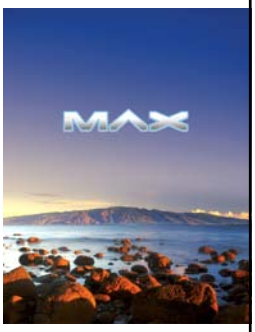
MAX 2006
Beyond Boundaries

Techniques for Delivering
Modular Flex Applications

Roger Gonzalez
Flex Compiler :: Adobe Systems



Flash movies and Flex applications

- SWF File Format
- Assets – Bitmaps, Vector Artwork, Sounds, Video, Fonts, etc.
- Code
- Frames
- Draw to Stage
- Streaming

vs

badger badger badger badger badger badger

SWF as streamed movie

- A SWF is organized into a stream of "frames" containing definitions and actions.
- A frame can be "played" only after it has been fully downloaded. Playing a frame executes all buffered instructions associated with the frame.
- The frame "play head" can be stopped or set to a different location.

Example

Problem: Animate a bouncing yellow "smiley face".

Bouncing Smiley-Face Animation: Approach 1

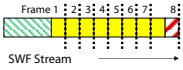
- Draw a face in different location on many frames, loop back to beginning when done.

Asset ("Symbol") Definitions
"Show Frame"
Render on Stage
Code

SWF files download left-to-right unless locale specifies otherwise

Bouncing Smiley-Face Animation: Approach 2

- Place instance of shared asset in new location on many frames, loop back to beginning when done.



Asset ("Symbol") Definitions
"Show Frame"
Render on Stage
Code

Bouncing Smiley-Face Animation: Approach 3

- Programmatically create an instance of a shared asset and animate it on the stage in response to timer events.
- Much more flexible…

Asset ("Symbol") Definitions
"Show Frame"
Render on Stage
Code

Approach 3 in the Real World

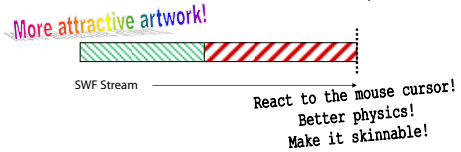
- Programmatically create an instance of a shared asset and animate it on the stage in response to timer events.
- Much more flexible… which leads to new requirements!

Asset ("Symbol") Definitions
"Show Frame"
Render on Stage
Code

Abject apologies to designers for representing their hard work via a cheesy rainbow text graphic.

Approach 3 in the Real World

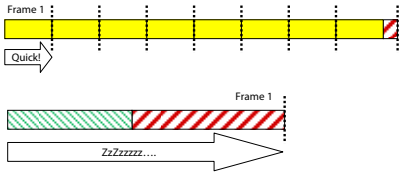
- Programmatically create an instance of a shared asset and animate it on the stage in response to timer events.
- Much more flexible… which leads to new requirements!



Asset ("Symbol") Definitions
"Show Frame"
Render on Stage
Code

Abject apologies to designers for representing their hard work via a cheesy rainbow text graphic.

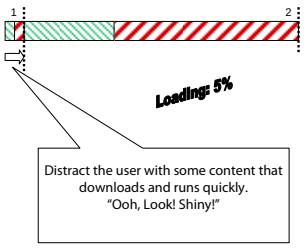
The cost of flexibility…



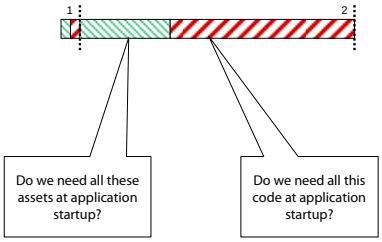
A lot more data needs to be downloaded (and initialized) before execution can start!

The application will be delayed by the tubes being filled by enormous amounts of material, enormous amounts of material.

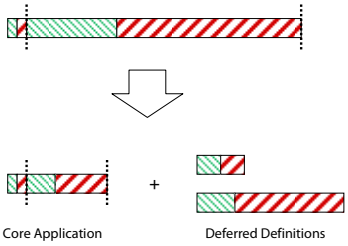
A "solution": the preloader frame



Another direction…



Two categories: immediate and "sometime later"



Factor out code and assets not needed at application startup!

Current Flex applications

- Monolithic
- Complex interdependencies
- Some code never runs until late (or never)
- Relies on preloader animation to distract user during application download
- Initialization cost managed via deferred instantiation

Future Flex Applications?

- Modular and/or streamed
- Functionality loaded on demand or asynchronously with core application
- Reduced interdependencies via interfaces
- Smaller "minimal buy-in" from frameworks via feature mixins?
- Dead code removal?

We can attack these problems today!

- Modular and/or streamed
- Functionality loaded on demand or asynchronously with core application
- Reduced interdependencies via interfaces
- ~~Smaller "minimal buy-in" from frameworks via feature mixins?~~
- ~~Dead code removal?~~

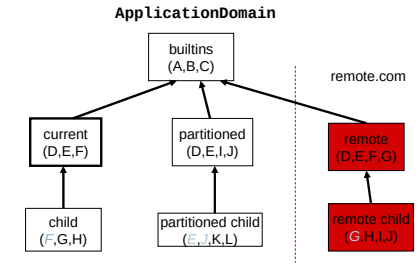
What do we need?

Know the enemy

```
--link-report <filename>
```

Writes XML file containing information on all definitions linked into your application

We need a diagram with boxes and arrows



Forms a hierarchy of containers holding AS3 definitions

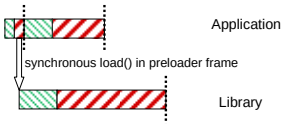
The resolution semantics are intended to allow both an "AS2-like" model (with better encapsulation) as well as partitioned definition spaces.

Dynamic linking

- **`--externs [symbol][…]`**
  List of definition names (classes, functions, etc.) to mark as external (omit from linking).
- **`--load-externs <filename>`**
  Load a file (link report!) containing a list of definition names to mark as external. Buggy in 2.0, requires 2.0.1.
- **`--external-library-path [path-element][…]`**
  Compile against libraries in this path, but add all definitions found to the list of externs.

Dynamic linking

`--runtime-shared-libraries [url][…]`



Specifies list of SWF files to load before the main application is initialized (generally in order to satisfy dependencies marked as external)

"Hard" vs. "Soft" dependencies

- The AVM verifier requires that hard type dependencies found in a definition signature must be satisfied before any Actionscript Bytecode ("ABC") block requiring them is visited. (This requires careful MovieClip timeline choreography!)
- Hard dependencies linked "extern" must generally satisfied early by loading RSLs.
- Hard type dependencies found only inside method bodies may be resolved late!
- A soft dependency can be formed using ApplicationDomain.
- Soft dependencies can be resolved by late-loading a SWF containing the definition.

```
// "Hard"
public var hi:Hello = new Hello();

class Yo extends Hello {…}

function x(hiya:Hello):void {…}
```

```
// "Soft"

var Hello:Class
  = Class(getDefinition("Hello"));
var howdy = new Hello();
```


Putting the SWF timeline to work

- **`--frame [label] [classname][…]`**
  Flex models each frame as consisting of a primary definition and its dependency graph. This configuration option allows you to specify the name of the primary definition for frames following the first (root class) frame. Buggy in 2.0, Requires 2.0.1.
- **`[Frame(factoryClass)]`**
  Flex AS3 metadata where a class specifies a requested factory class to be exported on the previous frame, implicitly constructing a backwards daisy-chain of frame definitions. *

*If found on a base class, it implies that an IFlexModuleFactory-compliant subclass of the specified factoryClass should be generated. If on a leaf class, it implies that the factory already exists. Yes, this is weird and overcomplex, but We Had Our Reasons At The Time™.

Partitioning Functionality

- Reduce dependencies
  - Interfaces
  - Mixins and registries
- Satisfy hard dependencies
  - Load a RSL
  - Load into a context where externs are satisfied by a parent ApplicationDomain
- Satisfy soft dependencies
  - Load a SWF
  - Provide a definition in a frame later in the same SWF

Runtime Shared Libraries: an example

- RSLs contain definitions loaded (into ApplicationDomain.currentDomain) by the preloader before the application is started.
- The **`external-library-path`** option is used to omit RSL definitions during linking.
- The SWF inside the RSL SWC must be extracted and deployed for runtime loading. The **`runtime-shared-libraries`** option is used to specify the URL where this SWF can be found.

The frameworks SWC needs to be stripped of debugging information before it is usable as a non-debug RSL. See my blog for more information.

Runtime Shared Libraries: performance analysis

- In general, the aggregate size of any single dynamically linked application plus the RSL it loads will be larger than if the application were statically linked.
- RSLs *may* improve amortized download performance when shared between multiple applications.
- Startup time for a dynamically linked application is potentially much longer than that of a statically linked application.
- Need to analyze your use case carefully!

Before:

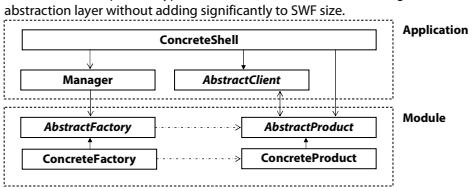
```
app1: 129,899
app2: 170,912
```

After:

```
app1: 9127
app2: 50,320
RSL: 140,627
```


A simple partitioning model: Shell and Modules

- Shell application loads external module SWFs
- Each module root class implements a class factory, with a standard interface
- Product of module class factory implements an interface known to the shell (and/or shell implements an interface known to the modules)
- Shared interfaces provide type-safe communication and enforce a good abstraction layer without adding significantly to SWF size.



mx.modules

- New package that implements the Shell/Module pattern!
- Available in 2.0.1 (pester Matt Chotin to get on private beta)
- Modules implement IFlexModuleFactory, compiled just like applications.
- Singleton ModuleManager maintains URL-to-IFlexModuleFactory map.
- Clients register for events to be notified of factory status.
- Used by new runtime stylesheet feature!

```
var m:IModuleInfo = ModuleManager.getModule( “assets.swf” );
m.addEventListener(
     ModuleEvent.READY,
     function(e:Event):void
     {
       var image:Bitmap = m.factory.create() as Bitmap;
       addChild(image);
     }
);
m.load();
```


Portal/Portlet

- Problem: portal that displays a configurable set of “portlet” mini-applications.
- Portlets may be developed by different remote teams, development efficiency is a major concern.
- More portlets exist than any one user will probably ever use.
- Multiple instances of the same portlet may simultaneously be running.
- Observation: no genuine need for hard dependencies from portal to portlet.

- Solution: Data file specifies list of portlets SWFs to load, each honors a common interface contract.
- Perfect example of Shell/Module pattern.
- Portal and portlets can be developed independently without full recompile as long as the interface API is preserved.
- Only portlets needed by a particular user will ever be loaded.

Huge application

- Problem: an application with hundreds or thousands of screens.
- Startup time and memory use is prohibitive when compiled into a single monolithic application.
- Key observation: any given run may only visit a few screens.

- Solution: bundle packages of related screens into individual SWFs.
- Fits Shell/Module pattern.
- Functionality can be loaded on demand, perhaps unloaded on a LRU basis.

Streamed assets

- Problem: an application with one or more very large (mandatory) assets.
- When assets are embedded in application, initialization time is significantly impeded.
- Observation: no real need to have assets mixed in with application code as long as they are guaranteed to eventually be available.

- Solution: export classes on frame(s) following application.
- Supported in mx.modules!
  - Frame class implements IFlexModuleFactory
  - Factory is “published” to ModuleManager when frame is fully streamed in
  - Clients register for factory notification events as usual
- Maintains single file deployment!

The fine print

- Deploying multi-SWF applications adds new failure paths.
- Embedded fonts don't work very well across SWF boundaries.
- ApplicationDomain.currentDomain is weirder than you might ever imagine.
- MXML framework has too many interdependencies to take significant advantage of modules, mainly of interest when writing large amounts of user code.
- Being unloadable requires care.
- Need to wait 'til 2.0.1 for official support!
- http://blogs.adobe.com/rgonzalez

Better by Adobe™