FoxR

PureMVC ActionScript 3 Framework

User Guide

Version 0.2.1

Last Updated 12/9/2009

Outline

I. Basic Info

Requirements

- Knowledge or proficiency with ActionScript 3 programming
- Flash Compiler
    - Flex SDK (version 4 recommended for all users)
    - Eclipse w Flex Builder Plug-in
    - Flex Builder 2 or 3 OR Flash Builder 4
    - HaXe compiler (www.haxe.org)
- Knowledge or proficiency with MVC design and programming and the PureMVC Framework. (STRONGLY RECOMMENDED)
- ActionScript IDE (RECOMMENDED)
    - Flash Develop (Win)
    - Eclipse with Flex Plug-in OR Flex/Flash Builder (Mac/Unix)
    - Flash CS3 or higher

License Agreement

Change Log

See http://foxr.aeoliandigital.com/releases/change-log for a complete list of changes to the library.

Credits

Jeff Fox – Founder, Lead Developer

Special Thanks To:

- Richard Kelly – Beta testing and Bug Checking
- Jordan Wilson – Architectural Inspiration, Adapter Workflow
- Todd Lynch – Map Plotting Code
- Thaylin Burns – Log inspiration, keeping me grounded

II. Installation

Downloading FoxR

Releases

0.1	Pre-Alpha Release – July 17, 2009
0.2	Feature Release – September 24, 2009
0.2.1	Maintenance Release – October 21, 2009

Subversion

FoxR is hosted with Google Code at: http://code.google.com/p/foxr/

Subversion access is available for non-members for download at: svn checkout http://foxr.googlecode.com/svn/trunk/ (Read–only)

Installation Instructions

FoxR is installed in four steps:

- Download and unzip the library package in a new directory
- Download the recommended Project Template zip file (http://foxr.aeoliandigital.com/releases#projectTemplate) and unzip into the same directory
- In the “New FoxR Project directory”, open flash/com/foxr/data/GlobalConfig.as file.
- Set the default properties for the movie. You can override base classes and set custom application settings here.
- Run the movie
    - Using FlashDevelop 3 or higher: Open **MyProject.as3proj**, and then go to **project > Test** Movie.
    - Using Eclipse orFlash/Flex Builder: Create a new project, set FoxR.as as the default executable file and ccompile and run FoxR.as as a Flash project.
    - Using Flash CS3 or higher, open src/Foxr.fla and publish.
    - Regardless of the IDE used, the compiled swf can be found in the deploy/ folder of the New Project directory.

If you're new to FoxR, read the Getting Started pages to begin learning how to build your own custom FoxR applications. And please, let us know if you’re using FoxR in your personal or commercial applications.

Upgrading

Instructions on upgrading from pervious version will go here.

Troubleshooting

Troubleshooting tips will go here.

III. Getting Started

Getting Started with FoxR

If you're new to FoxR, welcome! We hope you find FoxR to be an intuitive and powerful addition to your ActionScript Development Toolbox.

As FoxR is based on the **PureMVC framework**, we recommend that your first visit the documentation pages of the PureMVC site (http://puremvc.org/content/view/98/189/) to better familiarize yourself with the general MVC architecture that the FoxR framework is based upon.

Your first steps to begin using FoxR are to first download the latest release, install it on your computer, then read through the **Introduction** section to familiarize yourself with the basics of the framework. We highly recommend downloading the Default Project Template package as a means to speed the setup of your first project and get you right to coding.

Next browse through the **General Topics** section as it contains many high level and simple explanations of the FoxR frameworks design and mission.

The Architecture Diagrams provide detailed information on the inner workings of the framework and how it works.

The FoxR API documentation is available for browsing at http://foxr.aeoliandigital.com/asdocs/.

FoxR At A Glance

FoxR is both an Application and a Site Building Framework

FoxR is built to be flexible enough for use in developing and building RIA (Rich Internet Applications) as well as graphical heavy rich web sites built in Flash.

FoxR provides a consistent visual structure

Tired of dealing with numerous approaches and philosophies for designing and implementing visual architectures within your Flash movies? They're a thing of the past with FoxR. FoxR provides a clean, simple and consistent visual architecture with which to organize and manage the visual elements in your movies.

And if you need to override FoxR's default view objects such as the header, footer or background, you can do so just by setting a new class name in the main config file!

FoxR allows you to choose how to build your movie

Need to utilize Library symbols in your movie? No problem. You can easily utilize symbols and integrate them right into your FoxR workflow. Need to work in Flex with SWC components. No problem. FoxR makes your life easier to do the repetitive, basic stuff. You can define how you'll do everything else.

FoxR is available free of charge

FoxR is licensed under the MIT open source license so are free to utilize it however you like. For more information regarding the license, see the FoxR license agreement or visit the free software foundation Web site. http://www.fsf.org/

FoxR Runs on ActionScript 3

FoxR is written in pure ActionScript 3 to take advantage of the best features of this evolving language.

FoxR makes Fast, lightweight SWFs

Especially when using the standard Flash or Flex component libraries. Since all FoxR components are drawn using the ActionScript drawing API, they're naturally faster and lighter for incomparable performance.

FoxR Uses M-V-C and is based on PureMVC

FoxR implements the standard Model-View-Controller design pattern, specifically implementing the PureMVC framework as its base design paradigm. This provides a consistent separation between your core application logic and the visual presentation.

FoxR's MVC design and PureMVC implementation are documented on the PureMVC implementation overview.

FoxR uses CSS

And we're not talking about CSS just for text or form components like in Flex. We mean CSS for *EVERYTHING*. If a FoxR visual element has a setter method, you can use CSS to define CSS classes than be be applied directly to an object by means of the built in CSSProxy. This powerful proxy also allows you to fully cascade styles, allowing successive stylesheets to override the default properties of previously loaded files.

See the Styling with CSS section of General Topics for more information.

FoxR is i18n ready

Need to support multiple languages? Good luck doing so with conventional Flash movie design when text is placed within symbols in your FLA!

FoxR includes a custom Copy Proxy that allows you to load one or more XML based copy files for use in your movies. XML based copy means quicker turnaround and lower costs with translation house and better for your project!

FoxR is ready to use

Download and install the companion starter project template and your ready to get cracking. Even without changing any of the configuration options and global XML text and CSS values, you're ready to roll!

FoxR Does Not Require Flash Cs3, CS4 or Flex

FoxR DOES NOT require you to own Flash CS3, CS4 or Flex Builder to build and compile movies using it. Using just Notepad and the free Flex SDK, you can author, compile and test FoxR movies anywhere, anytime.

More information on the Flex SDK is available at:
http://opensource.adobe.com/wiki/display/flexsdk/Flex+SDK

FoxR has lots of documentation

We know that everyone loves to write code and make stuff work quickly and that documentation is a drag. Some of the best open source tools out there lag or founder due to a lack of documentation. Not FoxR. Not only is there a user guide, but FoxR code is clean, organized, well commented and has a complete ASDoc API companion guide!

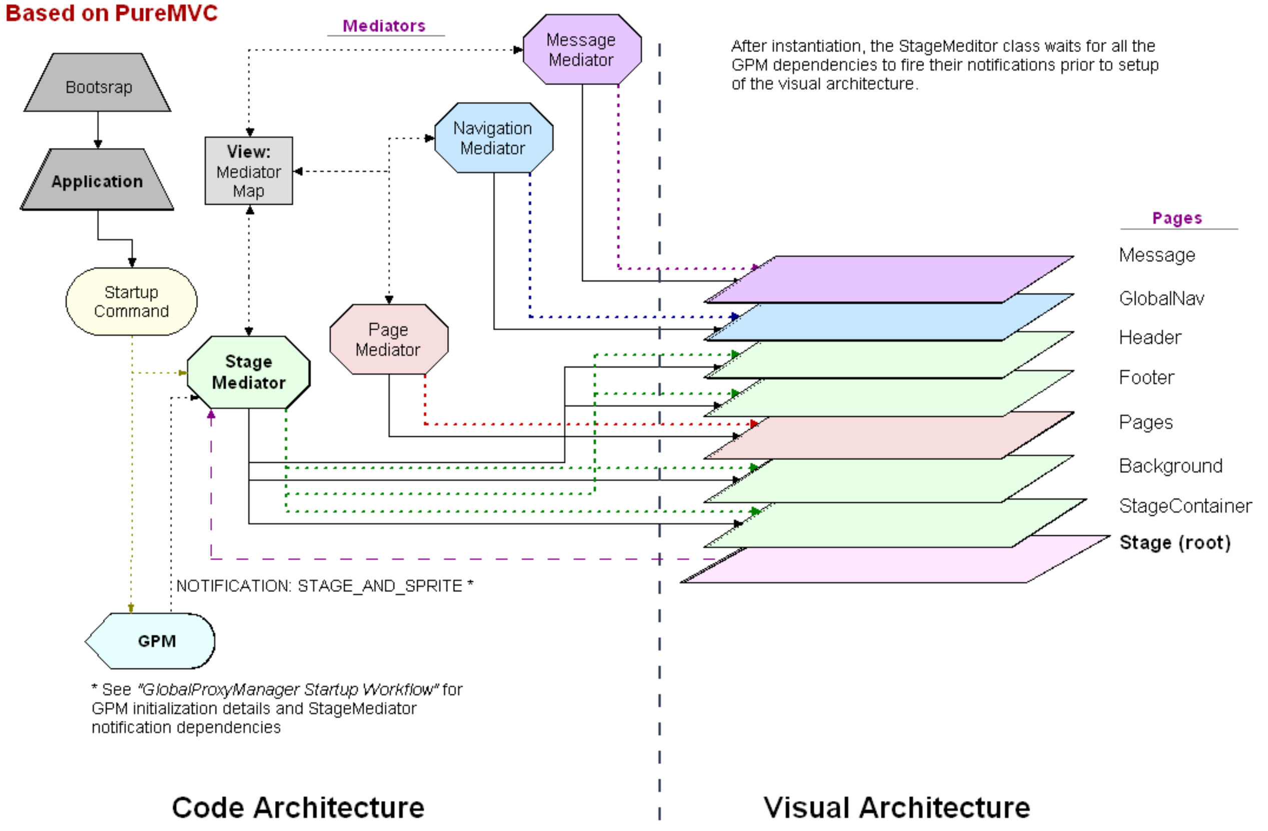
Supported Features

Here's a list of FoxR's main features.

- Model-View-Controller design based on PureMVC
- Written in AS3
- Generates Light Weight compiled SWF's
- Large, custom built component library
- Media library features such as slideshows with custom controls
- Utilizes CSS for visual specifications
- Supports external XML copy files
- Allows for three tiers of configuration
- Rich adapter support for AMF, HTTP and XML
- Form and Data Validation
- Built in authorization support
- Rich I18n workflow support
- Trace and external logging support
- Analytics ready

FoxR Function Architecture

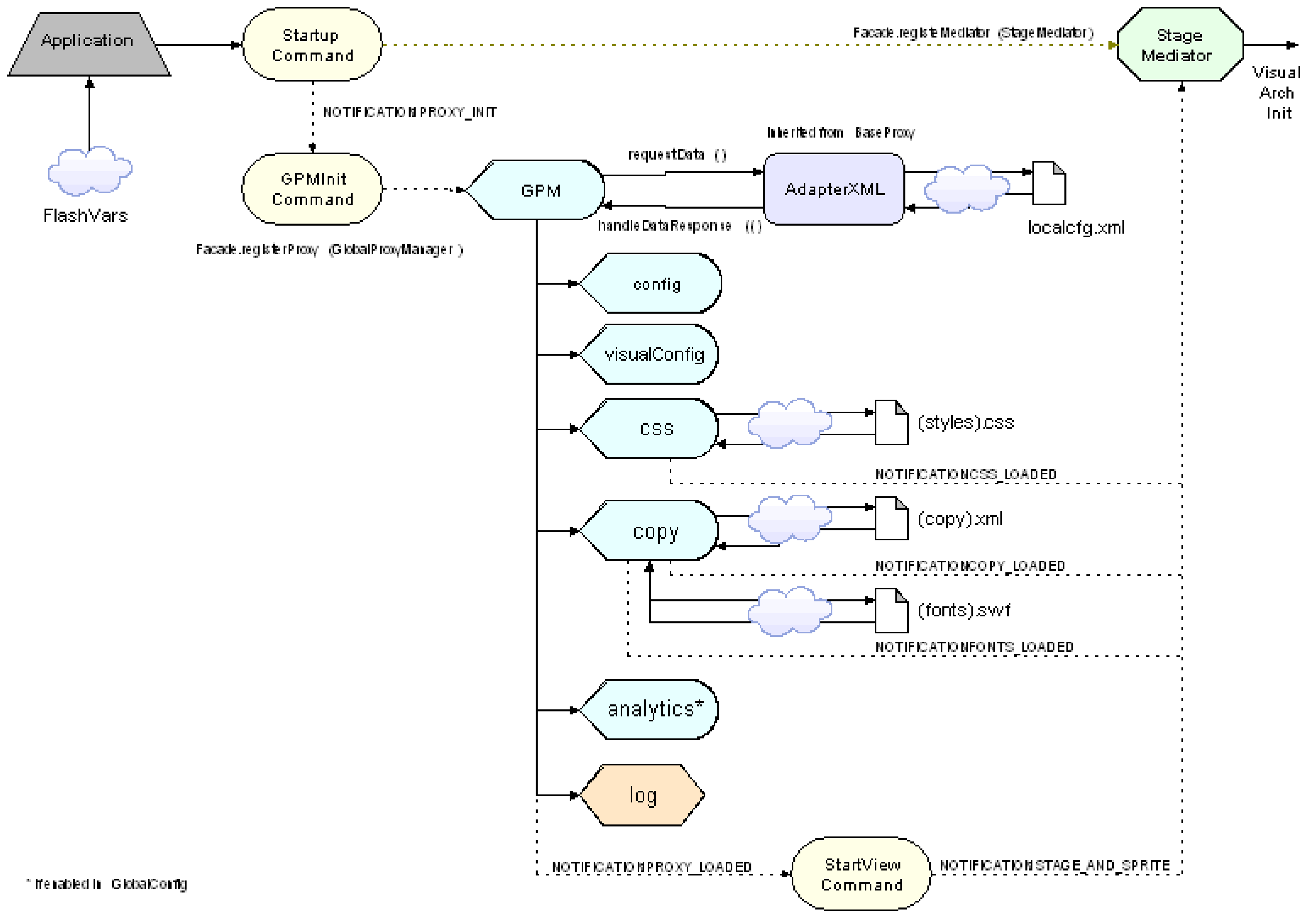
Based on PureMVC



After instantiation, the StageMeditor class waits for all the GPM dependencies to fire their notifications prior to setup of the visual architecture.

* See *"GlobalProxyManager Startup Workflow"* for GPM initialization details and StageMediator notification dependencies

GlobalProxyManagerStartup Workflow



Architectural Goals

FoxR's primary goals revolve around three things: high performance, speed of execution and scalability all within the smallest possible footprint.

To meet these goals, Aeolian Digital Studios is committed to real world testing with benchmarking, unit testing, re-factoring, condensing, simplifying and streamlining every component along the entire application roadmap.

From both an architectural s and development perspective, the FoxR framework has been created with the following goals in mind:

- **Ease of setup and entry to application:** Build as much of the repetitive supporting architecture ahead of time allowing for the bulk of development time to be spent on developing against the specific business rules and look & feel requirements for the application
- **Get out of the FLA:** Flash is a notoriously poor choice for a development platform when the bulk of the development work is centered within a single binary FLA file. FoxR is built for collaborative development being that it is based solely in ActionScript files, requires no FLA to be compiled from and is designed using industry standard conventions such as the MVC, singleton and factory design patterns.
- **Loose Coupling:** The term coupling describes the degree to which different classes must rely upon one another to get their job done. The less a class must depend on

another class, the more flexible and reusable it is in the long run. FoxR strives to provide a loosely coupled and flexible development framework.

Roadmap

Versioning

FoxR will follow the MONO version convention:

MAJOR.MINOR.SUBRELEASE

Even-numbered minor releases are stable releases, while odd-numbered releases are unstable, typically used during development and is contained as part of the code that is only checked in on subversion or that is available as part of the daily builds.

The sub-release indicates a minor release or an update to a specific release.

Current Release

The current release of FoxR is an unstable 0.1 Pre-Beta release for a select group of developers to review.

FoxR 0.2.1 Alpha Developer Release
Release: October 21, 2009

- Maintenance Release

Upcoming Releases

Below is a tentative list of future scheduled releases. Dates to change as necessary.

FoxR 0.5 Beta Release 1
Planned Release: Winter 2010

- Full AMF support with working AFM gateway tests
- Video playback support (simple components and API)
- SWFAddress Proxy support
- Run time and JS log enabling/disabling
- Run time dashboard
- Integration of peer feedback and suggestions
- Bug Fixes
- Re-factoring (As necessary)

FoxR 1.0
Planned Release: TBD

- Component quirks fixed
- Bug Fixes (critical to medium)
- Re-factoring (As necessary)
- Stable Build
- Full Demo/Tutorial project
- Project Examples

IV. General Topics

Code vs. Visual Architecture

Configuring FoxR

FoxR allows you to set configuration values using a cascading, three step design. What this means is that global values set in the first step can be overridden by values set in the next step and so on.

- **Global Config Class** (located in /flash/com/foxr/data/GlobalConfig.as in the Project Template zip)
  This is the top most config level. These settings are applied when the GPM first begins its init() sequence. Any value set in this file can be overridden in either of the two succeeding levels.

  Setting values in this class is beneficial if you plan to create a base RIA/web site that will be re-skinned or re-branded after the project has been compiled and distributed.

- **Local Config XML** (located in /deploy/xml/localcfg.xml in the project template)
  Local Config is required and automatically loaded by the GPM on movie start-up. Any value set here overrides any default settings in the Global Config.

  This file is best utilized by projects that wish to reskin or rebrand a compiled project without having to recompile or change the SWF. Changes made here are applied at run-time.

- **Flash Vars** (set in the embed code of the calling HTML page, only affect the VisualConfig Proxy at this time)
  Values set in FlashVars will supersede those set in either the Global or Local config files, but at this time, are only applied to the VisualConfig Proxy.

Using non-specified variables

While FoxR has a wide range of architectural and visual configuration properties prebuilt into the Config and VisualConfig Proxies, you can also pass in and use your own custom variables and values as well. Any Config or VisualConfig variable/value pair that is not recognized as a standard property of either Proxy class is automatically assigned to a special **FlashVars** property available in both Proxies.

The Element and CompoundElement classes

All display classes in the FoxR framework extend from the base **Element** class. The Element class is an extension of the core AS3 Sprite class that provides a number of time saving properties and methods to all child objects in the framework.

For the most part, Element is used to create simple display objects, or objects that will not contain other child Elements. This includes graphic classes such as Box and Circle which create visual shapes and patterns to be utilized and manipulated in more complex container classes such as pages.

The Element class does provide some very useful options for display including the overflow property and the ability to add a drop shadow to the object.

All FoxR elements include a built in reference to the GlobalProxyManager singleton which can be referenced from the built-in **gpm** property.

CompoundElement

The CompoundElement class extends the base Element class and should be used as a basis for all complex UI objects. This includes objects that utilize one or more primitive Element objects to form a new class or component.

By default, CompoundElement contains a background object (Box) and a text field (TextElement) and the methods and properties necessary to customize them via CSS classes. CompoundElement also adds additional overflow support, alignment, and scrolling.

When to use Element vs. CompoundElement

Scenario	**Use Element?**	**Use CompoundElement?**
An instance where you would use an AS3 Spite and want to utilize the GPM	🟩	🟥
Creating an object that extends the AS3 Flash Sprite object and uses the drawing API	🟩	🟥
Creating an object that contains a text field and background by default	🟥	🟩
Creating an object that should mask content outside a specified height and width range	🟩	🟩

Understanding the objReady() function

Every Element supports a protected **addedToStage()** method which alerts the class that it is part of the active DisplayList. This method is called when the Event.ADDED_TO_STAGE event is fired after object instantiation.

Every Element includes a public method called **objReady()** which is fired when addedToStage() completes. It is recommended practice to utilize objReady() to implement any functionality that depends on the object having been added to the stage.

Global Variables and Constants

There are two types of methods of storing global variables in FoxR.

Global Variables

These are variables that are available globally to all classes in the framework. They may be set either via FlashVars or from within the code. The recommended method is to use the FlashVars property of the ConfigProxy class to store and access these variables.

Examples:

Set FlashVars data:

```
gpm.config.flashVars[‘varName’] = ‘This is a value’;
```

Access FlashVars data:

```
var data:String = gpm.config.flashVars[‘varName’].toString();
```

One of the benefits of using this method is that FlashVars can be accessed and more importantly created and modified at run-time. If you Flash project requires a dynamic way to store data during movie execution, config.flashVars is the recommended method.

Static Constants

Global Static constants are stored in one of two places:

- **Global Constants** – located in the FoxR library at com.foxr.data.GlobalConstants – This file stores constants utilized by the main library itself.

- **Local Constants** – Stores constants for the current working project. In the local project template, these are stored in flash\com\foxr\data\LocalConstants.as

Unlike Global Variable seen above, static constants CANNOT be modified at run-time.

Using CSS

FoxR uses CSS Stylesheets as its primary mechanism to manage the presentation of the movies content. Stylesheets are formatted in standard CSS format. There are, however, important usage differences between HTML CSS and Flash CSS, which are noted in the **"Important CSS Usage Notes"** section.

The CSSProxy Class

The CSSProxy class provides custom CSS support to all Elements within the FoxR framework. It is loaded and instantiated by the GPM on movie start-up. It parses and stores all specified external spreadsheets.

Loading Stylesheets

Stylesheets are specified in the localConfig.xml file as such:

Example:

```
<stylesheets>
      <file>global.css</file>
      <file>winter.css</file>
      <file>summer.css</file>
</stylesheets>
```

Stylesheets are loaded in the order specified and any properties of selectors defined in subsequent stylesheets will override the values set in previously loaded files. This allows for stylesheets overriding global settings with more local or specific values.

Applying CSS styles to elements via the CSSProxy getStyle() method

To apply a CSS style to a FoxR element call the CSSProxy function **getStyle** and pass the appropriate style sector from your CSS Stylesheet.

Example:

```
CSS
square {
      color:#C0C0C0;
      height:25px;
      width:25px;
      x:15;
      y:100;
}

Code
square = Box(addElement('square',Box,gpm.css.getStyle('square'));
```

NOTE: The Element.addElement() method allows you to pass either a custom properties object OR a CSS style object as its optional third and fourth parameters.

You can also apply CSS styles using the Element applyProperties method:

Example:

```
square.applyPropeties(gpm.css.getStyle('square'));
```

Applying CSS Styles to Text

TextElement styles are applied differently from display element styles. They are passed as TextFormat objects and generated by the CSSProxy's **getTextFormat()** method.

Example:

CSS
```
text_style {
      color:#C0C0C0;
      text-size:10px;
}
```

Code
```
text = TextElement(addElement('text',TextElement,{text:'Some text
here',style:gpm.css.getTextFormat('text_style')}));
```

Important CSS Usage Notes

Using CSS in Flash differs from CSS used on HTML pages. The following important restrictions apply to CSS syntax and usage:

Selector case

Flash CSS **converts all selectors, regardless of case, to lowercase** when they are parsed. Selectors that use upper case or camel (mixed) case syntax when being called from wither getStyle() or getTextFormat() **will be ignored**. **Underscore characters are supported** to allow you to break selectors into separate meaningful words for readability.

Example:
```
goodselector { }
good_selector { }
badSelector { }
BAD_SELECTOR { }
```

Comments

CSS (C Style) comments are **not allowed** and **will break** the CSS if used. To add comments to your Flash CSS, you can add them as a separate selector with comment labels specified as selectors and comments as the values:

Example:
```
comments {
     author: Jeff Fox;
     version: 1.0;
     player-version: 9.0;
}
```

Arrays – HTML CSS does not support embedding arrays as CSS values. The FoxR CSSProxy object, however, does provide support for simple data arrays. The CSSProxy allows you to pass an array of values and will automatically convert the data into a Flash array when the CSS is loaded.

CSS Example:
```
selector {
     var-list:[var1,var2,var3];
}
```
ActionScript Example:
```
public function set varList(a:Array):void { [code…] }
```

Booleans – Flash CSS does not support passing Boolean values. They are converted to strings when parsed and the Boolean value is ignored by ActionScript. Instead, **alternate properties that use CSS text** equivalents should be used for any set/get methods that would accept Booleans as arguments.

The following example shows that to set an Elements visible property to TRUE, the Element.visibility property (a valid CSS1+ selector) is used passing the “visible” argument.

Example:
Instead of `visible: true;`
use `visibility: visible;`

Using Fonts and Copy

FoxR is designed to use external font and copy XML files as a means to embed textural content into its movies and applications. The CSSProxy compliments this object by applying text styles and formatting via web standard CSS class.

FoxR Copy XML Workflow

The **com.fox.model.CopyProxy** object is the main class used to load, store and access copy within the movie.

Loading external copy XML files

External copy XML files are stored in the /deploy/xml folder of the projects directory. By default, FoxR automatically retrieves external copy XML files that are specified in the LocalConfig.xml file. Copy XML files are specified as such:

Example:

```
<copyXML>
      <file>global.xml</file>
      <file>instructions.xml</file>
</copyXML>
```

XML files are loaded in the order specified and the copy strings contained in each files are accessible via object path notation.

Accessing copy from loaded copy XML files

Copy XML Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<global>
      <general>
            <title>Sample Title</title>
      </general>
      <home>
            <body>Hello World! I'm here</body>
      </home>
</global>
```

When copy XML files are loaded, they are stored in the CopyProxy object and referenced by the name of the base node. In the case of the above example, this XML copy file would be accessible under the **global** namespace.

Loaded copy files are available as part of the global copy property of the **GlobalProxyManager** class and referenced by using the following code:

FoxR ActionScript Example:

```
var title:String = gpm.copy.getCopyString('global.general.title');
```

Applying CSS formatting to text

Font styles are specified as CSS styles and applied to text in the FoxR framework via the GPM CSSProxy object.

CSS Example:

```
header_text {
        font-family:Arial;
        font-size:32px;
        color:#000000;
}
```

FoxR ActionScript example:

```
if (gpm.css.getTextFormat('header_text') != null) {
        txt.style = gpm.css.getTextFormat('header_text');
}
```

Loading Fonts

Supporting Multiple Languages

Analytics

FoxR includes built in support for sending page click (as well as any other user generated event actions) to external Analytics tracking applications. At present, FoxR only sends analytics events to one application, but this may change in future releases.

Google Analytics

By default, FoxR includes built in support for Google Analytics. To utilize Google Analytics within your FoxR project:

In the GlobalConfig file, Set the ConfigProxy Analytics setting to TRUE.

Configure the Google Analystics output by uncommenting the *analyticsConfig* object. The trackingURL property of this object is configured for the modern JS tracking string and should not need to be modified unless you are using the older urchin.js Analytics script.

In the *pathIDs* property, add each "page" or event that should be tracked. Each tracking event consists of an event label and a tracking value that is sent through to Google Analytics.

Example:
```
      pathIds: {
            'home': 'Home',
            ‘main’: ‘Main Page’
      }
```

In the above example 'home' is the simple tracking label used throughout the movie to identify the tracking event. The 'Home' property is the value sent through and tracked by Google Analytics.

Additional Analytics Applications support

FoxR has conventions in place to add support for third party tracking applications such as Omniture. The built in Analytics Factory allows you to add support for custom built Analytics applications "driver" objects in your movie. Consult your analytics engineer for requirements and specifications for connecting FoxR tracking to your desired application.

Logging

FoxR allows you to log and trace your flash using a number of output options.

Default Log Object

By default, FoxR sends all internal logging messages to the generic com.foxr.util.log.Log class which sends all output through the Flash `trace` function. You can optionally choose to direct log output to one of three other types of supported external logging applications including:

- LuminicBox
- FireBug (for Firefox users)
- ActiveLog (for IE users)

To change the log output device

In the GlobalConfig file, set the externalLogging property to TRUE and set the loggerType property to one the available options (which are cataloged in the GlobalConstants.as file):

- LOG_LUMINICBOX
- LOG_FIREBUG
- LOG_ACTIVELOG

Note: FoxR will need to be recompiled after a change is made to these settings.

Adapters

FoxR provides a streamlined and powerful way to interact with external data sources by means of the Adapter system.

All adapters are based off a global Adapter class which provides the basic functionality common to all adapters. Each individual adapter customizes and extends this base class adding addition custom functionality for the individual type of data workflow. The adapters currently available in FoxR include:

- **AMF Adapter** – Specialized adapter for working with AMF data sources such as OpenAMF (Java) and AMFPHP. Currently somewhat incomplete. Requires definition of ValueObjects and Request/Response objects for external data.
- **XML Adapter** – enables a streamlined way to load and parse external XML files
- **HTTP Adapter** – Streamlines submitting and loading data through HTTP GET and POST data sources.

Planned adapters

- WSDL/SOAP

V. Appendix I – CSS Selector Support

Standard CSS Selectors currently supported:

Key: 🟩 = Full Support, 🟨 = Partial, Incomplete Support, 🟥 = No Support,

Selector	Element	ComplexElement	Children
top	🟥	🟥	🟥
left	🟥	🟥	🟥
width	🟩	🟩	🟩
height	🟩	🟩	🟩
overflow	🟩 (visible, hidden only)	🟩 (visible, hidden only)	🟩 (auto and scroll available on in ScrollPane or its descendants)
position	🟨	🟨	🟨
padding	🟥	🟩	🟩
padding (left, right, top, bottom)	🟥	🟥	🟥
text-align	🟥	🟩	🟩
vertical-align	🟥	🟩	🟩
visibility	🟩	🟩	🟩
background-color	🟥	🟩	🟩
background-image	🟥	🟩	🟨*
border-width	🟥	🟩	🟩
border-color	🟥	🟩	🟩
border	🟥	🟥	🟥
border (left, right, top, bottom)	🟥	🟥	🟥
color *	🟥	🟥	🟨
scrollbar-face-color &	🟥	🟥	🟨
scrollbar-track-color&	🟥	🟥	🟨
scrollbar-arrow-color&	🟥	🟥	🟨
scrollbar-3dlight-color	🟥	🟥	🟥
scrollbar-darkshadow-color	🟥	🟥	🟥
scrollbar-highlight-color	🟥	🟥	🟥
scrollbar-shadow-color	🟥	🟥	🟥
Text Styles			
font-family ☐	🟥	🟥	🟨
font-size ☐	🟥	🟥	🟨
color ☐	🟥	🟥	🟨
text-decoration ☐	🟥	🟥	🟨
font-weight ☐	🟥	🟥	🟥
font-style ☐	🟥	🟥	🟥

* - See special usage details below
& - Supported by the Scrollbar object implemented by ScrollPane and descendants
☐ - Supported by TextElement:style or stylesheet methods

Custom FoxR CSS selectors and support

Selector	Element	ComplexElement	Children
x	□	□	□
y	□	□	□