

BIRDBASE / 2011

VISH VISHVANATH LEAD

Developer's Guide

Spring 2011

TODO LIST

## CONTENTS

---

I	BIRDBASE: A MICROSITE FRAMEWORK	1
1	INTRODUCTION	2
1.1	Explaining BirdBase	2
1.1.1	What is a microsite framework?	2
1.1.2	What features does Birdbase have?	2
1.1.3	Features in More Detail	2
1.1.3.1	Deep Linking and Browser Navigation	2
1.1.3.2	Asset Loading	2
1.1.3.3	Boot Management	3
1.1.3.4	Runtime Configuration Management	3
1.1.4	Accessing Configuration in Actionscript	5
1.1.4.1	Support for locale-specific variables	7

Part I

BIRDBASE: A MICROSITE FRAMEWORK

## INTRODUCTION

---

### 1.1 EXPLAINING BIRDBASE

#### 1.1.1 *What is a microsite framework?*

BirdBase is a tool to build flash websites. Microsites are often used for single product areas, such as cars and games, as part of larger sites. A microsite will typically have to load images, videos and localized text from the main site's back-end, as well as supporting tracking/analytics and deep-linking from marketing newsletters or advertising.

#### 1.1.2 *What features does Birdbase have?*

The feature list is based on my experience in building microsites and the common, recurring features that are required.

- Deep Linking and browser navigation compatibility
- Asset Loading for both design assets (look of site, fixed) and rich media (locale-specific, dynamic)
- Boot Management
- Configuration Management
- Support for locale-specific variables

#### 1.1.3 *Features in More Detail*

##### 1.1.3.1 *Deep Linking and Browser Navigation*

Deep linking is the ability to control a single flash application from the URL. Instead of going to the first screen and having to click your way through the application, a deep link should recreate an application state. It's helpful to clients if we can send customers directly to specific parts of the application, such as a video section. It's helpful to customers if they can bookmark a specific part of the application.

Browser Navigation with Flash means being able to go backwards and forwards using the normal browser buttons. This is user-friendly.

These two are accomplished with judicious use of the SWFAddress library, which consists of a few AS3 classes and some JavaScript code to pass URL information between Flash and the browser.

##### 1.1.3.2 *Asset Loading*

The AssetLoader AS3 library is used to load external files, and is configured at boot from the load.xml file in the locale folder. All rich media assets such as

images and sounds should be in file. These will be loaded before the application starts, and consequently available all the time.

### 1.1.3.3 *Boot Management*

Boot control is all about starting an application in a systematic way. BirdBase runs through bootstrap step-by-step, and these steps are defined in the Boot-Management command. These steps are easily extended in the configuration, so long as new steps are added only after the framework has initialized.

The steps run in the following order:

1. **Preferences** This step sets up the internal preference model, which is split into restricted and unrestricted zones. The restricted zone is for internally saving values throughout the application life. The unrestricted is the same, except it is modified by any parameters in the URL query string.
2. **Navigator** BirdBase uses Eric-Paul's AS3 Navigator to manage application state from the URL and needs to be initialized early on, before any data regarding state/URL is loaded and parsed.
3. **Models** The application data models are created before their related services load any data.
4. **Services** These load data into the models.
5. **Fonts** Loads and registers fonts.
6. **Assets** Loads all assets listed in load.xml
7. **Application Ready** This checks the url at load time, to see if there is any state that need to be loaded at this time. If not, sets application state to the root, as defined in configuration.xml. Application is ready to be started.

Once Boot control broadcasts that the application is ready, your typical application will trigger the StartApplication command.

You have your own ideas on how you want to organize your application, but I suggest that the Start Application process, whether it is one command or several, does at least this:

- Link the views and mediators using the navigator.

The sample navigator application shows how we read the mappings from the configuration and combine them for the navigator.

Your application should now be available for user input.

### 1.1.3.4 *Runtime Configuration Management*

Runtime configuration is, basically, a bunch of keys and values that the SWF loads at bootup. These keys could represent:

- Strings - text used in the site.
- Navigation - a list of items for a menu.
- URLs - that we want to launch in the site or read RSS feeds from, or load media.
- Layout information - x/y coordinates, alpha, etc., so display is configurable.

Clients often want to tailor sites, especially marketing websites, to a variety of locales. This will obviously mean translating the strings, but it may mean turning sections off in some countries or customizing certain graphics for other countries. Runtime configuration might be a text file, but it can be generated by ASP/PHP/Ruby/Java/Whatever, and a CMS can manage it.

I have chosen YAML for the configuration because it's human-readable, processable in a single-pass, and was designed explicitly for data-serialization. It's also lightweight in file size.

A sample YAML configuration file, from the demo application:

```
# --- Start Navigator Configuration ---

# The lines below map between Actionscript classes and URL states.
# the "destination" maps an array of URL routes that should cause the "view" class to display when

nav:
  - item:
      destination: [ "home", "home/*" ]
      label:       Home Section
      view:        HomeView
  - item:
      destination: [ "media" ]
      label:       Media Section
      view:        MediaView

# --- End navigation configuration ---

# --- Begin strings configuration ---
# A key, such as "sound_on" maps to the translated string. You may notice "%s" in some of the strings

strings:
  sound_on:          "on"
  sound_off:         "off"
  tagline:           "This is a tagline, right here. Good Morning %s!"
  newsletter_copy:  "Pellentesque nibh felis, eleifend id, commodo in, interdum vitae"

# --- End strings configuration ---

# --- Begin application configuration ---

root:                "home"
base:                "assets/en_GB/" # the base URL for the application
fontsFile:           "fonts/fonts.swf" # the fonts SWF

# These assets below are loaded at bootup time.

assets:
  - item:
      key: "dynamic_library"
      uri: "assets.swf" # The dynamic library
  - item:
      key: "background"
      uri: "images/main_background.jpg"
  - item:
      key: "foreground"
      uri: "images/main_foreground.jpg"

# Some entirely arbitrary data for your app
```

```

rss_feeds:
  news:
    uri: "http://www.example.com/news.rss"
  video:
    uri: "http://www.example.com/video.rss"
  media:
    #screenshots
    - item:
      uri: "http://www.example.com/screenshots.rss"
    #wallpapers
    - item:
      uri: "http://www.example.com/wallpapers.rss"
    #artwork
    - item:
      uri: "http://www.example.com/artwork.rss"
layout_data:
  - item:
    id:    graphic_one
    x:    190
    y:    320
  - item:
    id:    graphic_two
    x:    390
    y:    170

```

Firstly, we have defined the navigation structure. This is not prescribed for your application, but we've provided this with a simple method of creating this navigation in the sample app. You can see that the media navigation item also has three subviews.

Let's break these navigation items down.

The item destination is the state, or URL, which relates to this view. The label is the locale-specific translation string, and the view is the AS3 Class which should be displayed for this state or destination.

The subviews are purely there for creating a tree data structure that is easily represented visually. Subviews are maintained within the application entirely by their destination, and not by any form of parent-child relationship. This is important. All views are independent of each other, but multiple views can be layered by way of compound destinations.

Further down the YAML, we see a restricted section. Some are mandatory, the root and viewpath, and you may put useful properties in here. For example, a client wishes to control the position of their logo depending on the locale (e.g. for countries that read from right to left). You can put properties in here that you can later read in the application and use to move the logo.

The unrestricted section is for user variables. These will be set or changed by appearing in the URL query string. For example, if your query string is `?colour=blue&town=london`, the application will set two unrestricted preferences, colour to blue and town to london. Restricted and unrestricted preferences are available through the Preferences Class, which contains both namespaces.

#### 1.1.4 Accessing Configuration in Actionscript

```
# --- Start Navigator Configuration ---
```

```
# The lines below map between Actionscript classes and URL states.
```

```
# the "destination" maps an array of URL routes that should cause the "view" class to display when
```

```

nav:
- item:
  destination: [ "home", "home/*" ]
  label: Home Section
  view: HomeView
- item:
  destination: [ "media" ]
  label: Media Section
  view: MediaView

# --- End navigation configuration ---

# --- Begin strings configuration ---
# A key, such as "sound_on" maps to the translated string. You may notice "%s" in some of the str.

strings:
  sound_on: "on"
  sound_off: "off"
  tagline: "This is a tagline, right here. Good Morning %s!"
  newsletter_copy: "Pellentesque nibh felis, eleifend id, commodo in, interdum vitae"

# --- End strings configuration ---

# --- Begin application configuration ---

root: "home"
base: "assets/en_GB/" # the base URL for
fontsFile: "fonts/fonts.swf" # the fonts SWF

# These assets below are loaded at bootup time.

assets:
- item:
  key: "dynamic_library"
  uri: "assets.swf" # The dynamic library
- item:
  key: "background"
  uri: "images/main_background.jpg"
- item:
  key: "foreground"
  uri: "images/main_foreground.jpg"

# Some entirely arbitrary data for your app

rss_feeds:
  news:
    uri: "http://www.example.com/news.rss"
  video:
    uri: "http://www.example.com/video.rss"
  media:
    #screenshots
    - item:
      uri: "http://www.example.com/screenshots.rss"
    #wallpapers
    - item:
      uri: "http://www.example.com/wallpapers.rss"
    #artwork
    - item:
      uri: "http://www.example.com/artwork.rss"

```

```
layout_data:
- item:
  id:    graphic_one
  x:    190
  y:    320
- item:
  id:    graphic_two
  x:    390
  y:    170

settings.getSetting( "rss_feeds").media
```

#### 1.1.4.1 *Support for locale-specific variables*

Well, this is essentially covered by the above point about configuration. All locales have their own configuration file, so all strings should be localized anyway. The strings section in the configuration looks like this:

```
strings:
  bluebutton: "A Blue Button!"
  redbutton:  "A Red Button!"
  tagline:    "This is a tagline, right here. Good Morning %s!"
  secondviewselected: "This is the <b>second</b> view of the %s application.
                  Hurrah for the colour %s!
                  Some more %s would really brighten up this view."
  thirdviewselected: "This is the third view of the application.
                    You've already selected: %s and %s!"
```

You can register any object that implements `IUpdateableText` with the `TextService` class. You register the string name that you want pushed to this object, along with any variables that should be injected into the `%s` points, and the `TextService` will do the the rest by automatically updating your text object.

Let me clarify that point about injecting variables. This is essentially a `sprintf` command, where you have a string as such "Good morning

```
textService.register( IUpdateableText, "tagline", "Vish" );
```

and you receive:

Good morning Vish!

The `TextService` supports runtime dynamic reloading of strings, so if these are reloaded, in say, a different language, all your `IUpdateableText` instances will be reformatted and updated immediately.

