# DENSE AND HOT


## Michael Labriola

Digital Primates


# Who are you?

Michael Labriola

Senior Consultant at Digital Primates

Flex Geek

Component Developer

Flex Team Mentor

# Who were you?

Michael Labriola

Software Engineer

Embedded Systems Developer

Reverse Engineer

# What is this session about?

There are hundreds of milliseconds of stuff that happens before your Flex App ever shows up on the screen

This session is dedicated to those milliseconds

# Why am I still here?

Pure geeky goodness

There are practical implications to everything here, but this is *way* more information than you need

Sit back. Enjoy the show

# One more item

I am going to lie to you a lot… a whole lot

Even at this ridiculous level of detail, there is much more

All of this is conditional based on parameters such as creationPolicy. So, we are just going to take one route and go with it

# Background Info

Flex applications are Flash Movies

These movies play in the Flash Player

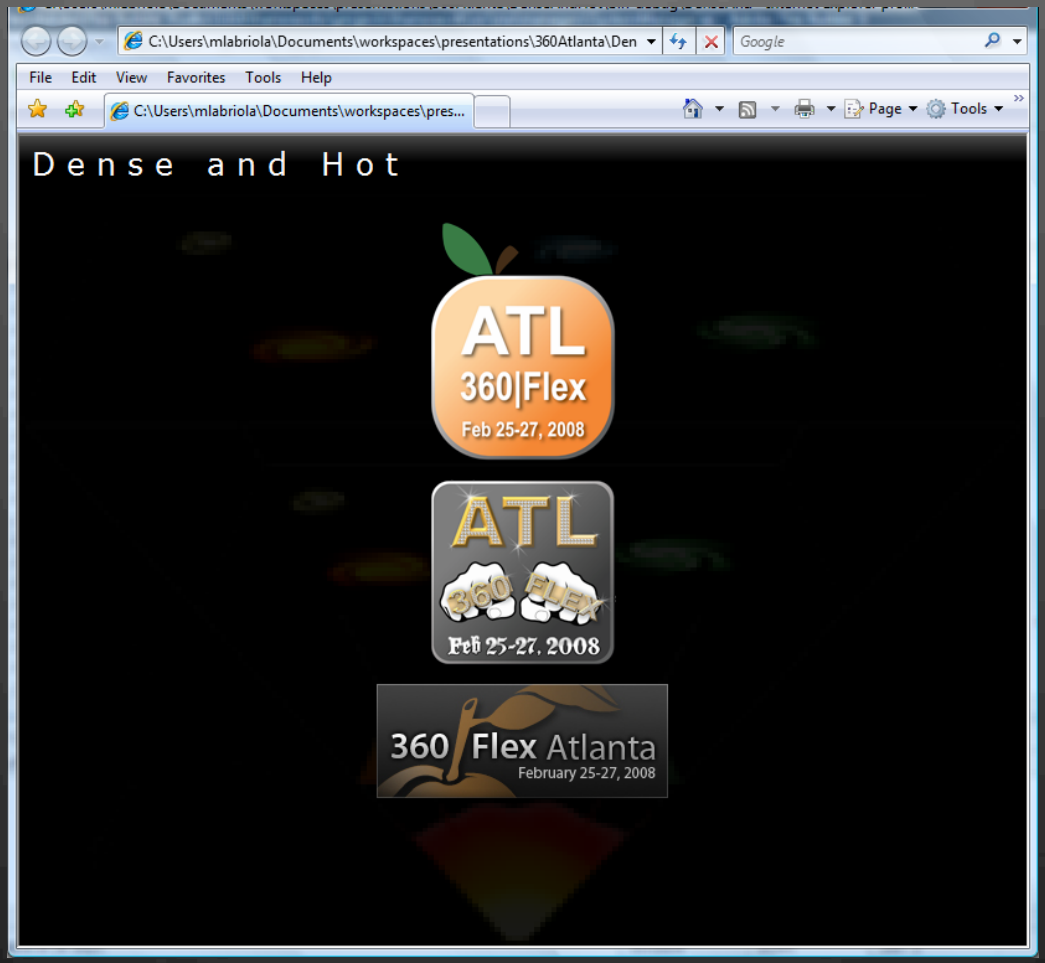
Like any movie they are composed of frames

# Movies

Flex Applications are movies with exactly two frames

Those of you from a Flash background understand this movie clip metaphor

# Appy Goodness




# Flash Movies Stream

The data in this stream arrives over time

As soon as all of the data arrives for a frame, flash player can act on it without waiting for the next frame to arrive

# Frame 1

The first frame of a Flex application movie contains only very critical classes including

- SystemManager
- Preloader
- DownloadProgressBar

# Frame 2

Frame 2 contains the remainder of your Flex Application code, fonts, images, whatever

This means it takes a lot longer to get all of frame 2 downloaded than the small stuff in frame 1

# Why?

The visible benefit of this whole idea is the loading bar.

Even on a big app, Flex quickly shows that it is loading (Frame1) and gives continual feedback to the user while the rest of the app loads (Frame 2)

# <Nitty-gritty>

Once frame 1 is downloaded, an instance of the SystemManager class is instantiated.

This is where it all begins

# System Manager

The system manager does all of the following things:

- It’s the parent of your <mx:Application> tag
- It’s the parent of all popups, tooltips, cursors…

# System Manager

- Informs the rest of your app if the browser resizes
- Any keyboard or mouse actions not handled elsewhere eventually find their way here
- Handles switching of focus between top level windows

# System Manager

The SystemManager is now responsible for setting everything else in motion to get your app on the screen

It all starts in the SystemManager constructor, where the next 5 slides happen

# The Constructor

The SystemManager now:

- Checks to see if it is the top level application or if it is being loaded into another app
- Tells the ResourceBundle class which locale we are going to display
- Tells the flash movie to stop playing

# Still Construct'n

The last thing the SystemManager does in its constructor is:

Add an event listener for the INIT event of the loaderInfo object to call the SystemManager's initHandler

# loaderInfo

The loaderInfo object

- Contains a bunch of readonly properties
- Knows things like the number of bytes loaded, bytes total, framerate of the movie, etc.
- It also broadcasts a series of events

# loaderInfo

Including:

- Complete – Broadcast once all data is loaded successfully
- Init – Broadcast once all properties and methods of a loaded object are available and the constructors for any child objects have complete
- Also broadcasts progress, httpStatus, ioEvents

# Init'n

Once the constructor of the SystemManager is finished executing, the INIT event of the loaderInfo object is broadcast

The SystemManager's initHandler method is now called

# initHandler

The startup continues here as the framework removes the event listener added in the SystemManager constructor (we don't want to be bothered again when the next frame is ready)

The framework also uses a nice mostly undocumented method called addFrameScript

# initHandler

The addFrameScript method is used to ensure a method is called whenever the play head of a flash movie enters a specific frame

In this case, the framework specifies a method named docFrameHandler to be called when we enter the next frame of our SWF

initHandler then calls initialize()

# Initialize

The initialize method creates a new Preloader and listens to both its progress and done event

The preloader is added to the SystemManager's children

If your application uses any type RSLs that information is gathered at this point

# Initialize

The ResourceManager, FontRegistry and StyleManagers are registered with the application so that they will be ready and available before they are needed in the next frame

The framework looks for any resource modules specified in the URL or flashvars that might also need to be loaded before frame 2

The preloader is initialized and sent on its way

# Preloader

The preloader initialize method facilitates loading an RSLs used by your application

It then creates an instance of its display class. By default, this is the DownloadProgressBar, however, you can specify a different class if you want to change the preloading display

At this point, the preloader starts a process of polling and updating the display until the RSLs and Frame 2 are loaded

# Wait for Frame 2

# Progress Handler

The preloader throws a progress event, which is handled in the preloader_initProgressHandler method.

The method checks the number of frames loaded and, if we are loaded, it advances the play head to the next frame

# docFrameHander

Earlier, we asked the player to call our docFrameHandler method when the play head moved to the next frame.

In this method Flex registers a bunch of additional application singletons, including items like the layout and tooltip manager

It then starts the process of initializing mixins on the System Manager

# mixins

Mixins are, in general, a way to add dynamic functionality to a class

Good old Steve and his ice cream

Each of the mixins is initialized and added to the SystemManager

# Resource Bundles

Soon, we are going to need to start moving toward actually making some components appear on the screen. But first, we need to know what language will be displayed.

So, the downloaded resource bundles are now installed and made ready to use.

# The UI

We now start the process of instantiating the UI.

- Flex creates an instance of your application class
- It listens to the creationComplete event of your app, calling appCreationCompleteHandler when it occurs
- The initial size of the main application is specified as the size of the stage

# Not Yet

Even though an instance of your application exists at this point, it has not been added to the system manager and hence has no where to appear

To prevent laying out the screen multiple times, Flex waits for the initialization to complete before handling this step

To facilitate this, the new app is registered with the preloader

# Back to the Preloader

Registering the new application instance with the preloader, just causes the preloader to listen to another series of events and respond later.

These events include:

- validatePropertiesComplete
- validateSizeComplete
- validateDisplayListComplete
- creationComplete

# Back to the Preloader

Registering the new application instance with the preloader, just causes the preloader to listen to another series of events and respond later.

These events include:

- validatePropertiesComplete
- validateSizeComplete
- validateDisplayListComplete
- creationComplete

# Still Preload'n

Each time one of these events occur, the preloader broadcasts an INIT_PROGRESS event

This event is used back in the SystemManager to wait for appropriate progress before continuing

# Back to SystemManager

The system manager starts the process of adding the new child application

It set a property on the application, letting it know that this particular system manager is in charge

It also sets other properties such as the document, nestLevel before beginning the process of building up all style information for the application

# Back to SystemManager

The system manager forces the new application to broadcast an Add Event and then begins the application's initialization process

It does this by calling the initialize() method of the Application subclass

# Application

The application initialize method sets up the context menus and then gathers information to create its own children from descriptors

Descriptors are instances of the ComponentDescriptor class. One of these is created for each of your MXML tags and allows actionscript to read those properties

# PreInitialize

At this moment, the preInitialize event is broadcast

This event occurs after everything about a component (in this case the actual application) is initialized but before any of the component's children are actually created

With the exception of the application, this event is broadcast after the component is 'attached' to its parent

# More - PreInitialize

PreInitialize is a great time to change properties or setup data that affects how components will be created

As it is broadcast before children are created, the preinitialize of a parent will always be broadcast before the preinitialze of a child

# createChildren()

At this time, the application's createChildren() method is called.

For those of you that do component development, createChildren() is the method you override in your own components to ensure that your child display objects are created

# More createChildren

The application's createChildren method creates:

- Any borders for the application
- Any needed scrollbars

It then calls a method called createComponentsFromDescriptors() which in turn calls a method named createComponentFromDescriptor()

# Recurse Away

Assuming that we haven't messed with the creationPolicy of anything in our app, Flex now creates a child from each of its descriptors.

After it creates each child, it adds it to its parent (the application in this case)

Much like what happened for the application, the moment this child is added, its initialize() method is called, which in turn calls its createChildren

# LayoutManager

One of the last thing each child container does it to listen to an Event of the LayoutManager (one of those classes we instantiated much earlier when we entered this frame) for an UPDATE_COMPLETE event

This becomes important later

# Finishing up Creation

After the application returns from recursively creating all applicable children, it calls the childrenCreated method

The childrenCreated method calls each of the invalidate methods

- invalidateProperties();
- invalidateSize();
- invalidateDisplayList();

# Invalidation

Each of these invalidate methods asks the flex framework to do something at the next available interval

- invalidateProperties() asks for the commitProperties() method to be called
- invalidateSize() asks for the measure() method to be called
- invalidateDisplayList() asks for the updateDisplayList() method to be called

These will be called when the player has time to do so, not necessarily immediately

# Invalidation

Each of these invalidate methods asks the flex framework to do something at the next available interval

- invalidateProperties() asks for the commitProperties() method to be called
- invalidateSize() asks for the measure() method to be called
- invalidateDisplayList() asks for the updateDisplayList() method to be called

These will be called when the player has time to do so, not necessarily immediately

# Bindings

The framework now executes all bindings on the new child, meaning if you bound the x and y position of your new child to some initial variable, this value will now be set

# Initialize Event

At this point, the initialize event for any child created inside of the application is broadcast

The children are all created, but have not yet been sized or positioned in any way.

# Rendering and Entering

Every time the flash player enters into a new frame, or accesses the same frame (which happens at the frame rate), Flex broadcasts a Event.ENTER_FRAME event

This event is used to call code inside the UIComponent class that communicates with the LayoutManager to check if a component needs to be measured, redrawn or have new values committed

# More Invalidating

This LayoutManager method is actually looking at the flags set by the invalidateSize(), invalidateDisplayList() and invalidateProperties() methods of the component which were called when we created them from their descriptors

The first time through the process we have not yet called any of these methods, so the LayoutManager calls the validateProperties(), validateSize() and then the validateDisplayList()

# Which..

Each of these method does a fair amount of checking for optimization, but eventually results in calling the components commitProperties(), measure() and updateDisplayList() method

# Initialized

After each of these method calls has completed, the LayoutManager verifies that there are no further children to create and this time and then sets the initialized property of the component

The initialized property changes the components visibility and then finally broadcasts the creationComplete event on that component

# Bringing it Full Circle

Many slides ago, we discussed the preloader. One of the things the preloader did was to listen to the creationComplete event of the application.

One this creationComplete event fires, that code from the preloader is executed

# Back to the SystemManager

Once the event is received, the preloader broadcasts an event indicating that its job is done.

The SystemManager who has been listening to that event since the beginning of this process, removes the preloader and adds the newly created application to the SystemManager, making it visible for the first time

# Application Complete

The Application Complete event is fired from both the Application and the System Manager.

The app is visible and startup is complete

# Q&A

Seriously? You must have some questions by now?

# Resources

Blog Aggregator (All of the Digital Primates)
http://blogs.digitalprimates.net/

My Blog Specifically
http://blogs.digitalprimates.net/codeSlinger/


ATL
360 FLEX