Flex Application Architecture

by Jonnie Spratley

The role of the View in a Cairngorm application is to throw events based on user actions (such as button clicks, loading, entering of data etc.). And bind to the Model for data representation. (Either through Value Objects or other structures).

What you will learn...

- Cairngorm Framework
- M-V-C Application Architecture
- Maintainable and Expandable Application

What you should know...

- Basic Action Script
- Basic MXML

Level of difficulty

Every Flex programmer, new and old, beginner or advanced will eventually need some type of framework when building applications of the real world. This is where Adobe's Cairngorm M-V-C style architecture comes into play.

The Cairngorm is a micro-architecture framework helping the developer separate the model, the view, and the control of the application in to small reusable classes and components that will make your application more efficient, more reliable, and extendible than ever before.

When building Flex applications for the first time, you are going to put all of your applications code, data, and control inside one component, you might think that is a good idea right now at your early stages of programming but it isn't. If you ever wanted to build a real world application you are going to have to get some type of structure, a set of guidelines, and a process in which your application flows.

Frameworks play a major role when building applications that people are actually going to use, so when choosing a framework to develop with be sure to do a lot of research before just using it.

The current framework of choice among Flex developers is Adobes Cairngorm Framework; with its M-V-C type architecture it becomes quickly the favored framework to use. Some say that learning the Cairngorm is a huge task, it is known to have one of the hardest learning curves of the frameworks, but that is just because developers are sometimes lazy and want do not want to take the time to study something new, being outside of ones comfort zone is never a good feeling.

I will explain to you that building Flex applications under the Cairngorm Framework is a piece of cake, you just have to forget all of what you have heard about it, and start fresh. In the end your application will become maintainable, re-usable, cleaner, expandable, and more reliable than ever before.

Getting ready to use the Cairngorm

To begin building Flex/Air applications using the Cairngorm framework, you have to download the free Cairngorm library from opensource.adobe.com.

After you have downloaded the file, you create a new Flex project or use an existing one, just copy and paste the Cairngorm.swc file to the libs folder of the project. After you project as re-build, you are now ready to start using the excellent features that Cairngorm has to offer.

Now it is time to start structuring your application, inside of your src folder of the application you some new folders to put your classes and component's. Using reverse domain syntax you should create a folder structure as shown below.

Cairngorm Application Folder Structure, see (Figure 1).

Learning about the components of Cairngorm

Lets take a minute to understand the components of Cairngorm. The Cairngorm is made up of numerous classes/packages that help the developer separate all logic of the application into maintainable classes of code.

But when creating these classes it is good to understand what each class is suppose to do, and how it works. Cairngorm involves a ModelLocator, a ServiceLocator, a FrontController, a Delegate, a Command, an Event; the view is made up of your components.

The Model Locator holds the data that is being used through out the application, it acts as a data storage bank for your application, any data is required to live inside of your application, should be placed inside of your applications ModelLocator.

The Service Locator holds the locations of all server side services that your application will

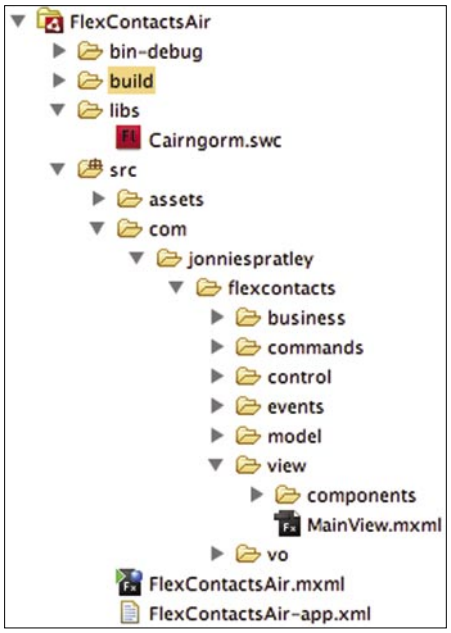

Figure 1. *Cairngorm Application Folder Structure*

be accessing such as HTTP, Web Services, Flash Remoting, etc. Placing all of your services in one central area will be very manageable and extendable for the future.

The Service Delegate is responsible for calling the actual server side methods on the server. It accesses those services by using the ServiceLocator to locate the server, and itself to actually invoke those methods.

The Front Controller is responsible for capturing Cairngorm Events that are dispatched throughout the application, such as clicking of a button, initializing of a component, response from user interaction, etc. The FrontController then maps the captured event to the proper Command.

The Command is responsible processing the event by running the Command class execute() method, this method is an ICommand interface method, which the Command implements. The Command is also responsible for updating the ModelLocator, as well as invoking a server side service with the help of a Delegate.

The Value Object represents an entity of something, the Value Object is nothing more than a representation of an item, or data structure, it just holds the attributes of data in which you are representing.

The View, the view of your application in which will be presented to the user for interaction, the view is responsible for dispatching user or system-generated events. It also binds to the ModelLocator for data representation.

Overview

- ModelLocator holds data
- Service Locator holds service locations
- Delegate holds service methods
- FrontController captures events and maps to proper Commands
- Commands carry out the event, by updating the ModelLocator and/or calling services defined inside the Delegate
- Views dispatch application wide events
- Views watch data stored inside the ModelLocator
  - Views are visual components (buttons, panels, data grids, etc.)
  - Views can contain child views and components
  - Even `<mx:Application/>` and `<mx:WindowedApplication/>` is a view

Benefits of using Cairngorm

- Allows designers, developers and data-service developers to work together
- Best used for medium to large sized applications
- Great in team environments
- Provides also easier maintenance, easier debugging, feature changes, and enhancements

Listing 1. *ContactVO.as*

```
package com.jonniespratley.flexcontacts.vo
{
	import com.adobe.cairngorm.vo.IValueObject;

	[RemoteClass(alias="com.jonniespratley.flexcontacts.vo.ContactVO")]

	[Bindable]
	public class ContactVO implements IValueObject
	{
		public var contact_id:int;
		public var contact_fname:String;
		public var contact_lname:String;
		public var contact_email:String;
		public var contact_url:String;
		public var contact_address:String;
		public var contact_address2:String;
		public var contact_state:String;
		public var contact_city:String;
		public var contact_zip:String;

		public function ContactVO( obj:Object = null )
		{
		if ( obj != null )
			{
				this.id = obj["id"];
				this.name = obj["name"];
				this.address = obj["address"];
				this.city = obj["city"];
				this.state = obj["state"];
				this.country = obj["country"];
				this.email = obj["email"];
				this.phone = obj["phone"];
				this.zip = obj["zip"];
				this.date = obj["date"];
			}
		}
	}
}
```

Cairngorm in Depth

Value Objects

When developing Flex applications it is best to create classes that represent the objects being represented or accessed through out your application, if you were displaying pictures from a art gallery, you would create a class named PhotoVO.as with public variables that represent the attributes of that photo such as height, width, filename, etc.

Value Objects are used to create a layer of data objects that can be transferred between components and server side tiers, instead of generic objects, arrays, etc.

When you use Value Objects to represent data you are allowing a stricter typing on the client, this is really great when working in Flex Builder because any references to properties inside of that value object that do not exist, or types that do not correspond with one another, the compiler will throw a error, letting the developer know exactly where the problem lies. You are setting a standard for the specific entity you are representing.

- Identify `ValueObject`:
  - Entity of something
  - Strong typed objects
  - Attributes represent that entity
  - Same layer as the `ModelLocator`
- Create `ContactVO.as`, see (Listing 1).
- Discuss – Value Object

The ContactVO.as value object holds all of the attributes for a contact inside of this application. This exact data is expected to come back from the server side service that this object is representing.

Model Locator

The ModelLocator acts as a central data bank for all of your applications data; any data that is required to live inside of the application should be placed inside of the ModelLocator. The Singleton class is used to restrict instantiation of a class to one object, and one object alone. There will only be one ModelLocator per application. For instance if you were to deposit money at an ATM machine, that money is going to be accessible everywhere, not just at that particular ATM machine. That is because you are storing your money inside of your bank.

Think of a ModelLocator as a bank, you only need one bank to store all of your data (money),

Listing 2. *ModelLocator.as*

```
package com.jonniespratley.flexcontacts.model
{
      import com.adobe.cairngorm.model.IModelLocator;
      import com.jonniespratley.flexcontacts.vo.ContactVO;

      import mx.collections.ArrayCollection;

    [Bindable]
      public final class FlexContactsModelLocator implements IModelLocator
      {
            private static var instance:FlexContactsModelLocator;

            public function FlexContactsModelLocator()
            {
                  if( instance != null ) {
                        throw new Error( "Error: " +
                                    "FlexContactsModelLocator can only be instantiated via getInstance() method!" );
                  }
                  FlexContactsModelLocator.instance = this;
            }

            public static function getInstance():FlexContactsModelLocator
            {
                  if( instance == null ) {
                        instance = new FlexContactsModelLocator();
                  }
                  return instance;
            }

            public var contactsCollection:ArrayCollection;
            public var selectedContact:ContactVO;
            public var isLoggedIn:Boolean = false;

            public var workflowState:uint = 0;
            public static const LOGIN_SCREEN:uint = 0;
            public static const MAIN_SCREEN:uint = 1;
      }
}
```

Listing 3. *Services.mxml*

```
<?xml version="1.0" encoding="utf-8"?>
<cairngorm:ServiceLocator xmlns:mx="http://www.adobe.com/2006/mxml"
                          xmlns:cairngorm="com.adobe.cairngorm.business.*" >

   <mx:RemoteObject
         id="AmfphpService"
         destination="amfphp"
         endpoint="http://codegen/amfphp/gateway.php"
         source="ContactsService"
         makeObjectsBindable="true"
         showBusyCursor="true"/>

</cairngorm:ServiceLocator>
```

and where ever you go (*view*) you are going to have access to that same amount (*data*) that is inside your bank (`ModelLocator`).

- Identify Model Locator:
  - Central data bank for a application
  - Singleton means that only one instance of this class should exist
  - Does not contain any business logic
  - Supports data binding
  - Updated by the Command
  - Accessed by the View
- Create `FlexContactsModelLocator.as`, see (Listing 2)
- Discuss – ModelLocator

To access data that is stored inside of the model you first create a Bindable private variable inside of your view or any class that needs to access the data, set the variable type to the name of your ModelLocator class (`FlexContactsModelLocator.as`).

Then you set that variable equal (=) to `FlexContactsModelLocator.getInstance()`, creating an instance of your model in which allows access to any part of that data bank.

Service Locator

The Cairngorm Service Locator provides a clean and effective way of defining multiple server side sources in a single component; you can define any type of (RPC) remote procedure calls such as HTTP Services, *Web Services* (WSDL), *Remoting Services* (AMF), etc.

The Service Locator is not created in the form of a class, but as a component based off of the Cairngorm Service Locator package. Creating this is just as simple as creating a new component, but instead of setting the based on to a VBox or a Canvas you set it to ServiceLocator, finishing it off by declaring the xml namespace to *com.adobe.cairngorm.business.**.

- Identify – Service Locator:
  - Only one per applcication
  - Based off a singleton class
  - Central area for all server side destinations
  - Called by the Delegate
- Create `Services.mxml`, see (Listing 3)
- Discuss – Service Locator:

Creating a new component that is based on the ServiceLocator will look like above then inside of that component you simply just place all of your services, be sure to give them the proper id for referencing them later on.

The Delegate

The Delegate in its simplest form is holder of the services in which you want to invoke on the server side. Delegates use the ServiceLocator for locating the service then

Listing 4. *GetContactsEvent.as*

```
package com.jonniespratley.flexcontacts.business
{
	import com.adobe.cairngorm.business.ServiceLocator;
	import com.jonniespratley.flexcontacts.vo.ContactVO;

	import mx.rpc.AsyncToken;
	import mx.rpc.IResponder;

	public class FlexContactsDelegate
	{
		private var responder:IResponder;
		private var service:Object;

		public function FlexContactsDelegate( responder:IResponder )
		{
			this.service = ServiceLocator.getInstance().getRemoteObject(
			    "AmfphpService" );
			this.responder = responder;

			trace( 'Locating Service' );
		}

		public function getContacts():void
		{
			var token:AsyncToken = service.getAllContacts();
				token.addResponder( responder );

			trace( 'Calling getContacts()' );
		}
	}
}
```

Listing 5. *GetContactsEvent.as*

```
package com.jonniespratley.flexcontacts.events
{
	import com.adobe.cairngorm.control.CairngormEvent;

	import flash.events.Event;

	public final class GetContactsEvent extends CairngormEvent
	{
		public static const GET_CONTACTS_EVENT:String = "GET_CONTACTS_EVENT";

		public function GetContactsEvent()
		{
			super( GET_CONTACTS_EVENT );

			trace( 'GetContactsEvent Dispatching' );
		}

		override public function clone():Event
		{
			return new GetContactsEvent();
		}
	}
}
```

calling methods on that service. The Delegate class has generally one argument when being instantiated, which is type IResponder, this provides any class using this Delegate to handle the results of the service in a nice manner.

- Identify – Service Delegate:
  - Holds server side methods with required arguments
  - Delegate class includes a responder to hold the result for command possessing
  - Does not update the model
  - One Delegate per service that is declared in the Serivce Locator
- `Create FlexContactsDelegate.as`, see (Listing 4)
- Discuss – Service Delegate

Listing 6. *GetContactsCommand.as*

```
package com.jonniespratley.flexcontacts.commands
{
	import com.adobe.cairngorm.commands.ICommand;
	import com.adobe.cairngorm.control.CairngormEvent;
	import com.jonniespratley.flexcontacts.business.FlexContactsDelegate;
	import com.jonniespratley.flexcontacts.events.GetContactsEvent;
	import com.jonniespratley.flexcontacts.model.FlexContactsModelLocator;
	import com.jonniespratley.flexcontacts.vo.*;

	import mx.collections.ArrayCollection;
	import mx.controls.Alert;
	import mx.rpc.IResponder;
	import mx.rpc.events.FaultEvent;
	import mx.rpc.events.ResultEvent;

	public class GetContactsCommand implements ICommand, IResponder
	{
		private var model:FlexContactsModelLocator = FlexContactsModelLocator.get
			Instance();

		public function execute( event:CairngormEvent ) : void
		{
			trace( 'GetContactsCommand Executing' );

			var evt:GetContactsEvent = event as GetContactsEvent;
			var delegate:FlexContactsDelegate = new FlexContactsDelegate( this
			    );
			      delegate.getContacts();
		}

		public function result( data:Object ) : void
		{
			trace( 'GetContactsCommand Result Handling' );

			var result:ResultEvent = data as ResultEvent;
			var contactArray:Array = data.result as Array
			var tempAC:ArrayCollection = new ArrayCollection();

			for ( var o:Object in contactArray )
			{
				tempAC.addItem( new ContactVO( contactArray[ o ] ) );
			}
			model.contactsCollection = tempAC;
		}

		public function fault( fault:Object ):void
		{
			var faultEvt:FaultEvent = fault as FaultEvent;
			Alert.show( faultEvt.fault.toString(), "Error" );
		}
	}
}
```

To use a Delegate you will generally create a variable inside of your Command that is of type FlexContactsDelegate, after the type you want to create a new instance of that Delegate. Now you will be able to call on a service that is declared inside of the delegate. Thus making a call to a server side service. See Commands below for more of a example.

Cairngorm Events

Cairngorm Events are just simple classes that dispatch to your application that something has happened, such as a user interaction of some sort or the completion of a process. When these events are dispatched via the View layer, the `FrontController` is the one that is listening for these events to happen, once an event has happened the controller then maps the event to the proper command for processing. Use cases of an application are usually the best events to create, (`AddToCart`, `Checkout`, `PaymentComplete`, etc)

Events must be unique, and always should include a static variable with the value set to the name of the event. Events that need to send data can just create arguments inside of the event along with public variables to hold the value of that data. Then when the processing of that event occurs, the processor has access to the public variables that are inside the event, holding the specific values of that event.

- Idenfity – Cairngorm Event:
  - View layer can create and dispatch Cairngorm Events
  - Control layer can also create and dispatch Cairngorm Events
  - Control layer manages and process Cairngorm Events
  - Dispatching Cairngorm Events should be 1-way to the Control layer
  - Cairngorm Events dispatch themselves to the Controller
- Create `GetContactsEvent.as`, see (Listing 5).
- Discuss Cairngorm Event.

When creating events keep in mind that events and commands have a one to one (1 – 1) relationship. When you create an event, it is recommend creating a command that carries out the process of when that event happens.

For instance you create a GetContactsEvent.as class be sure to create a `GetContactsCommand.as` class as well, when this event is dispatched the corresponding command will begin to process and carry out the functionality.

Listing 7. *LoginCommand.as*

```
package com.jonniespratley.flexcontacts.commands
{
	import com.adobe.cairngorm.commands.ICommand;
	import com.adobe.cairngorm.control.CairngormEvent;
	import com.jonniespratley.flexcontacts.events.GetContactsEvent;
	import com.jonniespratley.flexcontacts.events.LoginEvent;
	import com.jonniespratley.flexcontacts.model.FlexContactsModelLocator;
	import com.jonniespratley.flexcontacts.vo.*;

	import mx.controls.Alert;

	public class LoginCommand implements ICommand
	{
		private var model:FlexContactsModelLocator = FlexContactsModelLocator.getInstance();

		public function execute( event:CairngormEvent ) : void
		{
			trace( 'LoginCommand Executing' );

			var evt:LoginEvent = event as LoginEvent;
			if ( evt.aUser.username == "admin" && evt.aUser.password == "admin" )
			{
				model.isLoggedIn = true;
				model.workflowState = FlexContactsModelLocator.MAIN_SCREEN;
			} else {
				Alert.show( "Username: " + evt.aUser.username + "\nPassword was not valid.", "Login Error" );
				model.isLoggedIn = false;
				model.workflowState = FlexContactsModelLocator.LOGIN_SCREEN;
			}

		}
	}
}
```

Listing 8. *FlexContactsController.as*

```
package com.jonniespratley.flexcontacts.control
{
	import com.adobe.cairngorm.control.FrontController;
	import com.jonniespratley.flexcontacts.commands.*;
	import com.jonniespratley.flexcontacts.events.*;

	public final class FlexContactsController extends FrontController
	{
		public function FlexContactsController()
		{
			initilize();
		}

		private function initilize():void
		{
			this.addCommand( GetContactsEvent.GET_CONTACTS_EVENT, GetContactsCommand );
			this.addCommand( LoginEvent.LOGIN_EVENT, LoginCommand );

			trace( 'Registering Events to Commands' );
		}
		}
}
```

Using your custom Cairngorm Event is really simple, so instance inside your view of the application if you created this `GetContactsEvent.as` file and you wanted to dispatch this event to your application, letting the application know that this event happened, you simply use the following.

```
private function getContacts():void
{
var evt:GetContactsEvent = new
                        GetContactsEvent();
evt.dispatch();
}
```

Commands

The Command class then processes the event by running the Command class `execute()` method, which is an `ICommand` interface method. The event object may include additional data if required by the developer. The `execute()` method can update the central Model, as well as invoke a Service class which typically involves communication with a remote server.

The IResponder interface, which is also implemented by the Command class, includes onResult and onFault methods to handle responses returned from the invoked remote service.

Commands come in two flavors, one flavor is if your command implements IResponder then you must be calling a service inside of the delegate of some sort, then if you implement IResponder you must import the `mx.rpc.IResponder`, since the Cairngorm IResponder has been depreciated.

You include two more functions inside of your command, `result()` and `fault()` these functions will handle the results from the server wither it be a success result or a fail result, you can also event dispatch another event from your command, such as when you get a fault from the call you can dispatch a FaultEvent passing in your parameters that are from the fault.

- Identify – Cairngorm Command:
  - Updates Model
  - Calls on Delegate
  - One to One Event
- Create `GetContactsCommand.as`, see (Listing 6).
- Discuss: `GetContactsCommand.as`

In this type of command you might notice something different, this command implements two interfaces, these interfaces are required when you command is going to be working with a remote service (RPC). In the result of this command we are taking the result object, and passing it to a instance of a new value object, then adding it to a temporary array collection before updating the model.

This lets us have an array collection of strong typed objects inside our model for any part of the application to reference this.

Doing a little more with commands

- Create `LoginCommand.as`, see (Listing 7).
- Discuss– `LoginCommand.as`

With this command we are simply checking wither or not the values passed with the event match what we are declaring in the `execute()` function of this command. The LoginEvent.as class has one argument which is a UserVO value object, when this event is dispatched it passes a user object holding username and password values.

When the command receives the event it checks if the values of the UserVO.username and password match what is declared above.

Listing 9. *LoginForm.mxml*

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox
      width="100%"
      height="100%"
      styleName="padding"
      horizontalAlign="center"
      verticalAlign="middle"
      xmlns:mx="http://www.adobe.com/2006/mxml">

      <mx:Script>
            <![CDATA[
                  import com.jonniespratley.flexcontacts.vo.UserVO;
                  import com.jonniespratley.flexcontacts.events.LoginEvent;

                  private function checkLogin():void
                  {
                        var user:UserVO = new UserVO( txt_username.text, txt_
                        password.text );

                        var e_login:LoginEvent = new LoginEvent( user );
                              e_login.dispatch();
                  }

            ]]>
      </mx:Script>

      <mx:Form>
            <mx:FormHeading label="User Login"/>
            <mx:FormItem label="Username:" required="true" width="100%">
                  <mx:TextInput id="txt_username"
                        text="admin"
                        width="100%"/>
            </mx:FormItem>

            <mx:FormItem label="Password:" required="true" width="100%">
                  <mx:TextInput id="txt_password"
                        text="admin"
                        displayAsPassword="true"
                        width="100%"/>
            </mx:FormItem>

            <mx:FormItem width="100%" horizontalAlign="right">
                  <mx:Button
                        label="Login"
                        click="checkLogin()"/>
            </mx:FormItem>
      </mx:Form>
</mx:VBox>
```

If the values match, then the command sets the workflowState variable inside of the model to the static variable located also inside the model, thus changing the view of the application.

Controller

The Controller is the most sophisticated part of the Cairngorm architecture. The Controller layer is implemented as a singleton FrontController. The FrontController instance, which receives every View-generated event, dispatches the events to the assigned Command class based on the event's declared type.

The Front Controller is a central place where all the commands are mapped to the relevant events. Once an event is dispatched, the Front Controller finds the corresponding command to execute.

- Identify – Front Controller:
  - Catches dispatched events and forwards each event to the appropriate Command for processing
  - Views dispatch events to this layer
  - All commands and events are registered with the controller
- Create `FlexContactsController.as`. see (Listing 8).
- Discuss – Front Controller:

In the Front Controllers constructor it is calling on the initialize method that is handling all of the registering events to commands, this is accomplished by using the `addCommand()` method inherited from the Cairngorm controller to link events and corresponding commands.

View

The View is one or more Flex components such as a button, panel, data grid, combo box, etc. all bundled together as one component. This is what the user will see when using the application, data that is displayed is bound to the Model Locator or other Bindable variables that essential get bound to the model, views also generate events, Cairngorm Events.

- Identify:

The view of the application is what the user will interact with, you as the programmer need to understand some of the gestures a user might use to.

Listing 10. *ContactsList.mxml*

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox
	height="100%"
	width="100%"
	creationComplete="getContacts()"
	styleName="padding"
	xmlns:mx="http://www.adobe.com/2006/mxml">

	<mx:Script>
		<![CDATA[

			import com.jonniespratley.flexcontacts.events.GetContactsEvent;
			import com.jonniespratley.flexcontacts.model.FlexContactsModelLocator;

			[Bindable] private var model:FlexContactsModelLocator = FlexContactsModelLocator.getInstance();

			private function getContacts():void
			{
				var e_getContacts:GetContactsEvent = new GetContactsEvent();
					e_getContacts.dispatch();
			}

		]]>
	</mx:Script>

	<mx:DataGrid id="dg_contacts"
		height="100%"
		width="100%"
		dataProvider="{ model.contactsCollection }">
		<mx:columns>

			<mx:DataGridColumn headerText="ID" dataField="id" width="50"/>
			<mx:DataGridColumn headerText="First" dataField="name"/>
			<mx:DataGridColumn headerText="Address" dataField="address"/>
			<mx:DataGridColumn headerText="City" dataField="city"/>
			<mx:DataGridColumn headerText="State" dataField="state"/>
			<mx:DataGridColumn headerText="Phone" dataField="phone"/>
			<mx:DataGridColumn headerText="Email" dataField="email"/>
		</mx:columns>
	</mx:DataGrid>
</mx:VBox>}
```

- Create:
  - `LoginForm.mxml`, see (Listing 9).
  - `ContactsList.mxml`, see (Listing 10).
  - `MainView.mxml`, see (Listing 11)
  - `FlexContacts.mxml`, see (Listing 12).
- Discuss – View

Listing 11. *MainView.mxm*

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox
      width="100%"
      height="100%"
      styleName="padding"
      xmlns:mx="http://www.adobe.com/2006/mxml"
      xmlns:view="com.jonniespratley.flexcontacts.view.*"
      xmlns:components="com.jonniespratley.flexcontacts.view.components.*">

      <mx:Script>
            <![CDATA[
                  import com.jonniespratley.flexcontacts.events.LogoutEvent;
                  import com.jonniespratley.flexcontacts.model.FlexContactsModelLocat
                        or;

                  [Bindable] private var model:FlexContactsModelLocator = FlexContacts
                        ModelLocator.getInstance();

                  private function logout():void
                  {
                        var evt:LogoutEvent = new LogoutEvent();
                              evt.dispatch();
                  }

            ]]>
      </mx:Script>

      <mx:ApplicationControlBar width="100%" visible="{ model.isLoggedIn }"
                  includeInLayout="{ model.isLoggedIn }">
            <mx:LinkButton label="Logout" click="logout()"/>
      </mx:ApplicationControlBar>

      <mx:ViewStack width="100%" height="100%" selectedIndex="{ model.workflowState }">
            <components:LoginForm id="loginForm"/>
            <components:ContactList id="contactList"/>
      </mx:ViewStack>

</mx:VBox>
```

Listing 12 *FlexContacts.mxm*

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
layout="vertical"
      xmlns:mx="http://www.adobe.com/2006/mxml"
      xmlns:business="com.jonniespratley.flexcontacts.business.*"
      xmlns:control="com.jonniespratley.flexcontacts.control.*"
      xmlns:view="com.jonniespratley.flexcontacts.view.*">

      <business:Services id="services"/>

      <control:FlexContactsController id="controller"/>

      <view:MainView id="view"/>

</mx:WindowedApplication>
```

These events can come from any interaction that the developer chooses such as button clicks, component initiation, drag-n-drop, mouse clicks, etc. These events get dispatched via the dispatch () method of the Flex framework; once this event is dispatched the controller catches that event and then maps it to the correct command, executing that action.

Inside of `LoginForm.mxml` – You can see that the view has no idea what happens when the event is dispatched, or what is suppose to happen, all it knows what to do is to populate a user value object, with a simple username and password, and dispatch a event. This is an excellent example of separating the business logic from the view logic of the application.

Inside of `ContactsList.mxml` – You can also see that once again, the view is doing nothing about how the application is going to work; all the view is doing is displaying data to the user. This is another great example of separation from model, view, and controller. The view just references the model, and dispatches events. And that is all that the view knows about.

Inside of `MainView.mxml` – You bring the two components LoginForm.mxml and ContactsList.mxml together inside of this component; this is the top-level component that will be the holder of the applications views. There is a view stack inside of the MainView.mxml component that is binding to the current workflow state of the application. This provides full control over what part of the application is visible, and what part is accessible to the current user. If the user was to logout, changing what the user sees is easy as changing the workflow state variable inside of the model. Keeping everything in sync and separated from each other.

Inside of `FlexContacts.mxml` – You bring in the service locator, the front controller, and the main view. Keeping only one instance of each item makes your application follow the standards of the framework, as well as keeping your applications architecture ready for any future enhancements it may need in the future.

JONNIE SPRATLEY

Flex Programmer from Sacramento, California
Age: 22
Have been programming with Flex/ActionScript for about over Two years
Enjoys a nice Caramel Frappichno and the latest Flash Flex Magazine at the local bookstore.
Website: http://jonniespratley.com