



## **Cheer Wine Technical Specifications**

**January 27, 2009**

**Version 1**

## Contents

Cheer Wine Technical Specifications .....	1
Introduction .....	3
Modules .....	3
The Current Asset Loading Process .....	3
Page Loading .....	5
The Preloader .....	6
PageProxy .....	6
Loading the Page Manifest .....	6
AssetProxy .....	7
Making a Page Request .....	9
Handling Page Requests .....	9
Deep Linking .....	10

## Introduction

The Cheerwine application has two primary pages displaying numerous assets that can be categorized in two ways; those that are required before the page is shown, and those that can be loaded after the page is shown, and introduced when available. This document details the changes required to the current application in order to facilitate loading pages and their assets, as well as handling changes to focus between them.

## Modules

Name	Module Name	Swf name
Beach Background	BeachBackgroundModule	beachbackground.swf
RumpleChillskin	RumpleChillskinModule	rumplechillskin.swf
Plane	PlaneModule	plane.swf
Beach host	BeachHostModule	beachhost.swf
Art Of Cheer wine	ArtOfCheerWineModule	artofcheerwine.swf
Tree	TreeModule	tree.swf
Space Background	SpaceBackgroundModule	spacebackground.swf
Musical Alien	AlienModule	alien.swf
Space Host	SpaceHostModule	spacehost.swf
Lunar Lander	LanderModule	lander.swf
Space Needle/Rage Gauge	RageGaugeModule	ragegauge.swf
Sparkletron	SparkletronModule	sparkletron.swf
Shuttle	ShuttleModule	shuttle.swf

Directory structure

BeachScene

## The Current Asset Loading Process

The Cheerwine application loads its assets based on detailed information provided in the asset\_manifest.xml file. This document provides data such as :

- The “url” attribute, which represents the asset URL to load.
- The “bytes” attribute, which provides the assets size in bytes. This is essential for the preloader to be able to calculate the total length of time necessary to load all assets for any given page.
- The “scale” attribute which provides the assets starting scale when added to the page.
- The “x” and “y” attributes which provide the starting coordinates for the asset.

Listing 1 provides an example asset\_manifest XML document.

```
<assets>
  <asset id="rumple" url="rumple.png" bytes="153738" scale="0.8" x="800" y="150" />
  <asset id="sign" url="sign.png" bytes="74882" scale="0.8" x="55" y="300" />
  <asset id="beach_tile" url="scene_tile.jpg" bytes="34007" scale="1" x="0" y="0" />
  <asset id="beach_bg" url="background.jpg" bytes="69161" scale="1" x="0" y="0" />
  <asset id="beach_scene" url="beach_scene.jpg" bytes="382820" scale="1" x="0" y="0" />
</assets>
```

### **Listing 1. The asset\_manifest.xml document structure**

Both the asset\_manifest.xml document and the assets themselves are loaded by the AssetProxy class. The AssetProxy is requested by the LoadAssetsCommand at startup which first loads the manifest XML document. Once the document has loaded, the proxy iterates through each of the documents asset nodes, loading each associated asset one at a time. Once all the assets have loaded, the notification NOTIFY\_ASSETS\_LOADED is sent.

Currently, the AssetProxy assigns data such as the assets x and y values, as well as the assets starting scale value which it has retrieved from the asset\_manifest.xml document, directly to the asset Loader object. This way, when the asset is added as a child to a visible DisplayObject, it is already positioned.

Figure 1 provides an overview of the current asset loading process.

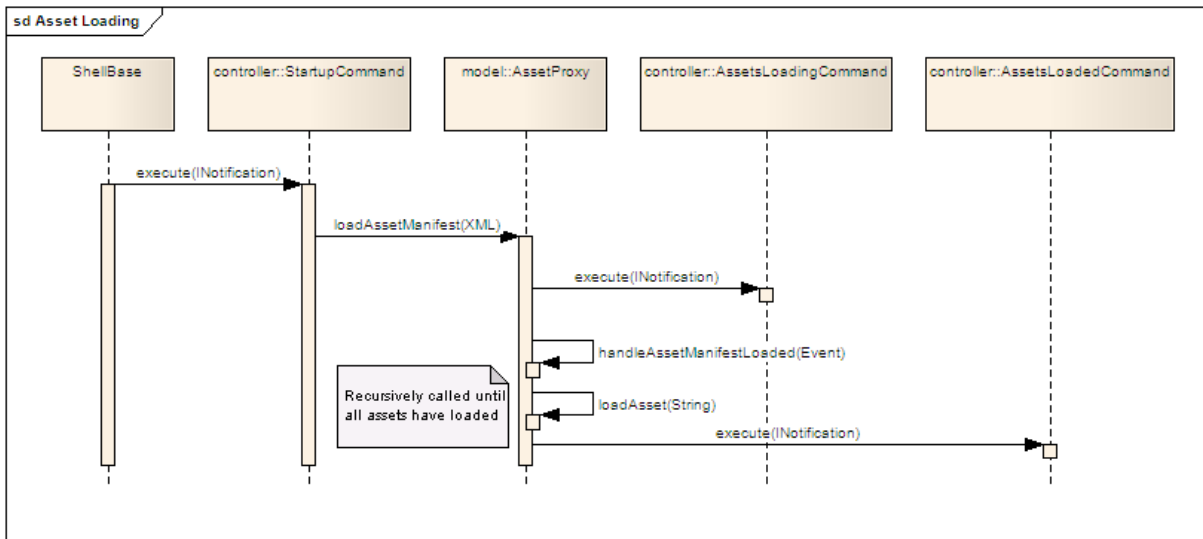


Figure 1 : The current asset load process

## Page Loading

With the advent of pages, the Cheerwine assets will be divided between what is required for the currently requested page, and what is required for other pages in the application. When the Cheerwine application is first requested, the AssetProxy will still be responsible for loading all of the assets detailed in the asset\_manifest.xml document at startup. However, each of the assets will be prioritized and loaded in order. Once enough of the assets required for the currently requested page have fully loaded, then the user can be presented with the page while the rest of the assets load silently in the background.

To facilitate this asset prioritization, a new document titled page\_manifest.xml will be created. This document will list the types of pages available in the application as “page” nodes and provide each of these nodes with a list of each of the assets the particular page requires. This way, when a page is requested by the user, the Cheerwine application can perform a lookup within the page\_manifest.xml file for the assets associated with the requested page, and mark those assets as a priority load. The application will then look to the asset\_manifest.xml for the location of the asset and commence loading.

As speed is of the essence, it is imperative that the page requested by the user be presented as soon as possible. This means that all the necessary assets for the requested page have been loaded and placed within the scene, and the preloader removed from view so that the user can begin to interact. While prioritizing the asset loading, so that the current page assets are loaded first, will create a speed increase in loading the application, further load time gains can be made by dividing a pages assets between those assets that are immediately required, and those that can simply be shown when they are ready. The page can be presented to the user before all of the page assets are loaded providing all of the immediately required assets are loaded and placed.

The page\_manifest.xml file, as detailed in listing 2, will tag each asset node for a given page with a “preload” attribute. if the asset needs to be loaded before the page is shown, the preload attribute will be set to “true”. Alternatively, assets that are of a lower priority will have a preload attribute value “false”, or will simply not have the attribute value at all. When the page\_manifest.xml document has been parsed, the assets will be further

prioritized based on the “preload” attribute, placing those assets with a preload value of true ahead of any other page asset.

```

<pages>
  <page id="home">
    <asset id="rumple" />
    <asset id="host" />
    <asset id="plane" />
    <asset id="sign" preload="true" />
    <asset id="beach_tile" preload="true" />
    <asset id="beach_bg" preload="true" />
    <asset id="beach_scene" preload="true" />
  </page>
  <page id="space">
    <asset id="alien" />
    <asset id="host" />
    <asset id="ragegauge" />
    <asset id="shuttle" />
    <asset id="lander" />
    <asset id="sparkletron" />
  </page>
</pages>

```

#### **Listing 2. The page\_manifest.xml document structure**

The asset nodes of the page\_manifest.xml file will be paired with those nodes in the asset\_manifest.xml document using the assets “id” attribute. This attribute is unique across all pages.

#### **The Preloader**

Before any assets can be loaded, the user must first be presented with the preloader. The preloader is an external asset, so will be loaded by the AssetProxy with a call to the loadPreloader method. Once the preloader has loaded, the NotificationConstants.PRELOADER\_LOADED notification will be sent, which will be handled by the PreloaderLoadedCommand. As the preloader loading signifies the start of the application, this command will react by sending the NotificationConstants.SHOW\_PRELOADER notification, which will execute the ShowPreloaderCommand.

#### **PageProxy**

The PageProxy class will be responsible for keeping track of which page and page area has the current focus.

#### **Loading the Page Manifest**

As the page\_manifest.xml file is an asset, the document will be loaded by the AssetProxy class at startup. Once the document has been loaded, the AssetProxy will send the NotificationConstants.PAGE\_MANIFEST\_LOADED notification, which will be intercepted by the PageManifestLoadedCommand.

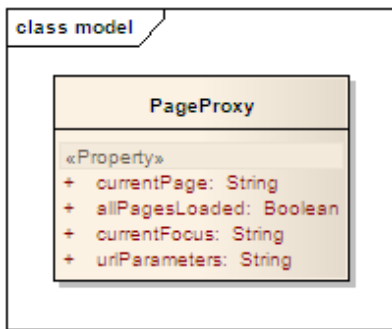


Figure 2. The PageProxy class

### AssetProxy

The AssetProxy is responsible for loading the asset\_manifest.xml document, all page assets, and maintaining references to all assets.

The AssetProxy is requested to load the asset\_manifest by the LoadAssetManifestCommand. The AssetProxy will use Bulkloader to accomplish this, and once loaded, will send the NotificationConstants.ASSET\_MANIFEST\_LOADED notification which will be intercepted by the AssetManifestLoadedCommand.

The LoadAssetsCommand will request the AssetProxy to load the assets, but before it does, it will first order the assets by priority. The assets will be grouped into four categories;

- those that are required by the requested page before it can be revealed to the user for interaction.
- those that are required by the requested page, but lazily, so they will not need to be introduced until they are ready.
- those that are required by the other application pages before a page change request can be fully completed.
- those that are required by other applications pages, but are not required for the other pages to be revealed.

All items within the asset\_manifest.xml that have the preload="true" attribute will be deemed required for its page before the preloader is withdrawn. The application will be able to work out what asset belongs to what page using the page\_manifest.xml document.

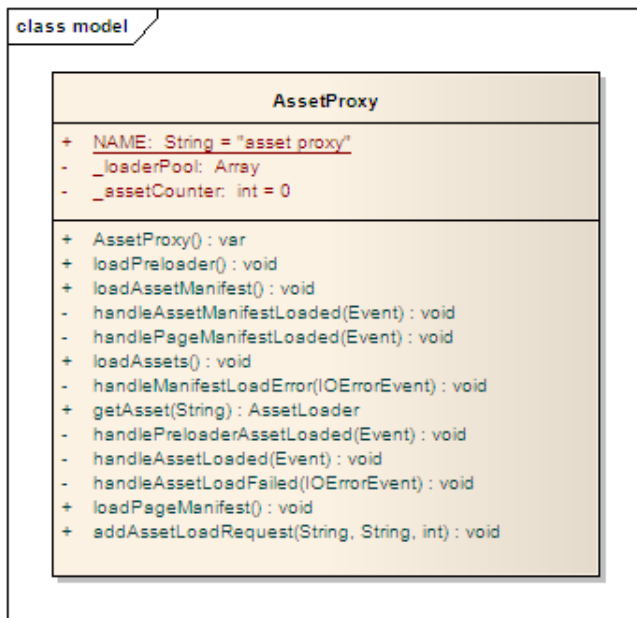


Figure 3. The AssetProxy class

The AssetProxy will load the assets using the Bulkloader framework, which can assign priority to the assets provided. Each asset URL will be passed to the AssetProxy addAssetLoadRequest method, along with its priority value, one at a time. Once all asset URL's have been passed, the AssetProxy loadAssets method will be called, which will start the Bulkloader loading all the requested assets.

Once the first and highest priority assets have loaded, which are those for the current page with the preload="true" attribute, the handleAssetLoaded method will send the notification NotificationConstants.REQUIRED\_PAGE\_ASSETS\_LOADED. This will alert the application that all the assets required for the current page have loaded, and thus the page can now be presented to the user.

For the second level priority assets, the handleAssetLoaded method will dispatch the NotificationConstants.ASSET\_LOADED notification for each and every asset, which will simply alert whomever is listening that the current asset is ready for use within the current page. The method will be able to assume this requirement by comparing the currently loading assets page value with the currently viewable page identifier property, currentPage, from the PageProxy.

Up until this point, the property allPagesLoaded, of the PageProxy class, will be set to false. Once the assets from the first three priority levels have been fully loaded, this value will be set to true, regardless of the loading state for the final fourth priority level. The fourth priority level assets are associated with the page the user is not currently viewing, for assets that are not immediately required for the page, even if the user navigates to the page immediately upon entering the application. Thus, these assets can be used simply when they are loaded, and will not affect the available areas of the application. If, however, the user changes page while allPagesLoaded is still false, then the preloader will be displayed until NotificationConstants.REQUIRED\_PAGE\_ASSETS\_LOADED is sent. The handleAssetLoaded method will be able to check for this requirement by comparing the currentPage property of the PageProxy value as before. When the page has changed, the current page will have changed, thus the

NotificationConstants.REQUIRED\_PAGE\_ASSETS\_LOADED becomes relevant to that page.

## Making a Page Request

Page requests can be made in one of three ways; a default page load, a URL page request and a navigation related page request. The default page load occurs when the Cheerwine application is loaded without specifying a page to view. Typically, such a request will default to the home page without any one area of the page receiving focus.

The URL page request occurs when a user enters a specific URL into the browser that contains details for loading a given page either with or without a required focus. If no page area is requested for focus by the user, then the page will be loaded in a zoomed out state, while specifying status will cause the page to zoom into the requested area once the page has been displayed to the user.

Finally, a page request can also be made by navigating to it via a mouse click on the Moon when currently on the beach page, or by clicking on the Earth when on the Moon page.

As the Cheerwine application endeavors to load all assets when the application is first loaded, performing a page request through navigation would not result in page assets being loaded for that page. However, should the request be made before all of the assets for the page are fully loaded, the user will be presented with a preloader until such a time as the asset loading has completed and all page assets have been positioned on screen ready for user interaction.

### Handling Page Requests

As page requests are made, which include page and focus changes, the PageProxy will be updated directly to store the current application state so that it may be queried at any time.

## Deep Linking

URL based page requests are performed with the use of deep linking. When making a request to the default application page, the URL in the browser address bar would look a little something like this :

<http://www.cheerwine.com/index.html/space/sparkletron3000/?video=12345>

We will be using the extension to the SWFObject library posted to the Ginormous blog, by Kris Range :

<http://theresidentialien.typepad.com/ginormous/2009/01/deep-linking-with-swf-address.html>

Basically the extensions allow organized arguments to be presented to swf object as generic objects and then the extensions convert the data into url friendly terms.

e.g.

SWFAddress.addToParamObj( "ArtOfCheerWine", { pictureId :123} );

Would result in the url

<http://www.cheerwine.com/index.html/artOfCheerwine/?pictureId=123>

The following deep links will be supported

Item	Link	Args
Beach	<a href="http://www.cheerwine.com/index.html/beach">http://www.cheerwine.com/index.html/beach</a>	"beach", null )
Rumple	<a href="http://www.cheerwine.com/index.html/rumple/">http://www.cheerwine.com/index.html/rumple/</a>	("rumple", null)
Art of cheer wine	<a href="http://www.cheerwine.com/index.html/artOfCheerwine">http://www.cheerwine.com/index.html/artOfCheerwine</a>	("artOfCheerWine", null)
Art of cheer with links to art	<a href="http://www.cheerwine.com/index.html/artOfCheerwine/?pictureId=123">http://www.cheerwine.com/index.html/artOfCheerwine/?pictureId=123</a>	("artOfCheerwine", {pictureId:123})
Space	<a href="http://www.cheerwine.com/index.html/space">http://www.cheerwine.com/index.html/space</a>	("space", null)
Alien	<a href="http://www.cheerwine.com/index.html/alien/">http://www.cheerwine.com/index.html/alien/</a>	("alien", null)
Rage Gauge	<a href="http://www.cheerwine.com/index.html/theragegauge">http://www.cheerwine.com/index.html/theragegauge</a>	("theragegauge", null)
Sparkletron	<a href="http://www.cheerwine.com/index.html/sparkletron/?videoId=123">http://www.cheerwine.com/index.html/sparkletron/?videoId=123</a>	("Sparkletron", {videoid:123})

Internally, the URL will be intercepted by the SWFAddressProxy and a NotificationConstants.PAGE\_REQUEST notification will be sent. This notification will result in the PageChangeRequestCommand being executed, which in turn will make the necessary calls to load the necessary assets, and to populate the PageProxy class. As the SWFAddressProxy is alerted even when first loading the application, the PageChangeRequestCommand will always be called at least once before any page is shown. It will therefore be left to the PageChangeRequestCommand to instigate initial asset loading, giving the requested page identifier to the PageProxy so that assets are loaded in the correct order.

When requesting a specific page from the application, the above URL will become appended with an anchor string, which provides a serialized representation of an application state. Such a state will contain data such as :





- Which page to load.
- Which area of the page, if any, should receive focus, thus be zoomed into when the page is presented.
- Further parameters to be passed to the focused page asset to provide instruction on what video / audio / information should be used and shown to the user.

Each property will be available from the PageProxy, utilizing the currentPage, currentFocus and urlParameters properties. The properties will be accessed by the governing SceneMediator when a page request has been made.

