it must parse and convert the data into that format.



**Figure 1** - Data Transmission in the Service to Worker Pattern

The goal of this separation of responsibility is that services should be able to be switched out with only a change in the delegate. This may sound a bit unimportant, but if you are working for a large company, the server-side implementation can change drastically from development to production. For this to work properly, the service call will need to have two responders: the Delegate, and the actual Responder. Figure 1 illustrates this separation. The data passed from the service to the delegate can be in virtually any format, but the transfer from the Delegate to the Responder needs to only be application data (VO's, model classes).

In this case, the Delegate now has an added responsibility, parsing the raw data. To accomplish this, you can create a factory class to parse the data. In the sample below the class DeliciousFactory (because it parses data from a [Del.icio.us](http://del.icio.us/) feed) has a method buildPostArrayFromJSON which handles this responsibility.

[PLAIN TEXT](http://www.davidtucker.net/2007/11/30/getting-started-with-cairngorm-%e2%80%93-part-5/)

**Actionscript:**

1. package net.davidtucker.deliciousbrowser.business {
3. import com.adobe.cairngorm.business.ServiceLocator;
5. import mx.rpc.AsyncToken;
6. import mx.rpc.IResponder;
7. import mx.rpc.events.FaultEvent;
8. import mx.rpc.events.ResultEvent;
9. import net.davidtucker.deliciousbrowser.factories.DeliciousFactory;
11. public **class** PostsDelegate {
13. private **var** responder:IResponder;
14. private **var** service:Object;
16. public **function** PostsDelegate( responder:IResponder ) {
17. this.responder = responder;
18. this.service = ServiceLocator.getInstance().getHTTPService( "sampleService" );
19. }
21. public **function** getRecentPosts():void {
23. *// Call the Service and Attach that call to an AsyncToken*
24. **var** token:AsyncToken = service.send(params);
26. *// Create a Responder for the call*
27. **var** responder:mx.rpc.Responder = **new** mx.rpc.Responder(onResult, onFault);
29. *// Add the Responder to the AsyncToken*
30. token.addResponder(responder);
32. }
34. private **function** onResult( e:ResultEvent ):void {
36. *// Parse JSON into Array of VO's*
37. **var** posts:Array = DeliciousFactory.buildPostArrayFromJSON( e.result as String );
39. *// Pass VO Array to Responder*
40. responder.result( **new** ResultEvent( ResultEvent.RESULT, **false**, **true**, posts ) );
42. }
44. private **function** onFault( e:FaultEvent ):void {
45. ...
46. }
48. }
49. }

**Code Example 1** - A Delegate Parsing Raw Server Data

Parsing the server data is only a part of this extended design pattern. For this to work properly with the traditional Cairngorm flow, the delegate must "intercept" the result of the service call. To accomplish this, you will attach an **AsyncToken** [[Reference](http://livedocs.adobe.com/flex/2/langref/mx/rpc/AsyncToken.html)] to the service call, create a responder for the call, and then attach the responder to the AsyncToken. This is illustrated in lines 23-30 of Code Example 1.

Now that the delegate has "intercepted" the result of the service call, it need to have methods to parse the data. When you created the responder inside of the delegate (line 27 of Code Example 1), you passed in a result and fault method. In this case it was onResult and onFault. You now need to add these methods to the delegate.

To pass the parsed data to the original responder, you will need to create a new instance of the **ResultEvent** [[Reference](http://livedocs.adobe.com/flex/2/langref/mx/rpc/events/ResultEvent.html)] with the parsed data in the result property (line 40 of Code Example 1). This event will be passed as an argument to the original responder's result method.

**Setting ModelLocator Values**

Within a Cairngorm application, it can be extremely tempting to set values of ModelLocator variables in your events, delegates, or views, but this should be avoided. These values should be set only in the Command Level (see Figure 1) of the application. This means that you can set these values in a Command or Responder. This keeps every class in tune with its true function.

**Sequence Commands**

Cairngorm also contains a class, **SequenceCommand** [[Reference](http://www.davidtucker.net/docs/cairngorm/com/adobe/cairngorm/commands/SequenceCommand.html)] that allows the developer to chain command/events together for sequential execution. Initially this may seems like it breaks the framework's rule oft having every command triggered by an event, but it doesn't. Inside of a SequenceCommand you define the nextEvent property (which is an instance of a CairngormEvent). This event is executed when the executeNextCommand method is called. This is the most effective way to attach multiple commands to a single user gesture.

[PLAIN TEXT](http://www.davidtucker.net/2007/11/30/getting-started-with-cairngorm-%e2%80%93-part-5/)

**Actionscript:**

1. public **class** SampleSequenceCommand extends SequenceCommand implements ICommand, IResponder {
3. public **function** SampleSequenceCommand() {
4. this.nextEvent = **new** GetPostsEvent();
5. }
7. override public **function** execute( e:CairngormEvent ):void {
8. ...
9. }
11. public **function** result( event:Object ):void {
12. this.executeNextCommand();
13. }
15. public **function** fault( event:Object):void {
16. ...
17. }
19. }

**NOTE:** The SequenceCommand class assumes that you are not separating your Command and Responder. You could easily extend the architecture and create a SequenceResponder class to achieve the same effect.

**ViewHelper and ViewLocator Classes**

Many people have inquired about the **ViewHelper** [[Reference](http://www.davidtucker.net/docs/cairngorm/com/adobe/cairngorm/view/ViewHelper.html)] and **ViewLocator** [[Reference](http://www.davidtucker.net/docs/cairngorm/com/adobe/cairngorm/view/ViewLocator.html)] classes. Both of these classes are now deprecated as of Cairngorm 2.1, and it is considered bad practice to use either of these classes in a new Cairngorm project. Should you be working with an existing project that still uses these classes, I would recommend refactoring your code to meet Cairngorm 2.2 standards.

**Summary of Tips**

1. Don't use ViewHelpers and ViewLocators
2. Only set ModelLocator variables in the Command Level (Command or Responder).
3. SequenceCommand can be used to chain multiple commands to a single user gesture.
4. Delegates deal in raw service data (which could be application data), but Responders only deal in strongly-typed application data.
5. Commands do not directly communicate with services.

[[](http://www.davidtucker.net/2007/11/30/getting-started-with-cairngorm-%e2%80%93-part-5/)](http://www.davidtucker.net/2007/11/30/getting-started-with-cairngorm-%e2%80%93-part-5/)

**Application Code**  
[Beginning Code Download](http://www.davidtucker.net/airtips/Cairngorm5-1.zip) (9 Kb)  
[Finished Code Download](http://www.davidtucker.net/airtips/Cairngorm5-2.zip) (12 Kb)

**References**  
[AS3CoreLib](http://code.google.com/p/as3corelib/downloads/list)

**Looking Ahead**

In the next (and final) tutorial on Cairngorm, you will learn tips to make your Cairngorm development go faster as well as learning more "best practices" for Cairngorm development.