Flash Camouflage

By Jesse Freeman

Who is Jesse Freeman?

Flash Bum

Homeless Web Developer

10 Years of experience

Worked for
Heavy.com,
Major League Baseball,
The New York Jets,
AKQA, HBO, FOX,
Arista Records and BBDO

Specialize in Flash workflow,
large scale xml driven Flash Apps
and building Flash Frameworks

Currently I am the Lead Developer at

RadicalMedia.com

THE FLASH ART OF WAR

The Oldest Flash Military Treatise in the World

Home About Contact Search

Flash Camouflage Framework August 30th, 2008

Flash Camouflage

I am happy to introduce my new Framework called Flash Camouflage. I have just posted a stable build of it on google code and will be continuing to add and refine it over the next few months. Here is a little background into what the framework can do:

Flash Camo (for short) is a display layer framework that allows AS 3 applications to be easily skinned from pngs, jpg, or gifs. Camo also supports CSS styles, a custom BoxModel, and layouts defined by XML. Camo aims to be a foundation for building themed/skinned display layers for your application. You supply the logic that controls it. Camo also allows live streaming of application's skins so you only display and load what you need when you need it. This allows you to dramatically cut down on the file size of your Flash App without sacrificing detail or design.

So what does this all mean? Here is what the basic core Camo Framework will give you:

Inspiration

" Fighting with a large army under your command is nowise different from fighting with a small one: it is merely a question of instituting signs and signals. "

by Sun Tzu, the Art of War

360/Flex Camp

FDT Roadshow

FlashArtOfWar.com

Flash Camouflage

Features

- Display Layer Framework
- OOP and Design Pattern Foundation
- Custom CSS Parser
- BoxModel inherited in the Display Classes
- Asset Library
- 100% XML Configurable
- Light Weight - less then 60k
- Completely Flexible

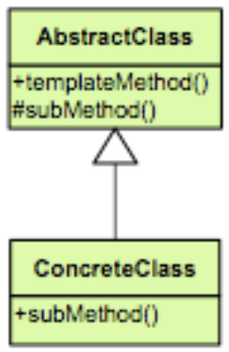
Design Patterns + OOP

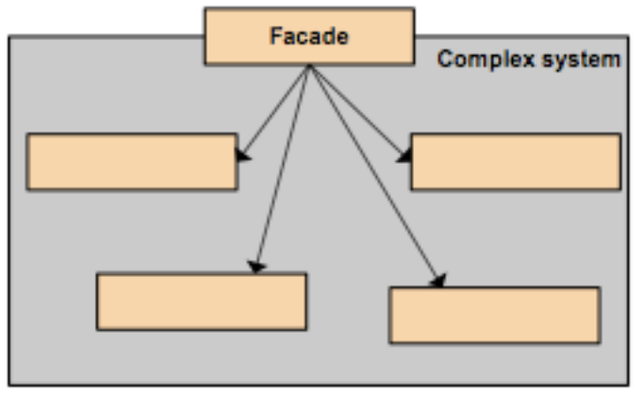
Design Patterns

Template Method

Type: Behavioral

What it is:
Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.





Facade

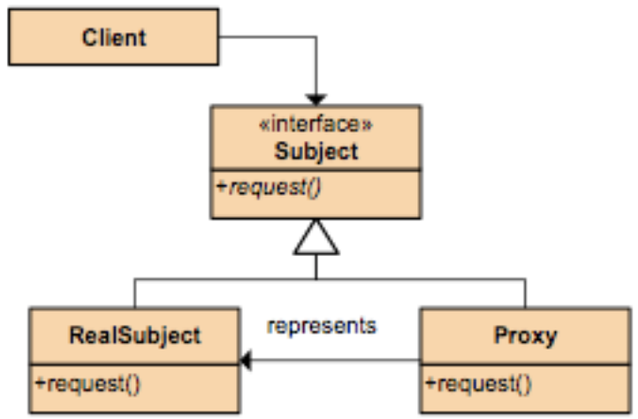
Type: Structural

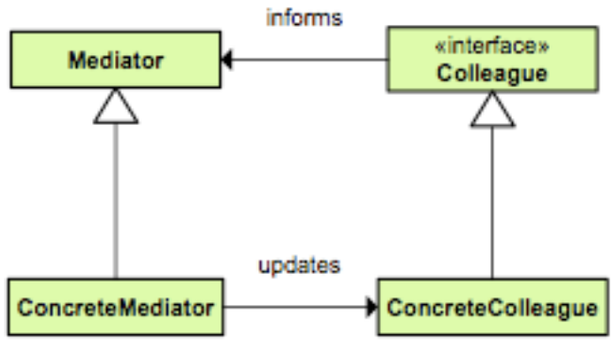
What it is:
Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.

Proxy

Type: Structural

What it is:
Provide a surrogate or placeholder for another object to control access to it.





Mediator

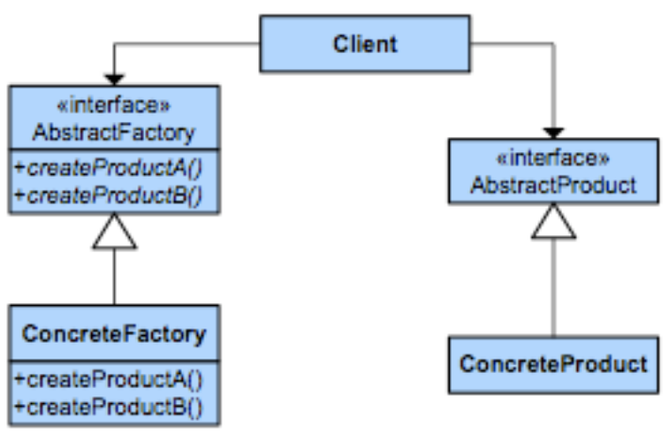
Type: Behavioral

What it is:
Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.

Abstract Factory

Type: Creational

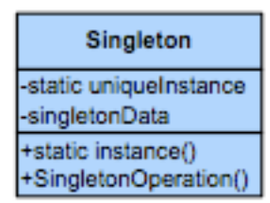
What it is:
Provides an interface for creating families of related or dependent objects without specifying their concrete class.



Singleton

Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.



Instantiating a class the traditional way

```
var tempClass:TempClass = new TempClass();
        tempClass.name = “Hello World”;
        tempClass.display = true;
        tempClass.delay = 5;
```

or

```
var tempClass:TempClass = new TempClass(“Hello World”, true, 5);
```

Using an Object To Configure a Class

```
var config:Object = new Object();
    config.name = “Hello World”;
    config.display = true;
    config.delay = 5;

var tempClass:TempClass = new TempClass(config);
```

Why use an Object to configure a Class?

- Encapsulate the data that goes into your class instance.
- Allow the class to handle its own construction by using the values set in the config class.
- Keeps class constructions clean and easy to follow
- Allows construction state memory.

Constructor Example

```
public class ExampleClass extends Sprite {

        private var _config:Object;

        public function ExampleClass(config:Object){
            // Do pre init logic here
            init (config);
        }

        protected function init(config:Object):void{
            // Read Config data here and set up class
            _config = config;
        }
}
```

Using a Class To Configure Another Class

```
var config:TempClassConfig = new TempClassConfig();
    config.name = “Hello World”;
    config.display = true;
    config.delay = 5;

var tempClass:TempClass = new TempClass(config);
```

Config Class Example

```
public class ConfigClass extends IConfig {
    public var text:String = “Hello World”;
    public var display:Boolean = true;
    public var delay:Number = 5;

    public function ConfigClass(){
        //Empty Constructor
    }
}
```

Why use one Class to configure another Class?

- You get all the same benefits outlined when using an object.

- Config Classes can validate data through custom getters and setters, can setup default values, and correct type issues at runtime or when compiled.

- ConfigClasses help teams of developers keep a common standard for setting up class instances and help promote cleaner more OOP code.

- When properly typed to an interface or Abstract Class, ConfigClasses can be extended and substituted on the fly which is the building block of polymorphism.

Polymorphism

"The concept of polymorphism implies having certain functionality appear in multiple forms throughout your application, but having a unified way to call it. Doing this allows different classes throughout the project to be easily interchangeable."

http://www.springerlink.com/content/j5344166213008911/

XMLConfig?

<asset id="flash_half" class="AssetDisplay" assets="default:flash_half"/>

Yes!

```
var xmlConfig:XMLConfig = new XMLConfig();
    xmlConfig.id = “flash_half”;
    xmlConfig.class = “AssetDisplay”;
    xmlConfig.assets = “default:flash_half”;
```

camo.models.XMLProxyModel

XMLProxyConfig

- Pass in XML data and it is converted into a ConfigClass.
- Takes an XML's attributes and makes them available as properties of the XMLProxyConfig intsance.
- Is interchangeable with any Config class that implements IConfig interface so you can use a ConfigClass or XMLProxyConfig to configure classes in the framework.
- This is the key building block that allows the framework to be dynamically coded or configured from XML.

Camo Library

camo.models.libraries.CamoLibrary

Camo Library

- Manages CSS, Assets, and Components (Layouts)
- Is defined in XML and loaded at run time
- Is a Singleton so it can be called at any time, in any place of your application.
- Handles loading of external resources automatically such as Source Images for assets, Fonts (embedded in swfs) and external SWFs

Camo Library XML

```
<library>
    <styleSheet>
    </styleSheet>
    <swfs baseURL="/swf/">
    </swfs>
    <decalsheets>
        <sheets baseURL="/images/">
        </sheets>
        <assets>
        </assets>
    </decalsheets>
</library>
```

Using the Camo Lib

- Use IDs to retrieve items from the library
- Get access to the library by calling CamoLibrary.instance
- Camo Library supports getAsset(), getStyle(), and getLayout()
- Simply pass in the ID of the item you want and the Library will return the requested instance to you
- CamoLibrary.instance.getAsset(“flash_half”);

CSS Support

camo.models.libraries.CSSLibrary

The CSS Parser

- CSS is loaded into the stylesheet node of the library.xml
- Camo's CSS parser uses RegEx to find all of the selectors and indexes them.
- CSS is not parsed until it is requested, after the first request the parsed CSS object is cached for quicker retrieval.
- CSS is turned into a generic Object that mirrors the requested CSS.
- Cascading is also supported by separating multiple selectors with a space. Example: Style1 Style2 { ... }

Raw CSS?

```
text_demo {
    width: 600;
    embedFonts: false;
    align: center;
}
```

Returned CSS Object!

{styleID: “text_demo”, width:600, embedFonts:false, align:”center”}

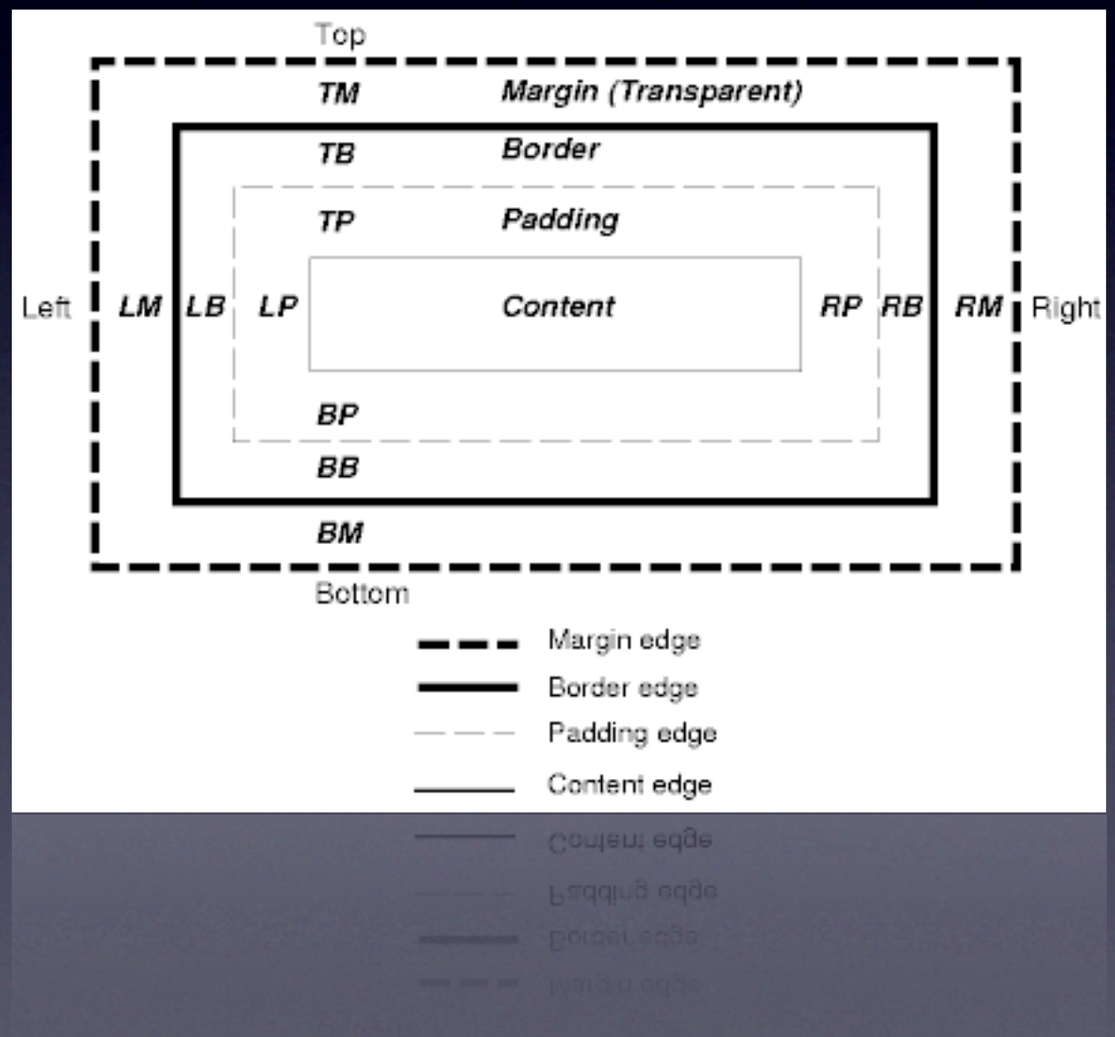
BoxModel

camo.views.display.BoxModelDisplay

BoxModel

Each Display Object in the framework extends the box model

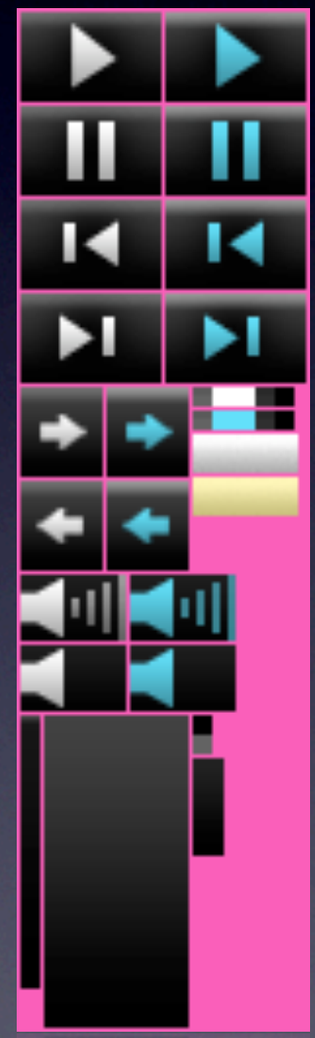
It supports
Margin, Padding,
Background (Color, Tiled ,
9 Sliced images) and
Border



Assets

Asset Sheets

- Asset sheets are used as the source for each asset.
- The sheet is one large image that can be broken down into smaller images on the fly based on asset coordinates defined in the library.xml
- Because the Asset Sheet source is loaded in at run time, “skins” for components can be changed and updated on the fly by simply loading in a new Asset Sheet.



Asset XML

```
<sheets baseURL="/images/flash/">
    <page id="controls">sheet.png</sheet>
</sheets>
<assets>
    <asset id="back_up" sheet="controls" x="1" y="49" w="36" h="23"/>
    <asset id="back_over" sheet="controls" x="38" y="49" w="36" h="23"/>
    <asset id="next_up" sheet="controls" x="1" y="73" w="36" h="23"/>
    <asset id="next_over" sheet="controls" x="38" y="73" w="36" h="23"/>
</assets>
```

Blitting

Bit blit (bitblt, blitting etc.) is a computer graphics operation in which several bitmap patterns are combined into one using a "raster operator".

http://en.wikipedia.org/wiki/Blitting

Why Use Blitting?

- Incredibly Fast
- Can perform compositing (such as adding masks, resize, and alter color) then cache as a rasterized image
- Uses less memory then Vectors, Sprites, and MovieClips
- Faster Scrolling, Displaying, and Updating
- Similar to what happens when you use CacheAsBitmap
- Flash Player 10 Hardware accelerated!!!!

Real World Example

CBS Burly Sports Video Player

How To Get Started

Download the source from

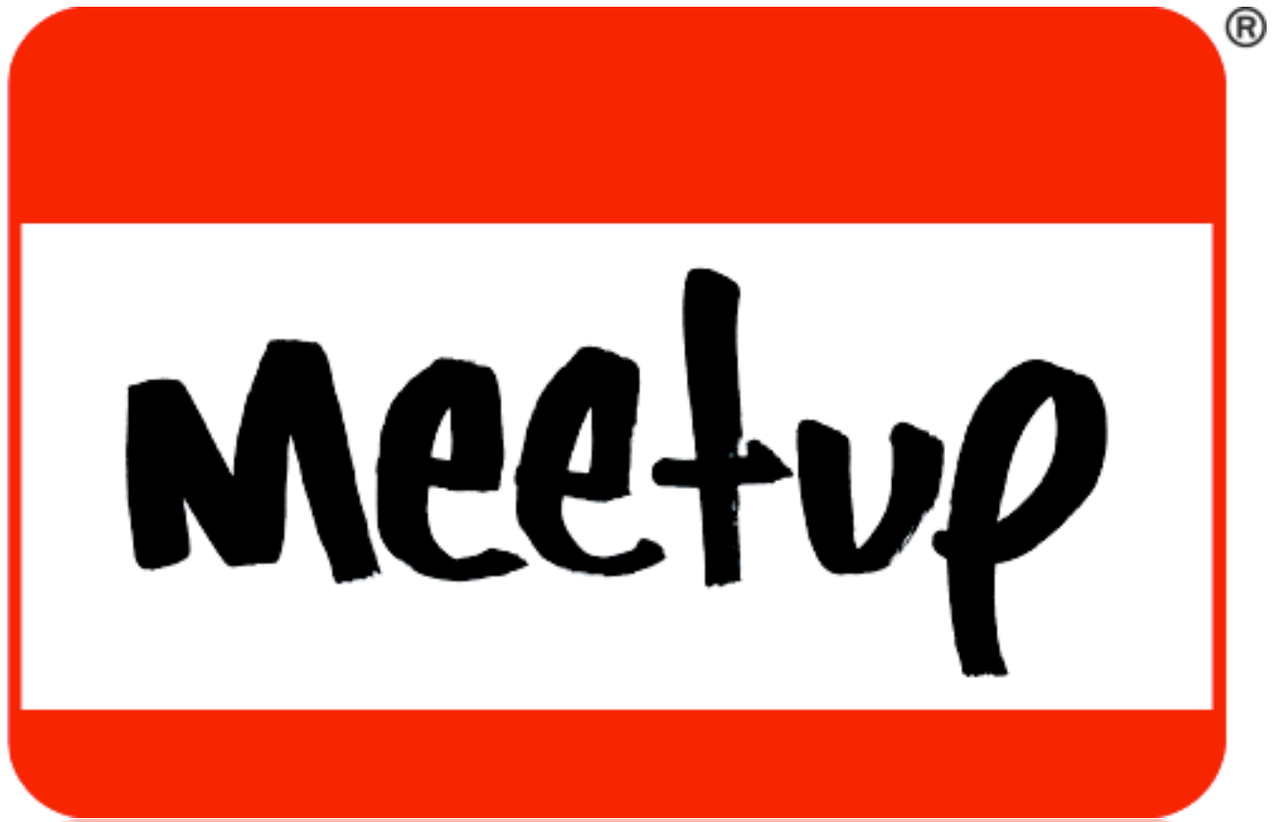
http://code.google.com/p/flash-camouflage/

Checkout from SVN

http://flash-camouflage.googlecode.com/svn

Downloads

- A SWC is available instead of using the svn checkout
- ASDoc documentation
- HelloWorld showing how to extend the framework, set up a project, and use Ant to build your project.



AS Programming Pushes Us Flash Meetup

http://flash.meetup.com/148/

Thank You For Watching