Scotch on the rocks 4-6 June

Modularizing Flex applications - Cairngorm & Modules

Børre Wessel - Adobe Consulting

Adobe

Modularizing Flex applications

- Splitting up a Flex based applications
- Know your requirements, why do you want to split up your application
  - Large application?
  - Slow startup?
  - Share commonly used components/modules in multiple applications
- How easy is it to modularize my existing application?
  - Is code separated in the project
  - Is there a lot of code reuse between potential modules
- Separate teams are responsible for different parts of the system
  - Increase team productivity
  - Reduce compile time, if application hasn't changed, just compile the module

ApplicationDomain

- Your ActionScript class definition bucket

- Classes loaded from different SWF/SWC are loaded into a hierarchy of different application domains
  - Main application is the parent
  - Subsequently loaded SWF/SWC are loaded into separate child domains

- Classes used in different SWF/SWC needs to be loaded in the top level application to be able to be used across multiple child application domains

Different ways of modularizing

- RSL - Runtime Shared Libraries

  A collection of shared classes and components

- Standalone applications, SWF's

  Compiled separately, can run standalone

- mx:Module - Flex Modules

  Runs within a Flex application

RSL - Runtime Shared Libraries

- Runtime Shared Libraries enables code sharing between multiple applications.
  - Commonly used classes and components in one reusable SWC
  - If parts of your code changes frequently, RLS's can be a way to effectively update common functionality in multiple applications at the same time.
  - Cached in the browser cache.
- I have one large application, should I use RSL's to modularize it? NO
  - RSL's make your application bigger, more kb
  - Startup time is slow, all RSL's are loaded at startup
- To leave out definitions found in the RSL, compile the application using --external-library-path. Dependencies from the RSL will then be left out from the application

Modules

- mx:Modules - Loading components at runtime
- Used to split up single applications when size is a problem
- Effective for applications with a lot of UI code

- Leaving out classes from the main application
  - Use --link-report when compiling the main class, a list of all classes used in your application
  - Compiling your module, use the --load-externs compiler flag to omit dependencies already loaded by the main application

- Modules where not intended to used to share code between applications, RSL's are.
- Modules cannot run independently, it need to be loaded by an application, and potentially unloaded

Standalone SWF's

- Runs separately, or as part of other applications

- You need to know your applications well
- Remember what is loaded where, ApplicationDomain isn't there to make your life easy :)

- Not modular in all senses
  - Doesn't reduce size, classes used across SWF's are compiled into all of them
  - Needs to be manually loaded into separate ApplicationDomains

Other ways to modularize

- Flex framework cacheing
  - Reduces size significantly, “Hello World” from 350kB -> 50k
  - Includes all base components
- Loading assets at runtime
  - No need to embed, reduces size of the application
  - Use the Flash/Flex component kit!
  - Make use of complex assets at runtime instead of embedding
  - Can slow down compile time

Cairngorm

- MVC based micro architecture framework for Flex
- Cairngorm is based on existing well known patterns
  - MVC
  - Singleton pattern
  - Observer pattern
- Separation of concerns
- How can a framework help the project and the developers.
  - Code organization
  - Shared common terminology
  - Commonly used frameworks and patterns help developers understand the code
  - Developers get quicker up to speed on projects

Cairngorm and mx:Module's

- There are several different ways to use mx:Modules with Cairngorm
  - FrontController and ServiceLocator in main application is responsible for handling all events, and services
  - FrontController and ServiceLocator is injected into the module when loaded
  - Inject your data using model objects
  - Instantiate a model object when the module is loaded.
- How to use the mx:Modules with the ModelLocator
  - Don't store data directly on the ModelLocator, It is the "locator" of your model objects
  - Call getInstance in the main view once
  - Use data binding to inject your models into view components and modules
- Code example on how to inject models using interfaces in modules

Module interface to enable model injection

```
package com.adobe.ac
{
    import com.abode.ac.MyModel;

    public interface ICairngormModule
    {
        function set model( model:MyModel ):void;

        function get model():MyModel;
    }
}
```

Extended mx:Module

```
package com.adobe.ac
{
    import mx.modules.Module;

    public class CairngormModule extends Module implements ICairngormModule
    {
        [Bindable]
        public var _model:MyModel;

        public function set model( model:MyModel ):void
        {
            _model = model;
        }

        public function get model():MyModel
        {
            return _model;
        }

    }
}
```

Extended mx:Module

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

   <mx:HBox>
      <mx:Text text="Hello world" />
   </mx:HBox>

</ac:CairngormModule>
```

Loading the module

```
<mx:Script>
      <![CDATA[
        ...
        private function moduleReady( event:ModuleEvent ):void
        {
           var sotrModule:ICairngormModule
                = ( event.target as ModuleLoader ).child as ICairngormModule;
           if( sotrModule != null)
           {
              sotrModule.model = model;
           }
        }

        private function init():void
        {
           myModule.loadModule();
        }

      ]]>
   </mx:Script>

<mx:ModuleLoader
    id="myModule"
    url="sotr_module.swf"
    ready="moduleReady( event );"/>
```

Singletons - working with them

- The singletons in Cairngorm are there for a reason!
  - They are a single point of entry to be able to access commonly shared functionality and data
- Who are they?
  - ModelLocator
  - FrontController
  - ServiceLocator
- How do you work with them and not against them?
  - There is no reason why you cannot pass objects to your modules
  - Leveraging the power of interfaces, and data binding

Model objects, Presentation Model

- What kind of objects do I store on the ModelLocator?
  - Value objects, business objects...

- Presentation Model
  - Keeps application state
  - UI and data logic
  - View binds to data on the model, observes and updates UI components based on changes in the underlying data
  - The view has knowledge about the model
  - The model has no knowledge of the view

- Testable
- Easy to inject into the module, and easy to reset when the module is unloaded

Questions?

http://blogs.adobe.com/borre/

ADOBE® CONSULTING

Adobe