MATE: A Flex Framework

"Extreme Makeover"

Our Agenda

- Introductions
- What Is MATE?
- Why Use MATE?
- Take A Look At The Original App – ClipSafe
- It's Time for an "Extreme Makeover!"
- Extending MATE
- Summary

Introductions

Me: Theo Rushin, Jr. (rushint@yahoo.com)

Senior Web Application Developer/IT Trainer

@ NicheClick Media

- ✓ Application Developer and Trainer Since '85
- ✓ ColdFusion Developer Since '99
- ✓ Flash Developer Since '99
- ✓ Flex Developer Since '05
- ✓ Deploying Flex application to AIR Since '07

What Is MATE?

- MATE is a tag based, event driven framework for Flex development.
- It is pronounced _ **"mah-teh"**
- It helps you build applications that promote loosely coupled components.
- There is very good documentation, examples, diagrams, and presentations to help you get started and learn more.

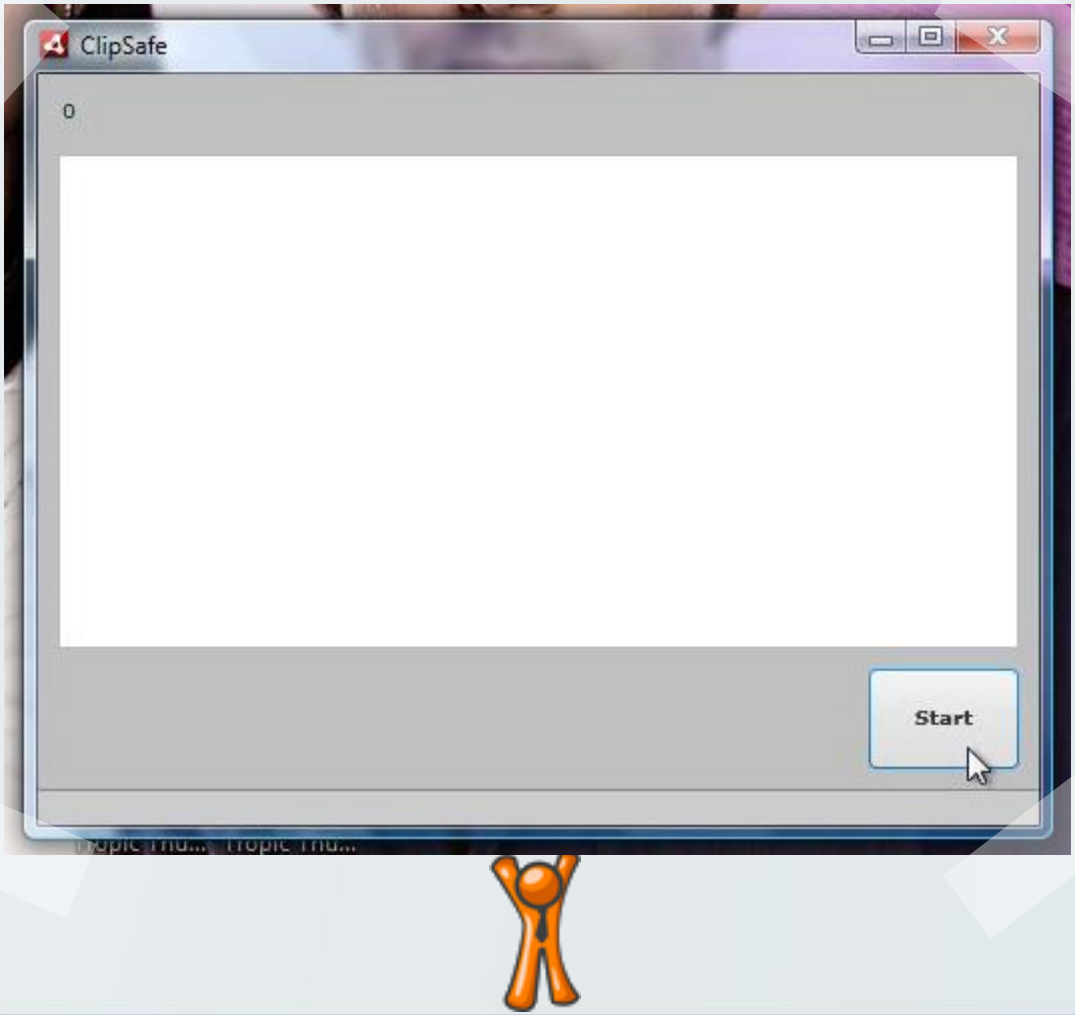
Why Use MATE?

- MATE helps you create applications that are highly decoupled.
- MATE helps you organize and encapsulate the code and the various components.
- A very easy framework to understand with plenty of documentation and examples.
- Communication throughout an application built using MATE use standard or custom Flash event.
- Tag-based NOT Actionscript-based.
- Very easy to extend.
- Your application is loosely tied to the MATE framework

Take a look at the original app

- ClipSafe!
- It's a simple Flex/AIR application.
- Take a snapshot of what's on your desktop (Prnt-Scrn).
- ClipSafe saves the snapshot into a folder on your PC.
- Application architecture? One MXML file containing ALL code.
- Not very scalable, extensible, nor reusable - but it works.

Take a look at the original app (cont.)



Time for an "Extreme Makeover!"

EXTREME

MAKEOVER

mate EDITION

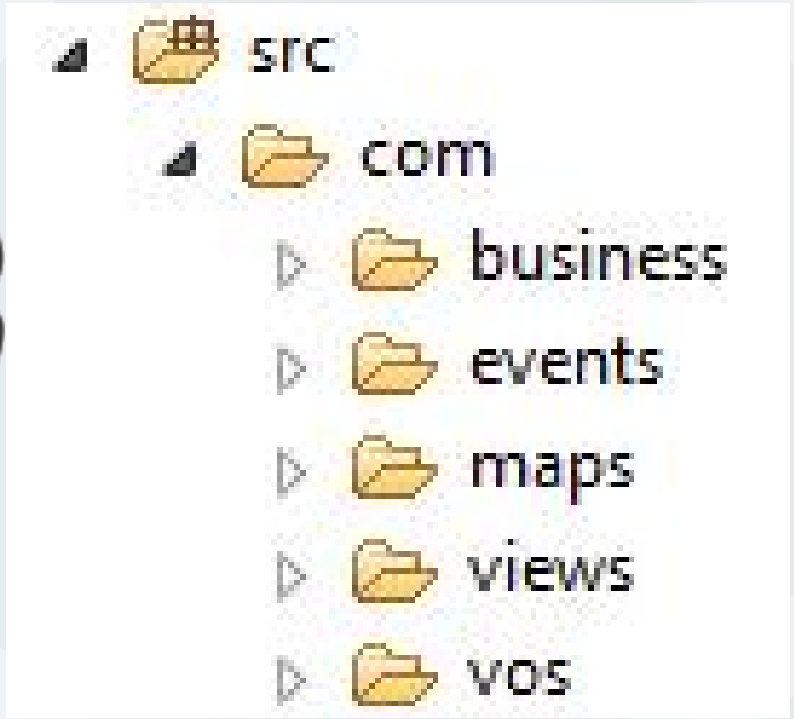
Time for an "Extreme Makeover!"

Setting up Your Environment

- Not unlike many other frameworks, the MATE framework enables the developer to build loosely coupled applications. Separating the application's views from the service layer and from the business logic results in an application that is scalable, reusable, and easier to understand.
- Even a small application that implements the MATE framework will consist of various smaller custom classes.
- Create the folders that will help define the structure of your application.

Time for an "Extreme Makeover!"

Setting up Your Environment (cont.)



Time for an "Extreme Makeover!"

Adding the MATE library

- ASFusion currently makes available two compiled libraries for download (as of this writing)
  - Version 0.7.7 for Flex 3
  - Version 0.7.7 for Flex 2
- You can also download and browse the source via SVN

http://mate-framework.googlecode.com/svn/trunk/src

Time for an "Extreme Makeover!"

Adding the MATE library (cont.)

- Add the compiled MATE (swc) to your project. For smaller applications this can be placed into your lib folder.
- From the main menu select **Project > Properties**.
- Select **Flex Build Path** from the left-hand menu in the Properties dialog box.
- Select the **Library Path** tab.
- Click on the **Add SWC** button.
- Browse to the folder containing the .swc file (libs?)
- Select the .swc file (MATE_?_?.swc) and click on the OK button (twice)

Time for an "Extreme Makeover!"

Create a Config Class

- It not necessary for you to create a **Config** class in order to build an application using MAT E.
- I do so only for this demo app to demonstrate the passing of values into the **ClipSafeManager** class (more on this class later).
- It's a simple custom class defining two constants and their values.

Time for an "Extreme Makeover!"

Converting the View

- Strip out the view and place it into it's own custom component.
- Place the custom view component into the views folder.
- Define a script block that will import the required classes, declare public variables, and define functions to support the view.

Time for an "Extreme Makeover!"

Dispatching Events from the View _ using dispatchEvent()

This is the standard way of dispatching a standard and custom events.

NOTE:

- The dispatched event must have its bubbling setting as "true"

AND

- The event must be dispatched from a component that has the Application as its root component.

Time for an "Extreme Makeover!"

Dispatching Events from the View _ using <Dispatcher ..>

This tag can be used to dispatch an event from anywhere in the application. Using this tag guarantees that the event will be received by ALL registered listeners.

- Use the **generator** attribute to specify the event class that should be dispatched.
- Use the **type** attribute to specify the type of event.

Time for an "Extreme Makeover!"

Defining the Event Map

- MATE is an event-based framework that revolves around what is known as an Event Map. The Event Map file is an MXML component extended from the EventMap class.
- An event map defines, among other things, one or more event handlers for the application. The <Event Handlers ...> tag is used to listen to events dispatched by the application.

Time for an "Extreme Makeover!"

Defining Event Handlers

- The <EventHandlers ...> tag is used to define one or more handlers that are executed whenever a specified event type occurs.

The event type can be either a standard Flex/Flash event

OR

The event type can be a custom event.

Time for an "Extreme Makeover!"

Create the Custom Event Classes

- You create the custom events that same way you always have (have you?)
- **ClipSafeEvent** is the event that will be triggered when the Start/Stop button is clicked from the view.
- **CapturePixEvent** is the event that will be triggered when the application detects that a an image has been captured.

see code

Time for an "Extreme Makeover!"

Create the Value Object (vo) class

- This class will be used in the CapturePixEvent class allowing us to pass the snapshot information along with the event.

Time for an "Extreme Makeover!"

Using the ObjectBuilder tag

- The **<ObjectBuilder ...>** tag is used to create an object instance. The type of object is specified in the **generator** attribute of this tag. Arguments can be passed to the constructor of the class by using the **constructorArguments** attribute of this class. This attribute expects an array of values.
- You can specify whether the object will be cached or not by using the **cache** attribute. By specifying "true" as the value of this attribute the object instance will not be instantiated when using the **MethodInvoker** or **PropertyInjector** tags. The default value is "true"

Time for an "Extreme Makeover!"

Creating a an Application Manager - ClipSafeManager

see code

- This is the business layer of the application.
- Your application may have one or more of these type of classes.
- This class will contain methods that are called from the **MainEventMap** component.

Time for an "Extreme Makeover!"

Using the MethodInvoker tag

- The **<Method Invoker ...>** tag is used to create an object instance of the class specified in the **generator** attribute (unless cache is set to "true" in a previous ObjectBuilder tag). Then it will execute the method specified in the method attribute.
- Arguments can be passed via the **arguments** attribute. The arguments attribute expects an array of values.

Time for an "Extreme Makeover!"

Using PropertyInjectors

- These tags are also defined in the **MainEventMap** component.
- The **PropertyInjector** tag is placed within the **Injectors** tag block.
- They are used to inject property values from a specified source into a specified target.
- The target class is specified in the **target** attribute of the **Injectors** tag.
- The target property, source class, and source, property are specified in the **PropertyInjector** tag.

Extending MATE

- It is very easy to create your own extensions to the MATE framework.
- MATE extensions are custom classes that extend the **AbstractServiceInvoker** class and implement the **IAction** class.
- Extenting from the **AbstractServiceInvoker** class will give your custom extension access to inner result and fault handlers.
- Implementing the **IAction** class allows you to use your custom extension tag inside the **EventHandlers** block.

Extending MATE (cont.)

- Your custom extension needs to override the run method and script it to do what you want it to do when.
- You need to specify what event will trigger your resultHandlers execution and who is dispatching that event.
- Use the **createInnerHandlers** method defined by the **AbstractServiceInvoker** class to define your result and fault handlers.

Summary

Download - Learn - Participate!

ASFusion Web site: http://mate.asfusion.com/


"I Have a Question!!"
Q&A