1

Introducing Cairngorm

In this chapter you are introduced to Cairngorm, its history, and basic structure. You will start by examining some definitions of what Cairngorm is. Then you will be given a brief history of how Cairngorm came to be. Finally, you will be given a broad overview of the basic parts that make up Cairngorm, its organization, and its handling of application logic.

What Is Cairngorm?

Depending on what source you go to, you will find varying descriptions of what Cairngorm is.

The Adobe Labs web site (the current home of the Cairngorm project) describes it as follows:

> Cairngorm is the lightweight micro-architecture for Rich Internet Applications built in Flex or AIR. A collaboration of recognized design patterns, Cairngorm exemplifies and encourages best-practices for RIA development advocated by Adobe Consulting [and] encourages best-practice leverage of the underlying Flex framework, while making it easier for medium to large teams of software engineers [to] deliver medium to large scale, mission-critical Rich Internet Applications (`http://opensource.adobe.com/wiki/display/cairngorm/Cairngorm`).

The Wikipedia entry for Cairngorm has the following description:

> Cairngorm is based on the MVC model. It is specifically designed to facilitate complex state and data synchronization between the client and the server, while keeping the programming of the View layer detached from the data implementation (`http://en.wikipedia.org/wiki/Cairngorm_(Flex_framework)`).

A further description of its intended use can be found at `http://opensource.adobe.com/wiki/display/cairngorm/About`:

> The Cairngorm micro-architecture is intended as a framework for Enterprise RIA developers. . . . The benefits of the Cairngorm architecture are realized when developing complex RIA applications with multiple use-cases and views, with a team of developers, and with a multi-disciplinary development team that includes designers as well as creative and technical developers.

You probably won't fully understand these descriptions at this point. Later sections of the book will explore the concepts that inform Cairngorm (such as design patterns) and the individual pieces that make up the Cairngorm framework. However, before moving on to examining the concepts and structure of the Cairngorm framework, I want to point out a few things that will make those discussions more informative.

You will see Cairngorm referred to as both a framework and a micro-architecture (as can be seen in the preceding quotations). Both are technically correct, with *micro-architecture* being the more specific of the two terms (that is, the one that describes Cairngorm on a more detailed level). The differences between a framework and an micro-architecture will be discussed in later chapters. For now, just be aware that if you see something describing "the Cairngorm micro-architecture," it is referring to the same thing as the Cairngorm framework.

Regardless of the differences you may find in descriptions of Cairngorm, most of them agree on the following:

- Cairngorm is targeted at the rapid development of rich Internet applications (RIAs) using the Flex framework.
- Cairngorm facilitates team development.
- Cairngorm is suited for medium- to large-scale projects in which code needs to be separated and organized.
- The Cairngorm framework endorses and encourages best practices for building Flex-based RIAs.

You will see in later chapters of this book that even these points are not without their critics and challengers.

A Brief History of Cairngorm

Cairngorm currently exists as an open source project on the Adobe Open Source web site (`http://opensource.adobe.com/wiki/display/cairngorm/Cairngorm`). However, Cairngorm was not created by Adobe. In fact, its creation is not even tied to the origin of the Flex framework.

Cairngorm was created by a company named iteration::two in Edinburgh, Scotland; the company was founded by Steven Webster and Alistair McLeod. The name Cairngorm actually comes from a range of mountains in the eastern Highlands of Scotland. Iteration::two was eventually was acquired by Macromedia and became Adobe Consulting.

Cairngorm's roots stretch back as far as Flash MX in the book *Reality J2EE—Architecting for Flash MX* (Steven Webster and Alistair McLeod, Pearson Education, 2003). As the technologies for RIA matured to ActionScript 2.0 and Flash Remoting with Flash MX 2004, the ideas behind Cairngorm were revisited in the section on "ActionScript 2.0 design patterns for rich Internet applications" in the *ActionScript 2.0 Dictionary* (Steven Webster and Alistair McLeod, Macromedia Press, 2003). The ideas expressed in these books originated from a subset of design patterns, advocated by Sun Microsystems, found in the Core J2EE Pattern Catalog (`http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html`).

Essentially, as Flash started introducing such capabilities as remoting, it started to face many of the same challenges previously faced in the world of J2EE RIA development. Therefore, many of the same solutions applied in the world of J2EE were now applicable to Flash development. By *solutions*, we are referring to design patterns, the specifics of which will be covered in the next chapter.

It was not until later that the aforementioned ideas were applied to the Flex framework in the book entitled *Developing Rich Clients with Macromedia Flex* (Steven Webster and Alistair McLeod, Macromedia Press, 2004). This was the first book on best practices for enterprise RIA development with Adobe Flex.

Since then Cairngorm has become one of the best-known frameworks for Flex development, and at the time of this writing it is the only one found on the Adobe Open Source web site. It officially became an open source project at the Macromedia MAX2004 conference in New Orleans in November 2004.

The first open source release of the Cairngorm framework was labeled the 0.95 version for Flex 1.5, with an updated 0.99 version (with new features) released soon thereafter.

The version at the time of writing is Cairngorm 2.2, released on April 30, 2007.

You can read about future plans for Cairngorm in a blog post by Steven Webster at `http://weblogs.macromedia.com/swebster/archives/2008/08/cairngorm_3_-_a.html`.

Overview of Cairngorm

This section is meant as a broad overview of the major components that make up the Cairngorm framework. Subsequent chapters will examine each of these parts in more detail.

Cairngorm Source Code

Because it is an open source project on the Adobe Open Source web site, you have full access to the source code for Cairngorm. The code is stored in a Subversion repository. Directions for installing a Subversion client and for checking out the code can be found at `http://opensource.adobe.com/wiki/display/cairngorm/Get+Source+Code`.

It is not necessary to check out the source code for the purposes of this book, but if you want to, that option is available to you.

Cairngorm Organization and Documentation

The documentation and class structure outline for Cairngorm can be found at `http://cairngormdocs.org/docs/cairngorm_2_2_1/index.html`.

Much like the standard Flex framework, Cairngorm's classes are organized into packages. These are:

- com.adobe.cairngorm
- com.adobe.cairngorm.business
- com.adobe.cairngorm.commands
- com.adobe.cairngorm.control

- com.adobe.cairngorm.model
- com.adobe.cairngorm.view
- com.adobe.cairngorm.vo

It is not necessary to know the contents of each package in order to build a Cairngorm project, but knowing the basic structure will help you understand how the framework is organized and, should you need to locate a particular class, make that easier to do. When applicable, this documentation will be referenced as you explore the major components of Cairngorm that are commonly used to build Cairngorm projects.

For now, here is a brief explanation of the classes and interfaces that you can find in each package:

com.adobe.cairngorm

The com.adobe.cairngorm package contains the following classes:

- **CairngormError:** An error class thrown when a Cairngorm error occurs
- **CairngormMessageCodes:** Stores Cairngorm message codes

com.adobe.cairngorm.business

The com.adobe.cairngorm.business package contains the following classes:

- **AbstractServices:** Used to manage all services defined on the IServiceLocator instance
- **ServiceLocator:** Allows services to be located and security credentials to be managed

It also contains the following interfaces:

- **IServiceLocator:** Defines an interface for the service locator
- **IServices:** Defines an interface for managing services on an IServiceLocator
- **Responder:** Implemented by classes to handle data returned as the result of a service call to the server. Deprecated as of Cairngorm 2.1 and replaced by mx.rpc.IResponder

com.adobe.cairngorm.commands

The com.adobe.cairngorm.commands package contains the following class:

- **SequenceCommand:** A "pseudo-abstract" (since ActionScript has no real concept of abstract classes) base class that can be extended when you wish to chain commands together for a single user gesture, or establish some simple form of decision-based workflow

It also contains the following interfaces:

- **Command:** Enforces the contract between the FrontController and concrete command classes in your application. Deprecated as of Cairngorm 2.1 and replaced by com.adobe.cairngorm .commands.ICommand
- **ICommand:** Enforces the contract between the FrontController and concrete command classes in your application

com.adobe.cairngorm.control

The com.adobe.cairngorm.control package contains the following classes:

- **CairngormEvent:** Used to differentiate Cairngorm events from events raised by the underlying Flex framework or Flash Player
- **CairngormEventDispatcher:** A singleton class used by the application developer to broadcast events that correspond to user gestures and requests
- **FrontController:** A base class for an application-specific FrontController, able to dispatch control to appropriate command classes following particular user gestures

com.adobe.cairngorm.model

The com.adobe.cairngorm.model package contains the following class:

- **ModelLocator:** Marker interface used to mark the custom ModelLocator. Deprecated as of Cairngorm 2.1 and replaced by com.adobe.cairngorm.model.IModelLocator

It also contains the following interface:

- **IModelLocator:** The new Marker interface used to mark the custom ModelLocator

com.adobe.cairngorm.view

The com.adobe.cairngorm.view package contains the following classes:

- **ViewHelper:** Used to isolate command classes from the implementation details of a view. Deprecated as of Cairngorm 2.1
- **ViewLocator:** A singleton class used to retrieve ViewHelper classes that can manipulate (get/set/switch) the user interface of a Cairngorm RIA. Deprecated as of Cairngorm 2.1

com.adobe.cairngorm.vo

The com.adobe.cairngorm.vo package contains the following class:

- **ValueObject:** A marker interface that improves readability of code by identifying the classes within a Cairngorm application that are to be used as value objects for passing data between tiers of an application. Deprecated as of Cairngorm 2.1 and replaced by com.adobe.cairngorm.vo.IValueObject

It also contains the following interface:

- **IValueObject:** A new marker interface that improves readability of code by identifying the classes within a Cairngorm application that are to be used as value objects for passing data between tiers of an application

As you can see from the class and interface descriptions, classes and interfaces are still included in the framework despite having been deprecated.

You can also see that a number of interfaces (e.g., IValueObject) simply serve as markers to identify types of classes.

Additionally, as you will see in the next section, which discusses the major components for Cairngorm, not all these classes are directly used in building Cairngorm projects. However, they are all part of the Cairngorm framework and the documentation exists should you need to refer to it.

Major Components of Cairngorm

In the article entitled *Flex 3: Introducing Cairngorm* (Adobe Customer Training/Partner Enablement Group), Thomas Burleson of Universal Mind, and Leo Shuman of Adobe Customer Training, under the guidance of Matt Boles (`http://www.adobe.com/devnet/flex/articles/introducing_cairngorm.html`), the major components of Cairngorm are defined as follows:

- ModelLocator
- Services
- Commands
- Events
- Controller

This list is a good starting point for understanding the components of Cairngorm, but it mixes specific components with general concepts. For example, the first item in the list is an actual class name found in the framework; services are used in the framework but are actually represented by a class called the ServiceLocator; commands and events are types of classes; and the controller is represented by a class called the FrontController.

In addition to components in this list, there are two other types of classes commonly used in Cairngorm that are not included in the list. These are value objects and delegates. The most likely reason for their omission from the list is that their use is generally considered optional. However, there is a com.adobe.cairngorm.vo package for value objects and, while there is no delegate class or interface in Cairngorm, their use is common enough to warrant examination as well.

This book will therefore be defining the major components of Cairngorm in terms of the classes commonly used in Cairngorm projects:

- ModelLocator
- ServiceLocator
- Commands
- Events
- FrontController
- Value objects
- Delegates

A chapter will be devoted to each of these components. For now, here is a brief description of each.

ModelLocator

The ModelLocator serves as a central repository for shared application data. It allows components of the application to have access to the same set of data to ensure synchronization. Additionally, through Flex's data binding mechanism, it notifies views of any changes to the data so that the views can update themselves to reflect those changes.

ServiceLocator

Most RIA applications require communication with a backend system. This communication with the backend is generally accomplished with one of the remoting classes, such as HTTPService, WebService, or RemoteObject. The ServiceLocator provides a centralized location for access and sharing such services so that they do not have to be declared in each component that needs to access the service.

Commands

Command classes are triggered by Cairngorm event classes to trigger business logic and update the ModelLocator. In standard Cairngorm practice only a command should ever update the ModelLocator.

Events

Event classes are used to trigger command classes and pass along any data they require.

FrontController

The FrontController provides a centralized location for defining how Cairngorm events are handled. This class registers event-command relationships and triggers the registered command for a given event when that event is fired.

Value Objects

Value objects are classes that represent conceptually related sets of data. Value objects contain public properties that represent the individual pieces of data being represented. They are commonly used to represent data in views by binding to their properties, and to transfer related sets of data between classes in the application and the server.

Delegates

Delegates are classes responsible for accessing remote services. They serve as proxy objects that take care of the details of accessing a particular service and return the data from the service via responder functions.

Basic Cairngorm Logic Flow

Now that you have a sense of the major components that make up Cairngorm, the following is a broad overview of how logic is implemented in a Cairngorm application.

The basic chain of events in a Cairngorm application can be seen in Figure 1-1.

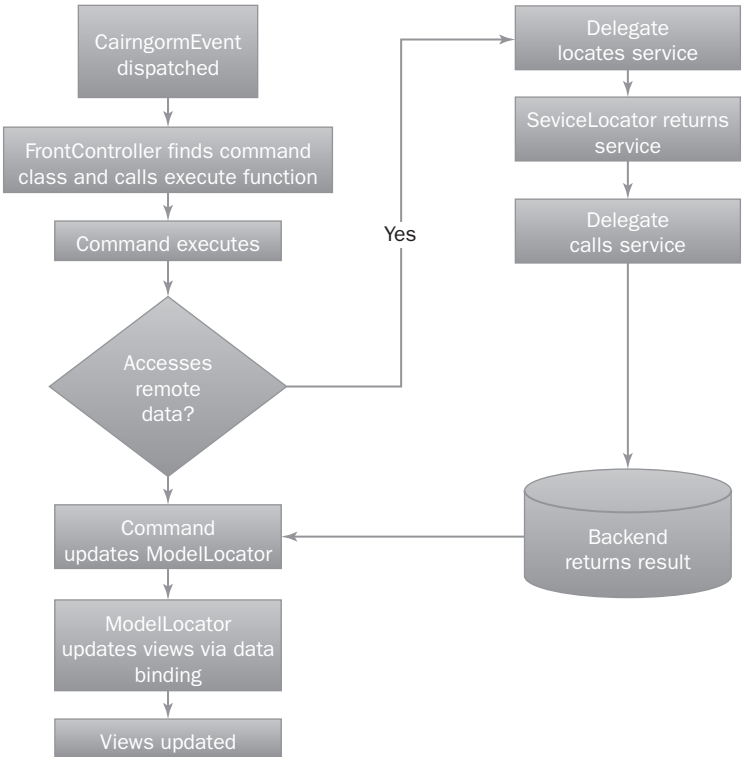

Figure 1-1

A view (or in some cases some other event, such as the loading of the application) dispatches a Cairngorm event. When this occurs, the FrontController intercepts the event and triggers the command class registered for that event. At this point the command class can do one of several things.

If no communication with the backend is required, the command may simply update the ModelLocator, causing the views to be updated via data binding.

If delegate classes are being used, a function on the delegate class will be called to perform actions on the backend via a remoting class found in the ServiceLocator. In this case the command acts as a responder to the remote call by registering itself with the delegate class. When the delegate function completes its call, the command class is passed the status of the call and any data returned from it via responder functions.

If the command class is directly calling the remote service, it performs the lookup on the ServiceLocator class and directly registers itself as a responder on the remote service. Once again, the results of the call are handled via responder functions defined in the command class.

Once the command class has performed any remote calls, it updates the ModelLocator. Once again, this causes any views that are binding to properties on the ModelLocator to update themselves.

Cairngorm Project Organization

Not only is the Cairngorm framework organized into packages, but so are projects created with the framework.

When you build a project in Cairngorm, the classes you create get organized into specific packages. You do this by creating sub-folders in your project's src folder. The particular organization of these packages can vary greatly. For example, the Cairngorm Store application (`http://www.cairngormdocs.org/exampleApps/CairngormStoreWeb2_1.zip`) organizes its packages as seen in Figure 1-2.

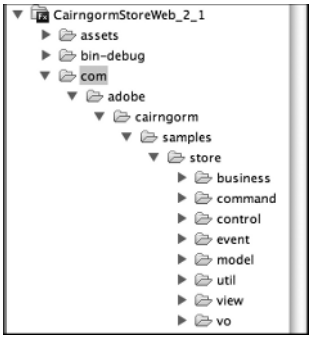

Figure 1-2

However, if you look at the FStop application included in the article "Flex 3: Introducing Cairngorm" (cited earlier in this chapter), it organizes its packages as seen in Figure 1-3.

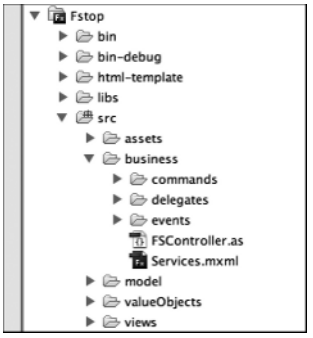

Figure 1-3

Both of these package structures work; however, it is not always clear where you would expect to find a given type of class. For example, in the Cairngorm Store, events are in a package on the same level as the business package, while in the FStop application, events are a subpackage of the business package.

To avoid any confusion as to where a particular type of class should be located, I am suggesting the following package structure:

- commands (stores command classes)
- controllers (stores the FrontController class)
- delegates (stores service delegate classes)
- events (stores event classes)
- models (stores ModelLocator classes)
- services (stores the ServiceLocator class)
- valueobjects (stores value object classes)
- views (stores MXML views)

In the above package structure, each type of class is contained in a package that describes the type of classes it contains. This is the package structure that you will use when creating the sample application in this book.

However, as we explore the various parts of the Cairngorm framework, the various locations that the classes can be located in will be pointed out, using the Cairngorm Store and FStop applications as examples. This knowledge can be useful if you inherit a project from another developer.

Benefits of Using Cairngorm

Now that you have a basic idea of the structure, components, and application logic flow of a Cairngorm project, it's time to examine some of the benefits the framework provides.

In the "Flex 3: Introducing Cairngorm" article cited earlier, the following purposes of Cairngorm are defined:

- Cairngorm is an approach to organizing and partitioning:
  - Code and packages
  - Component functionality and roles
- Cairngorm is a "best practice" methodology for Flex software design and development.
- Cairngorm encourages developers to identify, organize, and separate code based on its roles/responsibilities.

Many of these points have already been mentioned in the quoted descriptions of Cairngorm earlier in this chapter, but what has not been discussed are the benefits that arise from this organization and best-practices methodology.

The "Flex 3: Introducing Cairngorm" article outlines several benefits of using Cairngorm. These include:

- Cairngorm is ideal for team development, since it allows designers, component developers, and data-services developers to work in parallel.
- Maintenance, debugging, and the adding or updating of features are easier than with a project created using just the standard Flex framework.

Both of these benefits arise from the organization provided by the Cairngorm framework.

Since code is organized by roles/responsibilities and is implemented via the Cairngorm methodology, designers can be working on views while component developers and service developers work on their respective parts. When these pieces of code are brought together into the application you can be pretty sure they will be compatible, since they are all using the standards defined by the Cairngorm framework. This allows application features to be developed in parallel, shortening development timelines.

Maintenance, debugging, and the adding or updating of features become easier because you know where to look to find the code related to a given feature. If you have ever developed an application in Flash, or looked at one developed by someone else, you most likely have spent time searching for code, since it can be spread across frames or hidden in deeply nested movie clips. The same situation can occur in Flex if code is scattered across the application or application components. However, when using Cairngorm you can quickly track down specific code because of the predictable way Cairngorm handles application logic. If you see that a view is dispatching a given event, you can look for the command registered for that event in the FrontController and quickly see what logic the command is triggering. Adding and updating features are simply matters of modifying existing Cairngorm classes or adding new ones.

Summary

In this chapter you were introduced to the Cairngorm framework, including its history, guiding principles, major components, application logic flow, and organization.

Cairngorm promotes the separation of code by roles/responsibilities and facilitates team development, in that different parts of the application can be worked on in parallel.

As I mentioned in the history section, Cairngorm borrows many of its solutions from the J2EE world. If you are coming from a computer programming background or have previously worked with Java or some other object-oriented language, many of these solutions may be familiar to you.

If you are coming from more of an ActionScript development background, they may be less familiar, since ActionScript is a less rigid language than Java and standards and practices still vary greatly in the world of ActionScript development.

Regardless of your current experience, the beginning chapters of this book discuss the underlying principles that inform Cairngorm and look in more detail at the major components defined in this chapter.

After examining these components, you will complete a brief sample application so that you can see examples of how these components work together to form application features.

Once you have completed the sample application, criticism and best practices related to Cairngorm will be discussed, as these will be more informative after you have worked with the framework.