Contents

[Overview of the Amaya Interactive Mobile Platform 5](#_Toc378603760)

[Amaya Gaming System 5](#_Toc378603761)

[Remote Game Framework 5](#_Toc378603762)

[Game Launch Page 6](#_Toc378603763)

[The “amaya” Namespace 7](#_Toc378603764)

[The “amaya.game” Namespace 7](#_Toc378603765)

[Loading JavaScript and CSS files 8](#_Toc378603766)

[Libraries used by the Casino 5 Framework 8](#_Toc378603767)

[The Remote Game Framework 9](#_Toc378603768)

[Casino 5 Game Project Structure 10](#_Toc378603769)

[The Generated Webapp (WAR) 11](#_Toc378603770)

[The Game Manifest (for Casino 5 Games) 11](#_Toc378603771)

[Building the WAR 13](#_Toc378603772)

[The Casino 5 User Interface 14](#_Toc378603773)

[Status Bar 14](#_Toc378603774)

[Display Information 14](#_Toc378603775)

[Displaying Financial Information in Cash 14](#_Toc378603776)

[Displaying Financial Information in Credits 14](#_Toc378603777)

[Game Play Id 15](#_Toc378603778)

[Menu Button 15](#_Toc378603779)

[Game Settings Button 15](#_Toc378603780)

[Menu 15](#_Toc378603781)

[Spin Button 16](#_Toc378603782)

[Display Information 16](#_Toc378603783)

[Using the Spin Button 16](#_Toc378603784)

[Dialogs 16](#_Toc378603785)

[Multiple Dialogs 16](#_Toc378603786)

[Canceling Dialogs 17](#_Toc378603787)

[Determining if a Dialog is Currently Displayed 17](#_Toc378603788)

[Errors 17](#_Toc378603789)

[Developing Games for Multiple Devices 18](#_Toc378603790)

[Aspect Ratios 18](#_Toc378603791)

[Graphic Quality 18](#_Toc378603792)

[Scaling and the “Stage” Size 18](#_Toc378603793)

[Resizing 19](#_Toc378603794)

[Aspect Ratio + Graphic Quality 19](#_Toc378603795)

[How Casino 5 Determines What to Preload 19](#_Toc378603796)

[Falling Back When a Quality/Aspect Ratio is Not Implemented 19](#_Toc378603797)

[A Note (And Warning) About ‘High’ Quality and Retina Displays 20](#_Toc378603798)

[Taking Control Over Sizing 20](#_Toc378603799)

[Working with Preloaded Files 21](#_Toc378603800)

[Using Files Preloaded From the <Quality>\_<Aspect Ratio> Directories: The "device" Keyword 22](#_Toc378603801)

[Using Files Preloaded From the <Quality> Directories: The "device" Keyword continued 23](#_Toc378603802)

[Avoiding Naming Collisions 24](#_Toc378603803)

[Using Preloaded Localized Files: The "lang" Keyword 25](#_Toc378603804)

[Developing Graphics for Multiple Screen Sizes 26](#_Toc378603805)

[Use JSON to Separate Layout from Code 26](#_Toc378603806)

[Design Large and Scale Down 27](#_Toc378603807)

[Design for the Narrowest Ratio First 27](#_Toc378603808)

[Use an Alternative Layout where Required 27](#_Toc378603809)

[Framerate 28](#_Toc378603810)

[Frames Per Second in EaselJS 28](#_Toc378603811)

[Managing the Framerate Manually 28](#_Toc378603812)

[Getting Started Writing a Game 29](#_Toc378603813)

[The amaya.game.Game() Function 29](#_Toc378603814)

[Initializing a Game 29](#_Toc378603815)

[Game Messaging Sequences 30](#_Toc378603816)

[Using V2Game.init() to Easily Start a Game 31](#_Toc378603817)

[Setting Up Your Game 31](#_Toc378603818)

[Removing the Loader 31](#_Toc378603819)

[Using the CanvasManager 32](#_Toc378603820)

[Game Services 32](#_Toc378603821)

[Building the Game 34](#_Toc378603822)

[Build Options 34](#_Toc378603823)

[Using the Client Development Harness 35](#_Toc378603824)

[Tomcat 36](#_Toc378603825)

[Overriding the Default Loading Screen Image 37](#_Toc378603826)

[Loader Image 37](#_Toc378603827)

[JPG Files 37](#_Toc378603828)

[Localized JPG Files 37](#_Toc378603829)

[PNG Files 37](#_Toc378603830)

[Localized PNG Files 38](#_Toc378603831)

[Loader Text 38](#_Toc378603832)

[JSON File Locations 39](#_Toc378603833)

[JSON File Contents 39](#_Toc378603834)

[Using the Credit Displays 40](#_Toc378603835)

[Game Settings 41](#_Toc378603836)

[Control Types 41](#_Toc378603837)

[Stepper Control 41](#_Toc378603838)

[Radio Group 41](#_Toc378603839)

[Boolean 41](#_Toc378603840)

[Auto Spin 42](#_Toc378603841)

[The gameSettings Object 42](#_Toc378603842)

[Creating a Control 42](#_Toc378603843)

[Listening For a Control Change 42](#_Toc378603844)

[Setting the Control Value 43](#_Toc378603845)

[The Spin Button 44](#_Toc378603846)

[Auto Spin 45](#_Toc378603847)

[Auto Spin Flow 45](#_Toc378603848)

[Code Required in the Game to Support Auto Spin 46](#_Toc378603849)

[General Auto Spin Notes 47](#_Toc378603850)

[Creating a Paytable for the Game 49](#_Toc378603851)

[Parameters 49](#_Toc378603852)

[Developing a Paytable 49](#_Toc378603853)

[Sound 50](#_Toc378603854)

[HTML5 Audio Limitations 50](#_Toc378603855)

[Sound Formats 50](#_Toc378603856)

[Using the SoundSystem API 50](#_Toc378603857)

[Encoding 51](#_Toc378603858)

[Naming and Storing the Files 51](#_Toc378603859)

[Preloading the Sounds 51](#_Toc378603860)

[The SoundSystem Object 52](#_Toc378603861)

[Playing Sounds 52](#_Toc378603862)

[Sound Instances 52](#_Toc378603863)

[Using Your Own Sound Solution 53](#_Toc378603864)

[Working Around Audio Limitations on Mobile Devices 53](#_Toc378603865)

[The Sound Selection Screen 53](#_Toc378603866)

[Game Notification Events 54](#_Toc378603867)

[notifyGameStart 54](#_Toc378603868)

[notifyGameEnd 54](#_Toc378603869)

[Mobile Device Information 55](#_Toc378603870)

[Application Icons 55](#_Toc378603871)

[Generating App Icons using the Photoshop Template 55](#_Toc378603872)

[Running the Game as a Web Application 55](#_Toc378603873)

[Localization 56](#_Toc378603874)

[Supported Languages 56](#_Toc378603875)

[Using Strings 56](#_Toc378603876)

[Text Encoding 58](#_Toc378603877)

[Using Images 58](#_Toc378603878)

[Developer Tools 60](#_Toc378603879)

[Query String Parameters 60](#_Toc378603880)

[Resizing The Browser Window in a Desktop Development Environment 61](#_Toc378603881)

[Advanced Topics 62](#_Toc378603882)

[Google Closure Compiler 62](#_Toc378603883)

[Postloading and the “Remote” Keyword 62](#_Toc378603884)

[Coming In Future Releases 65](#_Toc378603885)

# Overview of the Amaya Interactive Mobile Platform

The Amaya Interactive Mobile Platform is a new set of features and tools that augment the existing Amaya Gaming System (AGS) by adding support for development of Remote Gaming Framework (RGF) games built using HTML5.

## Amaya Gaming System

A complete description of the AGS is beyond the scope of this document but at a high-level, the AGS provides components which support the operation of games including managing transactions, providing game replay information, player management etc.

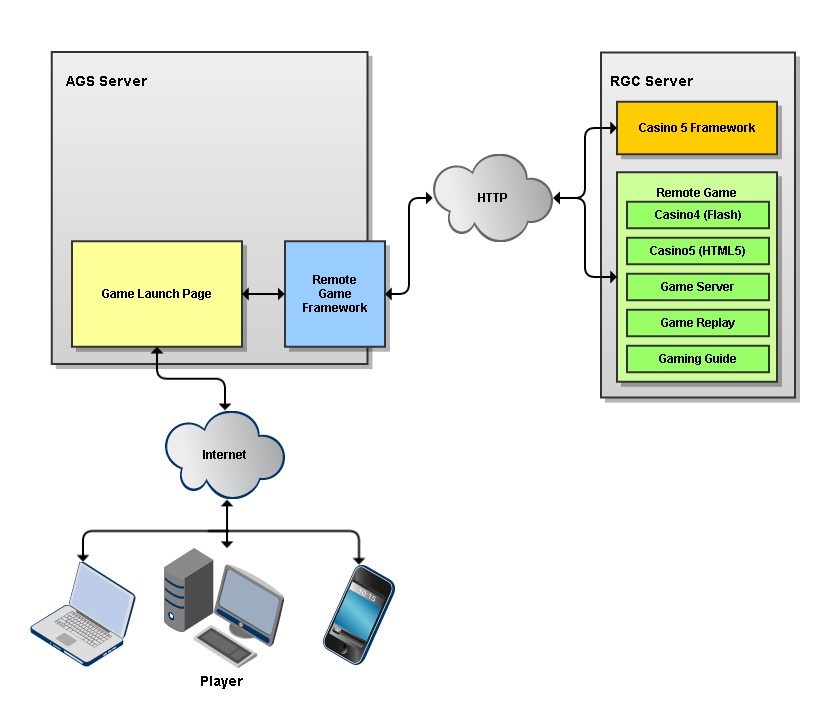
## Remote Game Framework

The RGF is a separate platform that allows developers to build games as individual webapps, deployed separately from the AGS on the Remote Game Container (RGC), which communicate to the AGS via http. Game clients developed on the RGF communicate with the AGS using a set of JavaScript libraries that manage the http communication on behalf of the game. These libraries also provide a number of additional services such as preloading game files, providing a common user interface, and managing the user’s viewport across multiple devices.

With this latest release of the RGF platform, games may now have multiple clients developed for a common server implementation. Currently, the RGF supports the following client technologies:

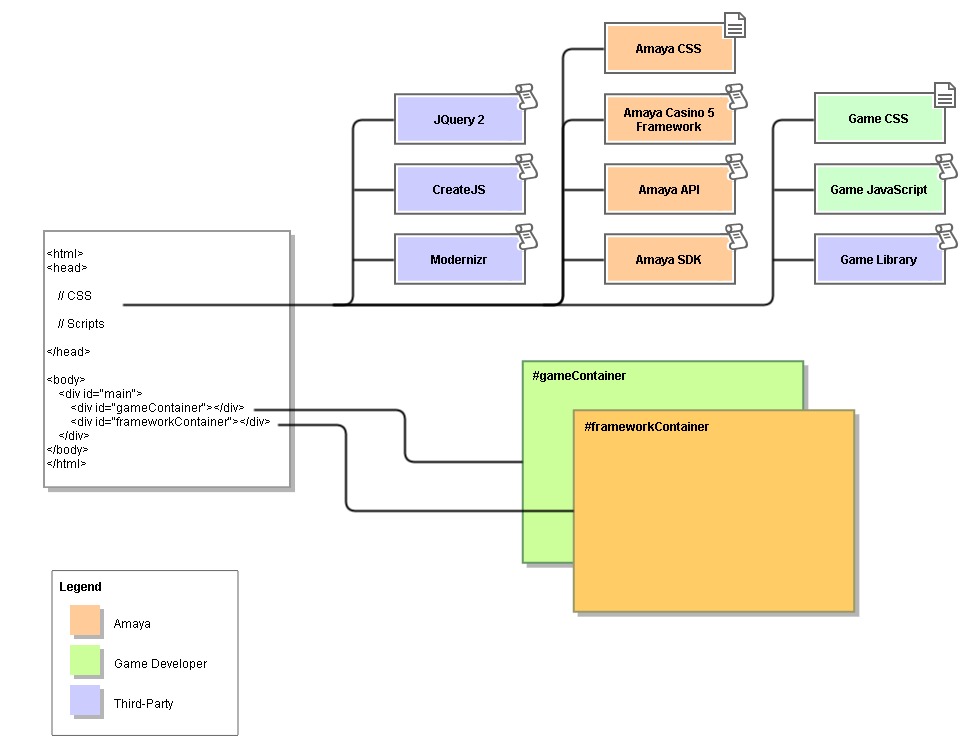
* Flash games built on the Casino 4 Framework
* HTML5 games built on the Casino 5 Framework

The Casino 4 and 5 frameworks are client-side frameworks that centralize and abstract common game functionality away from the individual game implementation. This allows game developers to focus on building the game itself without worrying about numerous other features such as managing server communication, responsible gaming and certification constraints and providing consistent user interfaces such as dialogs and errors.



## Game Launch Page

At the core of the mobile game platform is the html page that contains the JavaScript, CSS, and DOM that the game will have access to. This page must facilitate the separation of the Casino 5 framework and the game’s JavaScript and DOM without colliding. Third Party libraries such as JQuery must also be considered. To help achieve this, two separate div’s exist on the page, one for use by the Casino 5 framework and one for use by the game. These div’s are layered on top of each other with the Casino 5 framework existing in the top-most div (as it layers elements on top of the game).



## The “amaya” Namespace

In addition to the separation of the DOM, all JavaScript code for the Casino 5 framework and the libraries shared with the game are scoped to the “amaya” namespace to avoid colliding with other objects in the game code or third-party libraries.

The “amaya” namespace contains a number of objects that further partition code. Many of these are intended solely for the use framework (such as amaya.framework) and some contain services available to games (such as amaya.GameServices and amaya.sdk).

The “amaya.game” Namespace  
  
One namespace in particular is reserved for the use of the game. This is the “amaya.game” namespace. This object is created at runtime but is empty, allowing game developers to scope their code here. Each RGF game client built on the Casino 5 framework must implement a function called “Game” scoped to “amaya.game” as the constructor for their game. This is the “hand-off” point where the framework is finished actively preparing the game and the game should take control. This function can create new objects in the amaya.game namespace or can create objects in another namespace (for example, “mycompany”).

**Note:** The Casino 5 framework does NOT use the new keyword to instantiate the game. Instead it simply calls amaya.game.Game() an expects a function to be executed.

**Note:** Do not scope code to the global namespace. While this may not cause an issue immediately, it could lead to problems in the future if those objects are overwritten.

## Loading JavaScript and CSS files

All JavaScript and CSS files needed for the game are preloaded by the Casino 5 framework and dynamically added to the page at runtime. You may have as many of either files as needed.

For maximum efficiency during the loading phase, combining your javascript into a single minified file (or simply concatenating your scripts into a single file) is recommended. This also allows you to control the order of JavaScript execution as it is loaded into the page. The Casino 5 framework does not attempt to control the order of the files in any fashion.

Further information on how the framework obtains a list of which files to load is provided in “The Game Manifest” section.

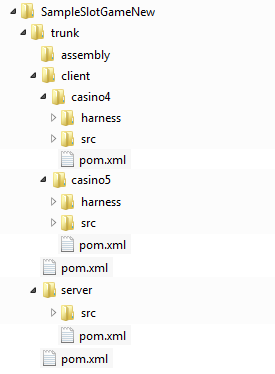
## Libraries used by the Casino 5 Framework

In an effort to provide as much freedom to game developers as possible, the Casino 5 framework uses a minimal set of libraries. The following libraries are used by the framework and are therefore available at runtime to the game.

* **JQuery 2**: Excellent library that provides DOM manipulation and AJAX calls in a simple, cross-browser format.
* **EaselJS and PreloadJS of the CreateJS suite**: These libraries provide an ActionScript-like API for working with the HTML5 Canvas. They are easy to learn and immediately usable by any developer with ActionScript 3.0 experience. EaselJS provides a retained-mode renderer that allows developers to use a hierarchical display list structure and event-model that is proven to work well for game development. While it is in not required, it is worth checking out, particularly when building a game in both ActionScript and HTML5 as the code ports very easily.
* **Modernizr:** A library that provides feature-detection in browsers. Modernizr provides an alternative to User Agent sniffing.
* **Amaya API**: The Amaya API library provides the main object used to communicate between the game and the Casino 5 framework, called “GameServices”. GameServices provides functionality such as sending messages to and receiving messages from the server, presenting dialogs, balance updates etc. Please refer to the API documentation for this library for detailed information.
* **Amaya SDK**: The Amaya SDK library provides additional objects that help when building games. For example, the V2Game object encapsulates all the required messaging each game must perform to start. It also contains classes for creating standard requests and reader classes for working with standard responses. In addition, classes are provided that assist with formatting currency, localizing games, and working with strings, numbers and JSON objects.

# The Remote Game Framework

As mentioned earlier in this document, the RGF is a system which serves games that are built as individual webapps. Each game webapp has a specific high-level structure that defines the project. A project template has been included with this documentation. The template includes a basic folder structure, a maven build for the each component (as well as a project-level reactor build), starter files and some development tools. The project looks like this:



At the top level, the project is divided into “client” and “server” directories. The client directory is further subdivided into “casino4” and “casino5” directories. The “server”, “casino4” and “casino5” each comprise their own sub-project with a maven build.

**Note:** A game does NOT have to provide both a Flash and HTML5 implementation. At a minimum, one client and the server portion must be implemented. The structure simply provides a way to develop the game for multiple client technologies. If you are not building one of the clients, do **NOT** delete the default structure that was created for it. This will likely break the build. Additional client implementations may be added at a later date.

# Casino 5 Game Project Structure

Below is an expanded view of the main directories under the Casino 5 game project. There are two main directories here, “src/main” and “harness”.

casino5  
 harness  
 src  
 main  
 assembly  
 graphics  
 javascript  
 webapp  
 casino5  
 css  
 framework  
 lib

paytable  
 postload  
 preload

high  
 high\_1.6  
 high\_1.33  
 high\_1.77

low  
 low\_1.6  
 low\_1.33  
 low\_1.77

standard  
 standard\_1.6  
 standard\_1.33  
 standard\_1.77  
 sound

“src/main” is the Maven standard location for all the files that comprise the project (source files, static files etc).

“harness” contains a separate tool used for testing the game during development. It will be explained in detail later in this doc. For now, it’s just important to know where to find it.

“assembly” contains information used by the build and should not be modified.

“graphics” contains source files such as .psd’s etc that are used to create the compressed images used by the game

“javascript” should contain all your game’s JavaScript files. Depending on how you architect your game, this could contain a single file or many sub-directories and files. The default build use the Google Closure Compiler to combine all the JavaScript files within this directory into the single file which is included on the page at runtime.

“webapp/casino5” contains all the files that should be copied to the resulting webapp. The files are not compiled or modified by the build but are simply included in the provided structure. You can and will add files and directories here as required by your game. Do not add files or folders outside of the “casino5” directory (i.e. directly under “webapp”) as other components under “client/casino4” and “server” are added here in other phases of the build and may conflict.

“css” should contain only .css files for your game that should be added to the page at runtime.

“framework” is a special directory that is only used to store files that specifically override files used by the framework. For example, this directory can be used to override the default loading screen image with a custom one for the game. However, files placed here must follow a strict naming convention. For more information, refer to the section of this document on “Overriding the Default Loading Screen Image”.

“lib” should contain only libraries your game is dependent on (for example if your game uses a library for animation, you would put it here so that it gets included on the page at runtime.

"paytable" **must** contain a paytable.html file and should contain any additional files to support it (images, additional pages etc).

“postload” contains files you would like to load during gameplay. Specifically files that should NOT be preloaded. In most games, all files can be preloaded and this directory would be empty.

“preload” contains all files that will be preloaded before your game is instantiated by the framework. As you can see, there are several subfolders. Many of these subfolders begin with a “quality” specifier and end with an aspect ratio. The "high", "low" and "standard" folders only contain the "quality" specifier and no aspect ratio. Only the directories that match the quality or the quality AND aspect ratio determined at runtime are downloaded. For detailed information, please refer to the section in this document on developing your game for multiple devices as well the game manifest section.

"sound" contains all sound files (in various formats) required for the game. This directory is only used when using the SoundSystem API from the Amaya SDK Library (See the Sound section later in this document).

## The Generated Webapp (WAR)

The top-level maven build will execute the build of each component and pack all the generated artifacts and static files into a WAR file that can then be deployed to the Remote Game Container.

The WAR file will contain the usual required directories for a WAR (WEB-INF and META-INF), additional directories added by the server portion of the game (ex: stylesheets directory) and “casino4” and “casino5” directories. As mentioned earlier, the “casino5” directory will contain the files and folders in the project’s “src/main/webapp/casino5” directory as well as a “javascript” folder containing the game’s compiled javascript resource. The webapp’s “casino5” directory will also contain a generated manifest, discussed next.

## The Game Manifest (for Casino 5 Games)

During the build, a Maven plugin is run to automatically generate a manifest.xml file. This file provides a list of all the other files within the “casino5” directory, organized based on the directory in which they are located. The file’s format is as follows:

<?xml version="1.0" encoding="ISO-8859-1"?>  
<manifest type="master" date="22-03-2013\_19:47:01">  
  
 <manifest type="preload">  
 <file>path/to/my.png</file>

... N additional <file> nodes ...  
 </manifest>  
  
 <manifest type="postload"/>  
  
 <manifest type="lib">  
 <file>lib/soundjs-0.4.0.min.js</file>  
 <file>lib/tweenjs-0.4.0.min.js</file>  
 </manifest>  
  
 <manifest type="css">  
 <file>css/game.css</file>  
 </manifest>  
  
 <manifest type="javascript">  
 <file>javascript/game.min.js</file>  
 </manifest>

<manifest type="sound" implements="m4a,mp3,ogg">

<file>sound/m4a/button.m4a</file>

<file>sound/mp3/button.mp3</file>

<file>sound/ogg/button.ogg</file>

</manifest>

<manifest type="framework">  
 <file>framework//high\_1.33/loader.png</file>

... additional <file> nodes for other quality/ratio combinations ...  
 </manifest>

</manifest>

Notice that the nested “manifest” nodes each have a type attribute (css, javascript, lib, preload and postload). Each of these manifest nodes contains 0 or more “file” nodes. Each file node contains a path relative to the “casino5” directory of the webapp.

* All files in the webapp’s “casino5/preload” directory are listed in the preload manifest. These files are loaded first and references to each file are provided to the game code at runtime via the GameServices.assets object. Only files that are required for the current device are preloaded. See the section on the developing for multiple devices for further information.
* All \*.js files in the webapp’s “casino5/lib” directory are listed in the lib manifest. These files are loaded next and inserted into the page. This ensures that all libraries are loaded before other game code.
* All \*.css files in the webapp’s “casino5/css” directory are listed in the css manifest. These files are loaded next and inserted into the page at runtime after the lib files.
* All \*.js files in the webapp’s “casino5/javascript” directory are listed in the javascript manifest. These files are loaded last and inserted into the page at runtime last.
* All files in the webapp’s “casino5/postload” directory are listed in the postload manifest. None of these files are loaded by the framework. The paths can be used by a game to load files after the game has been started if required.
* All files in the webapp's "casino5/sound" directory are listed in the sound manifest. These files are preloaded by the framework and registered as sounds with the SoundSystem API. See the section on Sound for more information
* All files in the webapp's "casino5/framework" directory are listed in the framework manifest. These files are used to override certain elements of the framework. For more information, see the section on "Overriding the Default Loading Screen Image".

## Building the WAR

In order to build the WAR, your system must first have Maven installed. In addition, you will also need to install any custom plug-ins provided. See the development environment setup instructions for complete details.

To build the entire package from the command line (server and clients):

mvn package

**Note:** If you use an IDE that supports Maven, you may be able to execute the “package” goal directly from the IDE. Amaya uses JetBrains IntelliJ which provides this functionality.

# The Casino 5 User Interface

The Casino 5 Framework contains some visual and interactive elements that are common to all games. These elements are created, positioned and managed by the framework in a layer the floats over the game.

The user interface elements include:

* Status bar
* Menu Button
* Menu
* Spin Button (Optional)
* Dialogs

This section will describe each element, the screen area it consumes and how the game can interact with it.

## Status Bar

The Status Bar is an area at the bottom of the screen that provides the user with financial and game session information.

### Display Information

The Status Bar floats over the game along the bottom edge of the game play area. It covers a small portion of the game (See the table below).

|  |  |  |
| --- | --- | --- |
| Quality | Height | Width |
| Low | 36px | 100% |
| Standard | 36px | 100% |
| High | 72px | 100% |

Games should fill the area underneath the Status Bar with some type of background. Since the Status Bar is part of the framework and not the game, its size could change in the future which may result in a display issue within the game.

### Displaying Financial Information in Cash

The financial display area displays the user's current balance, current wager and current win amount. These values are all displayed as cash in the user's currency. It is the responsibility of the game to set these values using the methods provided by the GameServices class.

* setBalanceDisplay
* setWageredDisplay
* setWonDisplay

**Note:** For more information on these methods, refer to the Amaya API documentation provided as a part of this package.

Displaying Financial Information in Credits  
  
The Status Bar also provides an option to allow the user to view the financial displays in credits (if applicable to the game). Since credits are not used by all games, games must choose to enable this feature through GameServices.

* enableCreditDisplays

Once enabled, the user can click on the financial displays to toggle between cash are credit displays.

It is important to note that the framework does not convert the cash values to credits in order to display them. This would require that the framework always be aware of the current credit value. Instead, it is the responsibility of the game to also set the credit value displays through GameServices.

* setCreditsDisplay
* setCreditsWageredDisplay
* setCreditsWonDisplay

Typically, if a game will make use of the credit displays, the "enableCreditDisplays" method is called during startup of the game. During the course of game play, whenever the cash values are updated, the credit values are updated as well. This way the fields are always correctly populated and the user is controlling which set of fields to view.

### Game Play Id

The Status Bar is also responsible for displaying the Game Play Id field. This field displays an id value that corresponds to the player's current hand/spin. The player can reference this number when contacting support if an issue occurs. The game play id field will only appear when a value is applicable. It will not be present until the first hand/spin is started (so it will not appear when the game is first loaded) and it may disappear between games.

**Important:** The GameServices API provides methods for setting and clearing the Game Play Id value. This is NOT something that most games should do. It is only provided in the API to support special integration cases. Unless you have been specifically instructed to use these methods, do NOT use them in your game.

### Menu Button

The menu button opens the menu. The menu can be disabled in which case this button does not appear. The menu button is positioned in the lower-left corner of the Status Bar.

### Game Settings Button

The game settings button opens the Game Settings panel. It is positioned in the lower-right corner of the Status Bar.

## Menu

The menu opens when the menu button is clicked. When open, the menu covers most or all of the game screen. All interaction with the game is blocked.

The menu panel displays a number of tabs (varies depending on configuration, play mode etc), the game options (sound on/off etc) and optionally may also display a clock (with current date and time) and/or a session timer displaying the length of time the user has been playing.

## Spin Button

The Spin button is a built-in button that the game can use to control initiating the game play.

### Display Information

The menu button is positioned in the upper-right corner of the screen and overlays the game content in this area. The close button that replaces the menu button when the menu is open is the same size and in the same position.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Quality | Width | Height | Distance from Right Edge | Distance from Top Edge |
| Low | 125px | 150px | 0px | 50% |
| Standard | 125px | 150px | 0px | 50% |
| High | 250px | 300px | 0px | 50% |

### Using the Spin Button

The Spin Button is hidden by default but can be turned on and off using through GameServices.

* showSpinButton
* hideSpinButton

When displayed, the button will broadcast an event through GameServices to inform the game that the button has been clicked.

* SpinButtonEvent

## Dialogs

The Casino 5 Framework provides prebuilt dialog boxes that can be used by the game to provide confirmation dialogs and Yes/No choice dialogs. These dialogs appear over the game and block user interaction with it. These dialogs can be initiated through GameServices

* showOkDialog
* showYesNoDialog

Once displayed, the user must interact with the dialog. When either the "Ok", "Yes" or "No" buttons are pressed, an event containing information about the user's selection will be sent through GameServices to the game.

* DialogResultEvent

### Multiple Dialogs

Dialogs can stacked. If a dialog is initiated when another dialog is currently waiting for a user response, the new dialog is queued. When the first dialog is acknowledged, the queued dialog will be displayed immediately.

### Canceling Dialogs

If a dialog is initiated but is no longer relevant, the dialog can be cancelled through GameServices by referencing the id assigned to the dialog when it was initiated.

* CancelDialog

### Determining if a Dialog is Currently Displayed

If the game needs to know if a dialog is currently displayed, the game can call GameServices.frameworkInfo.isModalDialogDisplayed() to return a Boolean.

In addition to directly querying the Boolean value, the framework will also dispatch a ModalDialogChangeEvent through GameServices when modal dialogs are displayer or removed. The event contains a state value indicating if a modal dialog is being displayed or removed from the screen.

## Errors

The framework provides an error screen to display fatal errors to the player. In almost all cases, displaying of errors is handled by the framework and requires no coding in the game. For example, if a game sends a request to the server but the server fails the request, the framework will display an error. The game does not need to handle the failure case and display the error itself. However, in some cases a game may need to display a fatal error that is not known to the framework. In this case, GameServices provides the "showError" method to allow the game to initiate an error.

When an error is displayed for any reason, the framework will dispatch an ErrorDisplayedEvent. This event can be used to gracefully shut down the game by stopping sounds and animations that are still running.

# Developing Games for Multiple Devices

A common problem in the “mobile” world is the numerous variations of screens sizes, resolutions, and aspect ratios. Designing content to fit every device perfectly is typically too costly to be practical. To address the problem, the Casino 5 Framework uses a combination of:

* graphics designed for common aspect ratios
* graphics designed for common screen sizes/pixel densities
* runtime calculation of the screen real estate available to the game (viewport)
* adaptive scaling

## Aspect Ratios

Analysis of the screen resolutions of numerous devices shows that most tablets and phones are exactly or very close to one of three aspect ratios; 1.33, 1.6 and 1.77. When also considering games running in a browser (on a device or on a desktop/laptop) that has visible “chrome” or that is resizable, the possibilities of aspect ratios become limitless.

When loaded, the Casino 5 Framework will first measure the viewport size and calculate its aspect ratio. It will then determine which of the three aspect ratios (1.33, 1.6, 1.77) is closest to the current viewport ratio. The closest value is set internally and used when generating canvases and determining which graphics to preload.

## Graphic Quality

While measuring the viewport size, the framework does a calculation to determine how many pixels are available (width x height in pixels). Based on this value, the framework determines what quality level of graphics to load; low, standard or high. Currently, any value lower than 500,000 is considered “low”. This value is most commonly detected on phones. Values between 500,000 and 3,000,000 are considered “standard”. This value is most commonly detected on tablets. Values larger than 3,000,000 are considered “high” and are most commonly detected on high-res ‘Retina Display’ tablets, and laptops/desktops running full-screen or maximized windows.

## Scaling and the “Stage” Size

Once the framework has determined the aspect ratio and graphic quality, it generates its graphic elements based on the following values:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Low (Phone) | Standard (Tablet) | High (‘Retina’ Tablet) |
| 1.33 | 800x600 | 1024x768 | 2048x1536 |
| 1.6 | 800x480 | 1280x800 | 2560x1600 |
| 1.77 | 854x480 | 1366x768 | 2560x1440 |

These width and height values become the “Stage Size”. That is, the framework lays out its graphic elements based on these values and then determines a scale ratio to apply to fill as much of the screen as possible without distorting the aspect ratio. This may result in black bars on either the sides or the top and bottom of the game. Even though the actual rendered area of the game may be slightly larger or smaller than the width and height of the stage, games can comfortably work with the stage width and height consistently throughout the running of the game.

Example:

|  |  |
| --- | --- |
| Viewport Size | 1050 x 800 |
| Actual Aspect Ratio | 1.3125 |
| Casino 5 Aspect Ratio | 1.33 |
| Casino 5 Graphic Quality | Standard (1050x800=840000px) |
| Casino 5 Stage Size | 1024x768 |
| Scale Factor | 1.025x |
| Actual Rendered Width | 1050 (Math.round(1024 x 1.025)) |
| Actual Rendered Height | 787 (Math.round(768 x 1.025)) |
| Black Bars | Top/Bottom, 6.5 pixels each |

## Resizing

If the user resizes their window, the rendered area resizes but the stage size does not change. Essentially, only the scaling factor applied to the stage size is changed. This is only noticeable on devices with a resizable viewport.

## Aspect Ratio + Graphic Quality

When considered together, the aspect ratio and graphic quality values allow games to download just the right amount of information to provide a quality experience for the player’s device. It also allows developers to tailor the experience to the device. Simply scaling a phone-sized game up to a 24” desktop is unlikely to provide the best possible experience. Conversely, scaling a full-size game down to a phone screen will likely result in text and buttons that are too small to be usable.

## How Casino 5 Determines What to Preload

Throughout this document, you have no doubt noticed the directories in the webapp’s “casino5/preload” directory named using the graphic quality and aspect ratio values mentioned above (ex standard\_1.33). When determining what files to preload, the framework looks at the aspect ratio and graphic quality values detected and ignores loading from the other sub-directories. For example, if the framework detects “standard” graphic quality and “1.6” aspect ratio, then it will only preload files in “casino5/preload/standard\_1.6” and any other files/directories added to the “casino5” directory.

## Falling Back When a Quality/Aspect Ratio is Not Implemented

If it is not possible or desirable to implement all combinations of qualities (low, standard, high) and aspect ratios (1.33, 1.6, 1.77), the Casino 5 framework will attempt to “fall back” to another combination automatically.

Example

Assume a game was developed in the 1.33 aspect ratio for low, standard and high qualities but not for the 1.6 and 1.77 ratios. Only three of the nine possible combinations are implemented. Continuing the example, assume that a player’s device loads the game and calculates a 1.6 ratio viewport at standard quality. The following would occur at startup:

1. The game’s manifest is downloaded.
2. The framework checks the manifest for files in the “standard\_1.6” directory.
3. Finding no files in this directory, the framework will examine the rest of the preload files, falling back to a different combination that has been implemented. In this case, the game was developed for “standard\_1.33” so the framework will use that quality and aspect ratio instead.

Similarly, if a game was developed for “standard\_1.33”, “standard\_1.6” and “standard\_1.77” but the device would ideally support “high\_1.77”, the framework would see that the 1.77 ratio could be supported but only in standard quality. Therefore it would fall back to “standard\_1.77”.

The key to taking advantage of this automatic fall back behavior is to simply leave the unsupported quality/aspect ratio directory empty. If **ANY** files exist within the directory, the framework will assume that quality/aspect ratio combination is supported. As example, if you wish to build a game for phones and tablets but do not want to support “retina display” quality graphics, you would use the “low\_1.33”, “low\_1.6”, “low\_1.77”, “standard\_1.33”, “standard\_1.6” and “standard\_1.77” directories. The “high\_1.33”, “high\_1.6” and “high\_1.77” would be empty.

Note: The directories for the unsupported quality/aspect ratio can also be completely removed as well.

## A Note (And Warning) About ‘High’ Quality and Retina Displays

While devices such as the recent iPad revisions and the Nexus 10 have very high resolution screens, the browser does not use these values directly. Instead, it maps a smaller virtual space to the real device pixels. For example, the iPad 4 has a retina display resolution of 2048x1536. However, when Safari executes JavaScript to measure the viewport, it would return a width value of 1024 (in landscape mode). Text and other scalable objects are rendered according to the actual device pixels, resulting in smoother text, but images must be handled separately.

In the case of canvases, the proper way to achieve a retina display game is to create a canvas the size of the devices resolution (2048x1536) and style it with CSS to make it fit the virtual resolution (1024x768). This provides the device with higher resolution data within the virtual space. The device will then render using that additional data, providing a sharper image.

While this produces very nice looking graphics, you will likely find that the performance of the game will suffer greatly due the much larger canvas it must repaint. It will also take significantly longer to load and require more data to be received by the end user. The slight improvement in sharpens does not typically justify the large decrease in load speed and performance.

The Casino 5 Framework was designed to allow developers the flexibility to develop games that target devices from phones all the way up to retina-enabled screens. However, most games will provide a better experience by not implementing the “high” quality resolutions supported by the framework and instead letting retina devices use the “standard” quality implementation. For more information on how the Casino 5 Framework falls back to a lower quality automatically, see the previous section “Falling Back When a Quality/Aspect Ratio is Not Implemented”.

## Taking Control Over Sizing

Depending on the needs and design of the game, there may be times you want to fully control the graphics yourself. Some scenarios might be:

* The game uses SVG graphics and can be scaled to the canvas size dynamically.
* Pre-existing code or libraries used by the game that have a conflicting method of managing the scaling.

In a case where this level of control is required, simply do not put any files in any of the quality/aspect ratio directories. For example, when using SVG, all files could be placed in the preload directory so they are always preloaded. The code from the game can then managing fitting them to the available screen space as necessary.

When none of the quality/aspect ratio folders contain any graphics, the Casino 5 framework will use the ideal resolution calculated for the device based on the viewport size. For example, if a device was determined to ideally fit “standard” quality and a “1.33” ratio, the framework would operate at 1024x768. Since the framework uses its own canvas and has bitmap graphics designed for all quality/aspect ratio combinations, it would load its own files as “standard\_1.33” and resize (if enabled) based on this setting. The game is then free to create its own canvas and manage its own graphics within the 1024x768 space.

Note: Be aware that the game will continue to be sized appropriately for the device, there simply will be no fall back action taken or any quality/aspect ratio files preloaded.

## Working with Preloaded Files

Each file listed in the preload manifest is loaded, converted to an appropriate type and indexed for lookup by the game. All preloading is handled using the PreloadJS library of the CreateJS suite.

Each preloaded file is stored based on its type. For example, if a .png file is loaded, it will be stored as HTML “IMG” element. The following shows how each file type is stored.

* Image: An <img /> element
* Audio: An <audio /> element
* JavaScript: A <script /> element
* CSS: A <link /> element
* XML: An XML DOM node
* SVG: An <object /> element
* JSON: A formatted JavaScript Object
* Text: Raw text
* Binary: The binary loaded result

Each file is added to a JavaScript object as it is loaded. Later, when the game starts up, this object is available through the “amaya.GameServices” classes using the “assets” object and its methods.

var gameServices = new GameServices ();

var myImg = gameServices.assets.getAsset(“path/to/myimage.png”);

The key that an individual file is stored under is the path to the file relative to the webapp’s casino5/preload directory, including the file name and extension.

Example:

You add a file called “myImage.png” to “webapp/casino5/preload”. It will appear in the manifest like so:

<?xml version="1.0" encoding="ISO-8859-1"?>  
<manifest type="master" date="22-03-2013\_19:47:01">  
  
 <manifest type="preload">  
 <file>preload/myImage.png</file>  
 </manifest>

</manifest>

The value of the <file> node, excluding “preload/”, is the key which the image will be stored under in the assets object. “preload/” is trimmed as a convenience since the assets object will only contain files from the preload directory.

var gameServices = new GameServices ();

var myImg = gameServices.assets.getAsset(“myimage.png”);

## Using Files Preloaded From the <Quality>\_<Aspect Ratio> Directories: The "device" Keyword

When placing files inside the quality/aspect ratio directories under “webapp/casino5/preload”, they will be manifested using this path for preloading. Often, you will place an image with the same file name under each of these directories, tailored for that specific device.

As a simple example, let’s assume you have a game logo called “logo.png” and you have created a high resolution, standard resolution and low resolution copy. You might store these files as follows:

+ webapp

+ casino5

+ preload

+ low\_1.33

+ images

logo.png

+ standard\_1.33

+ images

logo.png

+ high\_1.33

+ images

logo.png

And the resulting manifest would look like this:

<?xml version="1.0" encoding="ISO-8859-1"?>  
<manifest type="master" date="22-03-2013\_19:47:01">  
  
 <manifest type="preload">  
 <file>preload/low\_1.33/images/logo.png</file>

<file>preload/standard\_1.33/images/logo.png</file>

<file>preload/high\_1.33/images/logo.png</file>  
 </manifest>

</manifest>

At runtime, the Casino 5 Framework will determine which quality and ratio this device will use. Let’s assume that the result is “standard\_1.33”. The framework will ignore preloading the “low” and “high” quality versions of the logo.png and will load the “standard\_1.33” image.

Following the typical rules for indexing the file, the logo would be added to the assets object using the key “standard\_1.33/images/logo.png”. However, this would make coding difficult because you would need to worry about what quality and aspect ratio were in use and write conditional logic to determine which key to use when obtaining a reference to the logo from the assets object.

To prevent this, all of the special <quality>\_<aspect\_ratio> folder names are replaced with a common keyword “device”. Therefore, to get this file from the assets object, you would write:

var gameServices = new GameServices ();

var myImg = gameServices.assets.getAsset(“device/images/logo.png”);

Now when this code runs, it doesn’t matter if the game is running under “high\_1.77” or “low\_1.33”, the logo will appear to be in the same location.

If you do not have a logo under each of the quality/aspect directories, be sure to use a check to avoid an error. This might occur if a specific layout requires additional images. Continuing with the logo example, you may have a logo for standard and high quality but not for low (maybe there isn’t enough space in this layout). GameServices.assets provides a “hasAsset()” method which will tell you if the file exists.

var gameServices = new GameServices ();  
var myImg;  
if ( gameServices.assets.hasAsset(“device/images/logo.png”) ) {   
 myImg = gameServices.assets.getAsset(“device/images/logo.png”);  
}

Additionally, a try/catch can be used. The “getAsset()” method will throw an error if the file does not exist.

var gameServices = new GameServices ();

try {

var myImg = gameServices.assets.getAsset(“device/images/logo.png”);

} catch ( error ) {

// No Image

}

**Note:** Refer to the Sample Slot Game implementation provided with this package. There are several examples of using images from these special quality/aspect directories.

## Using Files Preloaded From the <Quality> Directories: The "device" Keyword continued

In some case, you will want to use the same image in all the aspect ratios for a given quality. Continuing with the "logo.png" example from the previous section, the logo for standard\_1.33, standard\_1.6 and standard\_1.77 might be the identical file in each case. Rather than copying it to each of the three directories, this file can instead be place in the "standard" directory. Any files in this directory will be loaded in addition to the ones in the quality/aspect ratio directory.

The same rules for the "device" keyword apply here.

Example:

Let’s assume you have a game logo called “logo.png” and you have created a high resolution, standard resolution and low resolution copy. Instead of storing the same file under each of the nine quality/aspect ratio, you could instead place the high resolution copy under "preload/images/high", the low resolution copy under "preload/images/low" and the standard resolution copy under "preload/images/standard".

+ webapp

+ casino5

+ preload

+ low

+ images

logo.png

+ standard

+ images

logo.png

+ high

+ images

logo.png

And the resulting manifest would look like this:

<?xml version="1.0" encoding="ISO-8859-1"?>  
<manifest type="master" date="22-03-2013\_19:47:01">  
  
 <manifest type="preload">  
 <file>preload/low/images/logo.png</file>

<file>preload/standard/images/logo.png</file>

<file>preload/high/images/logo.png</file>  
 </manifest>

</manifest>

Assuming the game loaded in standard\_1.33, the framework would download the "preload/standard/images/logo.png" file and the resulting id would be "device/images/logo.png".

## Avoiding Naming Collisions

The "quality" directory and the "quality\_ratio" directory are essentially merged when preloaded. This is done to prevent the need for a separate keyword for the "high", "low" and "standard" directories and to make it easier to promote graphics from a "quality\_ratio" directory to the "quality" directory.

If a file with same name is placed in both the "quality" and "quality\_ratio" directories, a collision will occur with the generated id.

Example:

Assume that you place files called logo.png in the following directories:

+ webapp

+ casino5

+ preload

+ standard

+ images

logo.png

+ standard\_1.33

+ images

logo.png

"preload/standard/images/logo.png" will resolve to "device/images/logo.png".

"preload/standard\_1.33/images/logo.png" will *also* resolve to "device/images/logo.png".

When this occurs, the last image to preload would effectively overwrite the first and there would be no way to get to the first image. Instead of letting this occur, the Casino 5 Framework will throw an error. This makes it easier for developers to immediately be warned of the collision and correct the issue.

## Using Preloaded Localized Files: The "lang" Keyword

When building your game, you will also need to consider how to localize text for various languages. In general, there are two high-level approaches to implementing localized text into a game: using translated strings inserted into text fields or using translated images that are inserted into the game display. In many games, a combination of both approaches will need to be used.

**Note:** This section discusses localization as it relates to preloading. For more information on localization, refer to the "Localization" section later in this document.

To use localized images, you create multiple files that have a common naming convention that is recognized by the build process when it creates the manifest for the game.

As an example, let's assume your game needs a button labeled "Bet" in English. In each other language you want to replace the button image with the word "Bet" translated into the specific language. To do this, you would create a separate image for language and name each as <myfilename>\_<two-char lang code>.<ext>

+ webapp

+ casino5

+ preload

+ low\_1.33

+ images

bet\_en.png

bet\_fr.png

bet\_de.png

bet\_es.png

...etc...

When preloading the game, we don't want to download all of these files as we only need the one applicable to the current language. To accomplish this, the plug-in that generates the manifest will recognize that the directory contains a series of files ending in "\_<lang code>" and will replace all matching files with a single entry as follows:

<?xml version="1.0" encoding="ISO-8859-1"?>  
<manifest type="master" date="22-03-2013\_19:47:01">  
  
 <manifest type="preload">  
 <file>preload/low\_1.33/images/bet\_${lang}.png</file>  
 </manifest>

</manifest>

When preloading files from the manifest, the framework will replace the ${lang} token with the current language and then request that file. So if the game is being played in English, it will download "preload/low\_1.33/images/bet\_en.png".

Once loaded the file will be added to the assets object with an id you can use to access it, just like any other file. However, indexing it with the language code in the id name would make coding the game difficult because you would have to change the id based on the current language. Therefore, the system replaces the language code in the id with the word "lang". This results in an id of "device/images/bet\_lang.png". This id will point to the localized image that was preloaded.

**Note:** In this example, the localized image exists within a quality/aspect ratio directory. Therefore, the id is manipulated twice. First, "low\_1.33/images/bet\_en.png" is transformed to "device/images/bet\_en.png". Then the id is further transformed to "device/images/bet\_lang.png" which is what would be used in the code.

In order for the localization process for images to work, all the localized images for a given case must exist within the same directory and follow the naming convention. Multiple sets of localized images can exist within the same directory (ex bet\_${lang}.png and clean\_${lang}.png can be in the same directory).

For a complete list of supported languages and further information on localizing a game, please refer the "Localization" section of this document.

## Developing Graphics for Multiple Screen Sizes

When designing your graphics, the aspect ratio and graphic quality information above is very important. You will need to design a set of graphics for each combination and program a layout that makes sense for each. There are many valid approaches but the following suggestions may be useful.

### Use JSON to Separate Layout from Code

Add a JSON file to each quality/ratio directory with property values. For example, in standard\_1.33, you may want the first reel of a slot placed at 100 in the “x” and 200 in the “y”. In standard\_1.77 where you have a wider area to work with, you may want to move the reels more towards the center, placing the first reel at 300 in the “x”.

Example:

// In “casino5/preload/standard\_1.33/mygame.json”

“reel1”: {

“x”: 100,

“y”: 200

},

// In “casino5/preload/standard\_1.77/mygame.json”

“reel1”: {

“x”: 300,

“y”: 200

}

By using the same file name in each directory and keeping a common structure within the file, your code can always refer to the same file and same objects but use different values.

**Note:** Refer to the Casino 5 implementation of the Sample Slot Game included with this package for a complete example of how JSON files are used to position elements in different quality/aspect ratios.

### Design Large and Scale Down

One technique that can be helpful when designing a game is to build graphics at the “high” quality size for a given ratio and then scale them down in Photoshop. Applications such as Photoshop offer better control over the interpolation algorithms used when scaling. The “high” quality graphics resolution is exactly or very close to two times (depending on the aspect ratio) the size of the “standard” quality resolution.

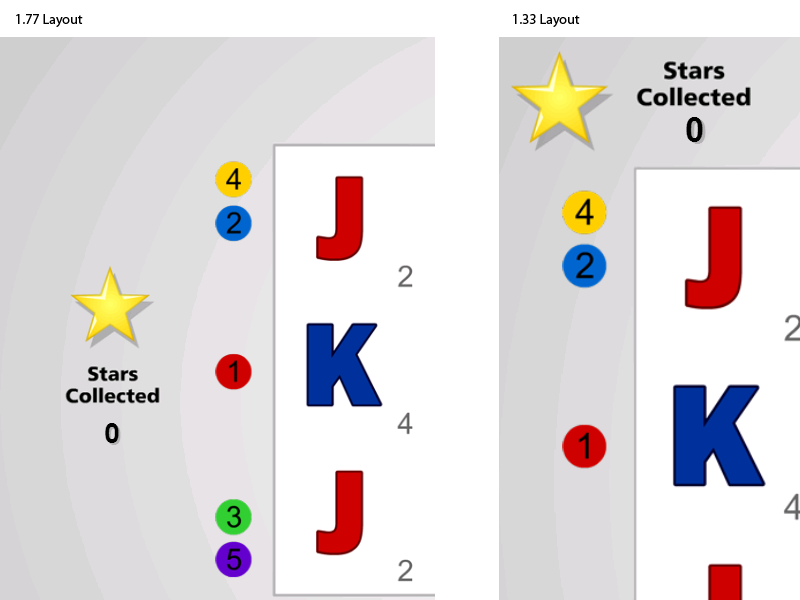
### Design for the Narrowest Ratio First

Another technique for adapting a game to different aspect ratios is to simply extend the width of the background and perhaps space out some of the elements. Design for the narrowest (1.33) size first and then simply use a wider background and different x/y values for elements such as reels and buttons.

**Note:** Refer to the Casino 5 implementation of the Sample Slot Game included with this package for an example of how the layout can be adjusted as the aspect ratio gets wider.

### Use an Alternative Layout where Required

Sometimes the same layout of elements doesn’t work well in all aspect ratios. For example, in the Sample Slot Game provided with this package, the 1.33 ratio was too narrow to fit the “Stars” counter and “Free Spin” counter on either side of the reels. Instead, in this ratio the counters were moved above the reels on either side of the logo. They were also laid out horizontally instead of vertically.



## Framerate

Unlike Flash, Javascript does not have a concept of a global framerate (or how many frames are drawn per second during an animation).

### Frames Per Second in EaselJS

The EaselJS library used by the framework provides a mechanism for setting the framerate by defining the frames per second value in the global Ticker class. Initially, the framework sets this value to 50FPS. Since the Ticker class is global, any change to it affects any animation that is reliant on it. For this reason, the Casino 5 Framework code does not use the Ticker class and instead defines its own mechanism. This leaves the createjs.Ticker class free for the game to use. If the default of 50FPS is not ideal for the game being built, simply change the value when the game loads (or as many times as needed throughout the life-cycle of the game).

### Managing the Framerate Manually

If the game being built does not use the EaselJS library (or the game does not want to use Easel's Ticker class), the framerate can be manually managed by defining its own mechanism for determining when to redraw the screen. For example, much of the graphics in the Casino 5 framework do not update frequently so instead of redrawing constantly, the framework only redraws once for each change. This greatly reduces the amount of redraws required.

# Getting Started Writing a Game

## The amaya.game.Game() Function

The provided game project template includes a Game.js file which implements a single function, “Game”, scoped to the “amaya.game” namespace. This function must be implemented by each game as it will be called to hand-off control from the framework to the game.

amaya.game.Game = function () {

// Called by Casino5 to instantiate the game

}

## Initializing a Game

At the point that the amaya.game.Game function is called, the framework has completed several things:

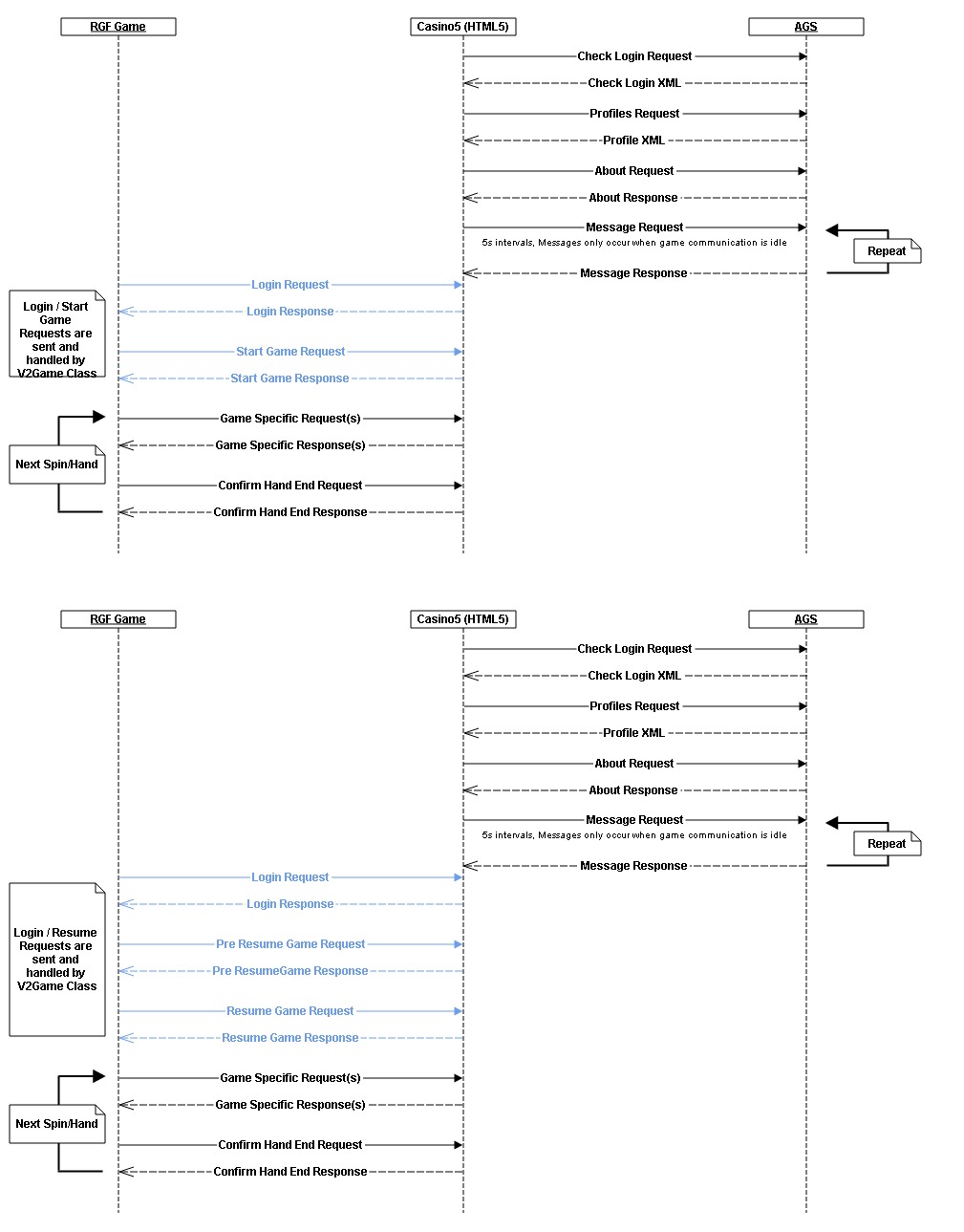
* Preloaded the framework files
* Preloaded the game files
* Performed the “CheckLogin” request to obtain account information
* Performed the “Profiles” request to obtain the profile(s) for this game (Game instance settings such as betting limits)
* Performed the "About" request to get RTP and version information
* Setup the display based on the device viewport
* Setup several internal objects to service the game

It is now up to the game to complete initialization and call the framework to remove the loading screen. To complete initialization, games must first:

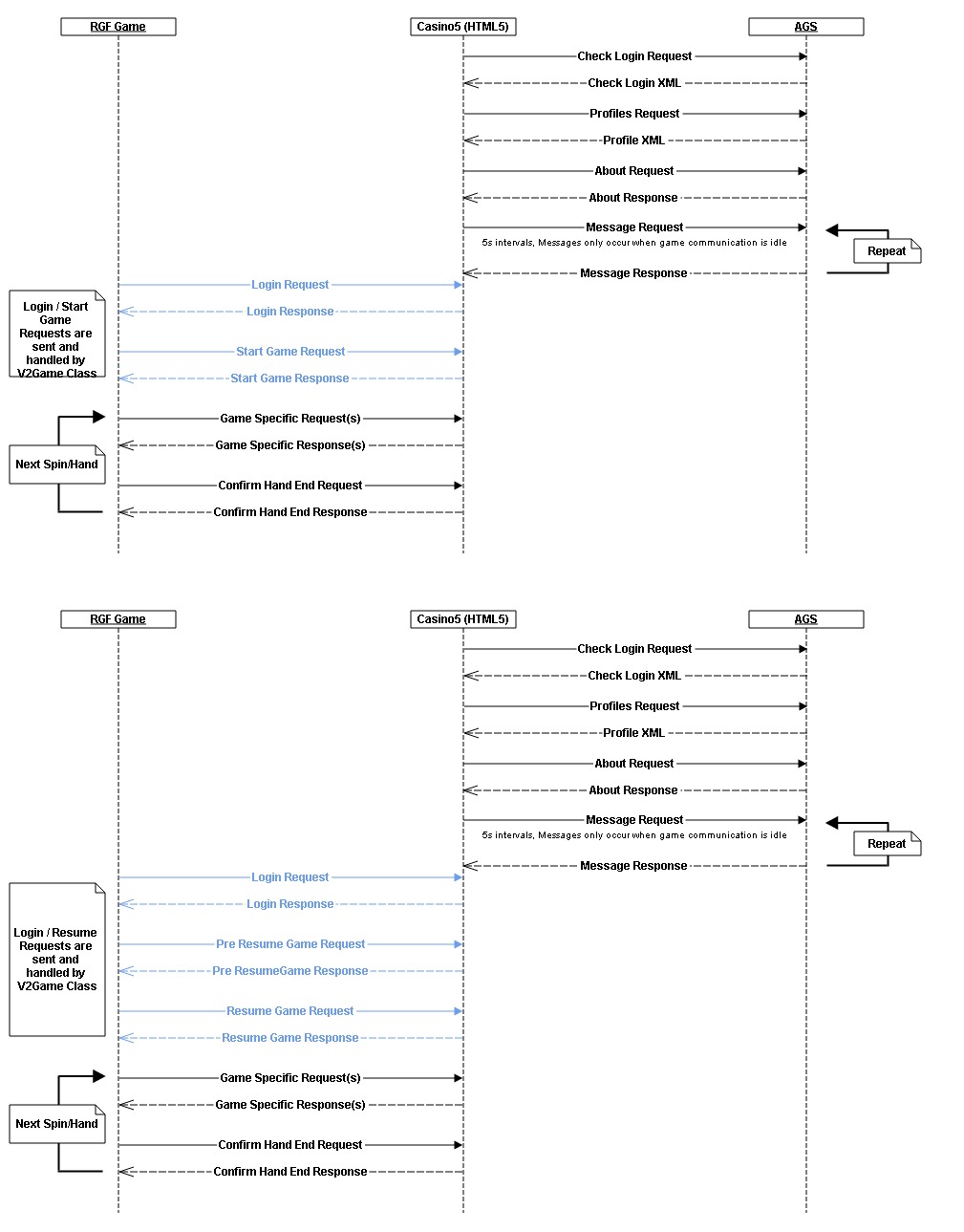
1. Complete the GameLogin request
2. Complete either the StartGame request (for new games) or ResumeGame requests (for unfinished games)
3. Initialize objects, graphics, set up listeners etc
4. Remove the loader

### Game Messaging Sequences

**New Game:**



**Resume Game:**



### Using V2Game.init() to Easily Start a Game

Steps 1 and 2 of the initialization process (Game Login request, Start Game / Resume Game request) can be completed easily by making use of the static V2Game.init() method (V2Game is a class in the SDK library and exists in the “amaya.sdk.singleplayer” namespace). Games should immediately invoke this method and pass it two function references. The first is the function to be called if a new game is started. The second is the function to be called if an unfinished game is being completed.

Example:

amaya.game.Game = function () {

// Called by Casino5 to instantiate the game

amaya.sdk.singleplayer.V2Game.init( startMain, resumeMain );

function startMain ( response, openingBalance ) {

// Callback executed when a new game is started.

}

function resumeMain ( response, openingBalance ) {

// Callback executed when an unfinished game is being resumed.

}

}

The code above satisfies the four steps required to start a game and by making use of the V2Game.init() method, does so with a minimum of code.

Note: The function names “startMain” and “resumeMain” can technically be any name but it is a best practice to use these names. The Casino 4 framework (for building Flash games) uses an identical pattern but the names are enforced in that platform.

### Setting Up Your Game

Once the game has been started and the appropriate callback invoked (startMain, resumeMain), it is time to setup any objects, listeners and graphics required for the game. This is essentially the constructor for the game. Since this differs widely from game to game, this document cannot provide a concrete example but you can refer to the SampleSlotGame implementation provided with this package to see a working example.

### Removing the Loader

Once the game has completed all its startup tasks, it is time to display it to the player. The Casino 5 framework is not able to know when the game has prepared its display, therefore it is the responsibility of the game to call removeLoader() when this should happen. The removeLoader method is a member of the GameServices object, discussed ahead.

amaya.game.Game = function () {

// Called by Casino5 to instantiate the game

var gameServices;

amaya.sdk.singleplayer.V2Game.init( startMain, resumeMain );

function startMain ( response, openingBalance ) {

// Callback executed when a new game is started.

// Handle the start of a new game here

// Initialize objects, add graphics etc.

gameServices = new amaya.GameServices();

gameServices.removeLoader();

}

function resumeMain ( response, openingBalance ) {

// Callback executed when an unfinished game is being resumed.

// Handle the completion of an unfinished game here

// Initialize objects, add graphics etc.

gameServices = new amaya.GameServices();

gameServices.removeLoader();

}

}

### Using the CanvasManager

Most HTML5 games will want to make use the Canvas object. The Amaya SDK Library, included on the page at runtime, provides a class to help with managing a simple canvas. When created, this class generates an appropriately sized canvas (based on the current aspect ratio and quality) and attaches it as a child of the “gameContainer” div on the page. A reference to the canvas can be retrieved using the “getCanvas” method.

This object also manages resizing the canvas in synch with the framework’s UI when the window is resized. It also takes into account the devicePixelRatio which allows for creating a correctly sized and styled canvas for retina display devices that have a pixel ratio greater than 1.

## Game Services

In the code example above, the amaya.GameServices object was instantiated in order to use its removeLoader method to remove the loading screen when the game was ready to be displayed. The loading screen is a part of the Casino 5 framework and is therefore not directly accessible by the game. In order to issue a command to the framework, the GameServices object is used. It provides a number of methods to perform various tasks such as sending a message to the server, displaying a dialog box with an “OK” button, or displaying the error screen.

In addition to sending commands to the framework, an instance of GameServices also dispatches events from the framework. Events are the frameworks way of sending messages to the game. GameServices will dispatch many different types events, for example when a response arrives from the server or the user’s balance changes.

Finally, GameServices also provides a number of objects containing properties about the game and the current environment such as the user’s language, the game’s profile information etc.

An instance of the GameServices object essentially wraps a number of singletons and provides a convenient way to access them without having to pass object instances throughout your code. If a class needs to communicate with the framework, simply create a new instance. Assign any listeners and use the properties and methods as needed.

Continuing with previous example, the following code uses the “assets” object to retrieve a preloaded image and then assigns a listener for messages from the server.

amaya.game.Game = function () {

// Called by Casino5 to instantiate the game

var gameServices;

amaya.sdk.singleplayer.V2Game.init( startMain, resumeMain );

function startMain ( response, openingBalance ) {

// Handle the start of a new game here

// Initialize objects, add graphics etc.

main();

}

function resumeMain ( response, openingBalance ) {

// Handle the completion of an unfinished game here

main();

}

function main () {

// Initialize objects, add graphics etc.

gameServices = new amaya.GameServices();

var img = gameServices.assets.getAsset(“background.jpg”);

// Do something with the image here.

gameServices.addEventListener(

amaya.api.events.IncomingMessageEvent.INCOMING\_MESSAGE,

onIncomingMessage

);

gameServices.removeLoader();

}

function onIncomingMessage ( event ) {

switch ( event.data.name ) {

case “MySpinResponse”:

// handle message as a spin response message

break;

}

}

}

GameServices provides a number of methods, properties and events that are valuable to game developers. Refer to the API documentation for a complete list.

## Building the Game

Once you are satisfied with the code, you will need to package the game for deployment. The entire project (Server, Clients) is all build in a single webapp archive (WAR) file that is deployed to the Remote Game Container (RGC).

For the Casino 5 Client, the process of the supplied build is as follows:

* Copy or compile JavaScript sources to the WAR
* Copy all files under src/main/webapp/casino5 to the WAR
* Create a manifest of all the files included in the package

### Build Options

The first step specifies copying or compiling the JavaScript sources. The default Maven build provided with the game template contains three build profiles of which you can use one depending on how you want the JavaScript sources handled.

**Source.GCC Profile (Default):** This profile compiles all the \*.js files from src/main/javascript into a single minified file using the Google Closure Compiler (GCC). By default there is no guaranteed order to how the GCC orders the files. However, using the goog.require() and goog.provide() calls at the top of each file, it is simple to assign a dependency-based order. For more information, refer to the Google Closure Compiler section of this document which talks about using the compiler.

**Source.Concat Profile**: This profile concatenates all the files under src/main/javascript into a single file under the webapp’s casino5/javascript directory. No removal of whitespace etc is performed.

**Note**: Using this option means that you will only have very minimal control over the order your files are loaded and inserted into the page. This profile will always ensure that a src/main/javascript/Namespace.js file is loaded first. No order of other files is guaranteed. If the load order of the files matters, you are better off using the Google Closure Compiler option or implementing your own system to manage the order.

**Source.Full Profile**: This profile simply copies all the \*.js files from src/main/javascript to the webapp’s casino5/javascript directory. Each file is individually manifested and at runtime, the files are loaded and inserted into the page one at a time.

**Note**: Using this option means that you will not have control over the order your files are loaded and inserted into the page. If the load order of the files matters, you are better off using the Google Closure Compiler option or implementing your own system to manage the order.

To change which profile is used, do the following:

1. Locate the pom.xml file in the root of the trunk/client/casino5 directory and open it for editing.
2. Near the bottom of the file, look for the build profiles tag, <profiles>. It will contain three <profile> node, each with an <id> tag named either Source.Full, Source.Concat or Source.GCC. Each profile will also contain a node called <activation>.
3. To enable a profile, change the value of it’s <activateByDefault> tag to true.
4. Ensure the other profiles are have their <activateByDefault> tag set to false.

**Note**: Only one of the three build profiles should be active at any given time.

# Using the Client Development Harness

The purpose of the Flash Client Development Harness is to allow Flash client developers to perform high-frequency testing cycles of the client during development. This is accomplished by building the harness in one of two ways (depending on your needs).

* **Standalone**: This mode compiles the game and runs it in the browser. All server responses are read from local xml files. This allows the client to run purely on the local machine.

Standalone mode is frequently used during the early stages of development when the server portion of the game has not yet been developed to a point that allows it to service requests. As long as the XML format of the messages has been determined, a client developer can mock the responses in XML files and change the values as needed. This allows the client to begin handling real game flow early while rapidly testing changes.

* **Live**: This mode compiles the game and runs it in the browser. All server communication is sent to a specified server that is running the game. This allows the client to run using local client files (as opposed to those on the server) while still making use of a real server's messaging.

Live mode is typically used when the server portion of the game is completed to a point that allows it to service the necessary requests. The client developer can run his local changes against the live server without needing to upload the changes. This allows rapid testing of changes and allows the server to run a stable version of the client for others to use (QA, Server developers etc).

Live mode requires a valid playerHandle value. This value can either be specified in the live.properties file before compiling or added as a query string value to the harness.html page (harness.html?playerHandle=1234567890). To obtain a valid playerHandle for live mode there are two options, both of which utilize the server's SDK API.

**Option 1:** Use the following url in a browser

<protocol>://<server>/sdk/servlet/com.chartwelltechnology.icd.ClientLoginSupportServlet?method=login&uid=<player name>&account=<account>&country=US&lsdId=zero&xmlresponse=true&bal=<balance>

Substitute the tokens in the url with valid values:

protocol: http or https

server: The server's domain such as opal.chartwelltech.com

player name: Any alphanumeric string. If the player specified does not exist it will be created.

account: 3-character account value such as USD, GBP, CAD etc

balance: The amount of money to add to the player's account. This value is optional.

**Option 2:** Use the playerhandle.air application

A GUI has been developed that can be installed and run to provide a form-based interface for creating playerHandles.

The "tools" directory of the Game Development Platform contains an AIR application called playerhandle.air which can be installed and run to provide a form-based interface for generating playerHandles.

**Note:** For complete documentation on running the Client Development Harness, refer to the client/casino5/harness/README.txt file that is created as part of the game project template.

## Tomcat

The harness builds either the standalone or live builds into an instance of the included Tomcat webserver. Each is built as a separate webapp and can therefore exist and run at the same time, each from a separate URL. The webserver is automatically deployed the first time a harness build is executed. Once deployed, it can be started and stopped using the supplied scripts.

**Note:** The harness for Casino 4 and Casino 5 are very similar but are not identical due to the differences between Flash and HTML5. Each harness has its own Tomcat instance that is run independently. The casino4 harness runs Tomcat on port 7777 while the casino5 harness runs Tomcat on port 9999.

The harness contains two Ant build scripts and four properties files. The main build script, build.xml, contains all the targets required to build, package and deploy the harness. The second build script, webserver.xml, contains the targets used to start and stop the Tomcat webserver. The included properties files contain values that can be customized.

For complete documentation, refer to the README.txt file in the root of the harness directory (The Flash-based harness under “casino4” and the HTML5-based harness under “casino5” in the project).

# Overriding the Default Loading Screen Image

## Loader Image

Game developers can replace the default loading screen background image with an image of their own. To do so, the game must simply provide loading screen images for the various aspect ratio/quality combinations in the correct location.

To override the loading screen image, you can use one of the following methods. Place files in the specified location. If using localized images, replace the <lang> token in each path listed below with the 2-char language code of the text contained within the image.

### JPG Files

|  |  |
| --- | --- |
| Path | Size |
| src/main/webapp/casino5/framework/high\_1.6/loader.jpg | 2560x1600 |
| src/main/webapp/casino5/framework/high\_1.33/loader. jpg | 2048x1536 |
| src/main/webapp/casino5/framework/high\_1.77/loader. jpg | 2560x1440 |
| src/main/webapp/casino5/framework/low\_1.6/loader. jpg | 800x480 |
| src/main/webapp/casino5/framework/low\_1.33/loader. jpg | 800x600 |
| src/main/webapp/casino5/framework/low\_1.77/loader. jpg | 854x480 |
| src/main/webapp/casino5/framework/standard\_1.6/loader. jpg | 1280x800 |
| src/main/webapp/casino5/framework/standard\_1.33/loader. jpg | 1024x768 |
| src/main/webapp/casino5/framework/standard\_1.77/loader. jpg | 1366x768 |

### Localized JPG Files

Use one image per language per quality/ratio combination.

|  |  |
| --- | --- |
| Path | Size |
| src/main/webapp/casino5/framework/high\_1.6/loader\_<lang>.jpg | 2560x1600 |
| src/main/webapp/casino5/framework/high\_1.33/loader\_<lang>. jpg | 2048x1536 |
| src/main/webapp/casino5/framework/high\_1.77/loader\_<lang>. jpg | 2560x1440 |
| src/main/webapp/casino5/framework/low\_1.6/loader\_<lang>. jpg | 800x480 |
| src/main/webapp/casino5/framework/low\_1.33/loader\_<lang>. jpg | 800x600 |
| src/main/webapp/casino5/framework/low\_1.77/loader\_<lang>. jpg | 854x480 |
| src/main/webapp/casino5/framework/standard\_1.6/loader\_<lang>. jpg | 1280x800 |
| src/main/webapp/casino5/framework/standard\_1.33/loader\_<lang>. jpg | 1024x768 |
| src/main/webapp/casino5/framework/standard\_1.77/loader\_<lang>. jpg | 1366x768 |

### PNG Files

|  |  |
| --- | --- |
| Path | Size |
| src/main/webapp/casino5/framework/high\_1.6/loader.png | 2560x1600 |
| src/main/webapp/casino5/framework/high\_1.33/loader.png | 2048x1536 |
| src/main/webapp/casino5/framework/high\_1.77/loader.png | 2560x1440 |
| src/main/webapp/casino5/framework/low\_1.6/loader.png | 800x480 |
| src/main/webapp/casino5/framework/low\_1.33/loader.png | 800x600 |
| src/main/webapp/casino5/framework/low\_1.77/loader.png | 854x480 |
| src/main/webapp/casino5/framework/standard\_1.6/loader.png | 1280x800 |
| src/main/webapp/casino5/framework/standard\_1.33/loader.png | 1024x768 |
| src/main/webapp/casino5/framework/standard\_1.77/loader.png | 1366x768 |

### Localized PNG Files

Use one image per language per quality/ratio combination.

|  |  |
| --- | --- |
| Path | Size |
| src/main/webapp/casino5/framework/high\_1.6/loader\_<lang>.png | 2560x1600 |
| src/main/webapp/casino5/framework/high\_1.33/loader\_<lang>. png | 2048x1536 |
| src/main/webapp/casino5/framework/high\_1.77/loader\_<lang>. png | 2560x1440 |
| src/main/webapp/casino5/framework/low\_1.6/loader\_<lang>. png | 800x480 |
| src/main/webapp/casino5/framework/low\_1.33/loader\_<lang>. png | 800x600 |
| src/main/webapp/casino5/framework/low\_1.77/loader\_<lang>. png | 854x480 |
| src/main/webapp/casino5/framework/standard\_1.6/loader\_<lang>. png | 1280x800 |
| src/main/webapp/casino5/framework/standard\_1.33/loader\_<lang>. png | 1024x768 |
| src/main/webapp/casino5/framework/standard\_1.77/loader\_<lang>. png | 1366x768 |

**Note:** These images should note contain any transparency as their intention is to hide the loading process of the game underneath.

**Note:** The framework will still overlay the loading message and progress on the image at runtime. Be sure the design of the image allows for this.

## Loader Text

During the loading process, the framework overlays text on the loader image to display the loading progress. For example, when loading a new game, you will see a message like "<Game X> is loading" with a percentage amount below it. This text varies in length based on the game name, the current language and whether a new game or an unfinished game is being launched.

When setting a custom loader image, it is sometimes necessary to change the color and position of this text. The default text is white and is horizontally centered. It is a vertically positioned a bit below the vertical center. The white color and position may not suit the custom background.

In this case, a json file can be used to override the color, x, y and alpha values of the message and the percentage fields. The json file must be called "loader.json" and place beside the custom loader image. This means you would typically have a json file for each quality/ratio combination (since the positioning will be different for each).

The loader text field is 400px wide and is usually no more than two lines of text (Game name dependent). The X position in the json file refers to the center of the field (i.e. Setting it to 0 would result in half the text being cut off the left side of the screen). The Y position in the json file refers to the top of the field.

**Note:** If the default loader text properties do not need to be overriden, do not include a loader.json file

### JSON File Locations

|  |
| --- |
| Path |
| src/main/webapp/casino5/framework/high\_1.6/loader.json |
| src/main/webapp/casino5/framework/high\_1.33/loader.json |
| src/main/webapp/casino5/framework/high\_1.77/loader.json |
| src/main/webapp/casino5/framework/low\_1.6/loader.json |
| src/main/webapp/casino5/framework/low\_1.33/loader.json |
| src/main/webapp/casino5/framework/low\_1.77/loader.json |
| src/main/webapp/casino5/framework/standard\_1.6/loader.json |
| src/main/webapp/casino5/framework/standard\_1.33/loader.json |
| src/main/webapp/casino5/framework/standard\_1.77/loader.json |

### JSON File Contents

In order to override the framework's properties, the following objects must be used as written here. Properties that are not to be overriden can be omitted. For example, to leave the message color as white, do not specify a tfMessage.color. Entire objects can also be safely omitted.

{

"tfMessage": {

"color": "#003366",

"x": 400,

"y": 300,

"alpha": 1

},

"tfMessageShadow": {

"color": "#FFF",

"x": 403,

"y": 303,

"alpha": "0.5"

},

"tfProgress": {

"color": "#003366",

"x": 400,

"y": 350,

"alpha": 1

},

"tfProgressShadow": {

"color": "#FFF",

"x": 403,

"y": 353,

"alpha": "0.5"

}

}

# Using the Credit Displays

The Casino 5 framework supports a user-selectable system for displaying key financial information represented as cash or credits. The status bar at the bottom of the game will always display this information one format or the other. The default is to display the "balance", "wagered" and "won" amounts as cash values. The same information can be displayed as credits if the game implements the following:

1. Call gameServices.enableCreditDisplays() during game startup. When called, this function enables a clickable area over the balance/wagered/won fields. When the user clicks this area, it will toggle between cash and credit views.

2. Use the "setCreditsDisplay", "setCreditsWageredDisplay" and "setCreditsWonDisplay" methods of the GameServices object to populate the values with the correct amounts. These methods are directly analogous to the "setBalanceDisplay", "setWageredDisplay", and "setWonDisplay" methods which populate the cash fields. The correct way to implement this is to always set both the cash and credits displays at the same time. Only one is visible at a time but both should always set so that the amounts are correct when the user toggles the display.

Example:

The player is currently playing a slot game and has selected to use a $0.05 credit value. Suppose the game receives a balance indicating the player's balance is now $100.00. The game would make these calls:

gameServices.setBalanceDisplay(100); // Displays $100.00 in the Balance field  
gameServices.setCreditsDisplay(100/0.05); // Displays 2000 in the credits field

Now suppose the user changes the credit value to $0.25. The game would update the fields with these calls:

gameServices.setBalanceDisplay(100); // Displays $100.00 in the Balance field  
gameServices.setCreditsDisplay(100/0.25); // Displays 400 in the credits field

The same process would apply to setting the wagered and won amounts. Set both fields simultaneously when the values need updating.

# Game Settings

The Casino 5 Framework provides a prebuilt set of controls for settings the wagers within a game. The goal of removing these settings from the game itself is to provide a consistent user experience and to decrease development effort.

Controls are created through GameServices and are included within a panel that appears over the game.

At a high level, the process of using the Game Settings controls is:

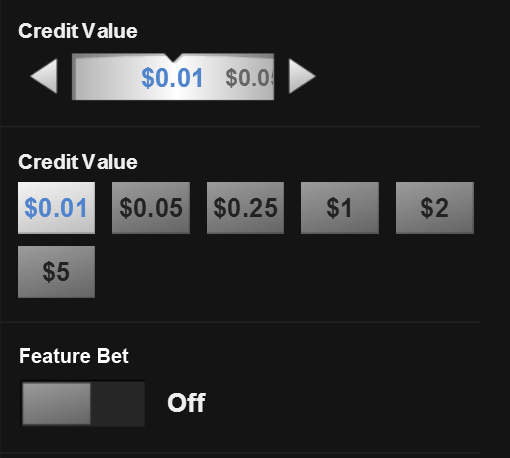
1. Create a control using the GameServices.gameSettings API
2. Configure the control with the range of values and default value required
3. Listen for events indicating the control has changed its value
4. Optionally update the control as needed

## Control Types

Currently, the Casino 5 Framework supports four types of controls:

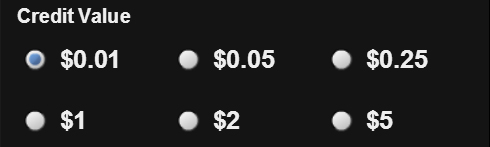
### Stepper Control

A stepper control allows the user to select a single value from a range of values. It requires an array of values and a default value. The label of the control can be specified. Values can optionally be formatted as monetary amounts.



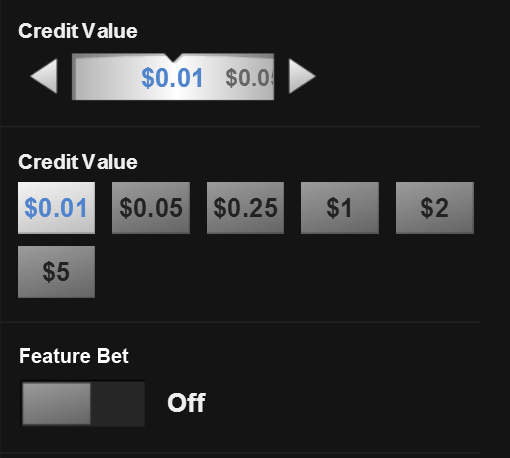
### Radio Group

A radio group control lays out a radio button for each value in a provided array of values. Only one value is active at a time. The default value and label can both be specified. Values can optionally be formatted as monetary amounts.



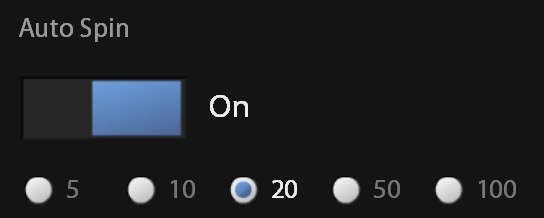
### Boolean

A Boolean control is a simple On/Off toggle. The default value can specified.



### Auto Spin

An Auto Spin control allows the user to select a pre-defined number of automatic spins. This control is first enabled by toggling the on/off switch. Once enabled, the number of spins can be selected (default is five spins).



Important: The Auto Spin control does not automatically provide a game with auto spin capability. The framework does not have enough information to be able to control the actual spinning of the reels. This control is always used in conjunction with the GameServices.autoSpin API and code included in the game. For complete information, refer to the Auto Spin section of this document.

## The gameSettings Object

Game Services provides an object called "gameSettings" which contains an API for creating controls and setting their values.

### Creating a Control

To create a control, use the "add" methods of gameSettings: addStepperControl, addRadioGroupControl or addBooleanControl.

var gs = new amaya.GameServices();

id = "myControl";

title = "My Control";

values = [10,20,30,40,50];

defaultValue = 20;

formatAsMoney = false;

gs.addStepperControl( id, title, values, defaultValue, formatAsMoney );

### Listening For a Control Change

To listen for a control change, add an event listener to GameServices for the GameSettingChangeEvent. The event will be dispatched when a control changes its values and will include the id of the control, the new value and the old value.

var gs = new amaya.GameServices();

gs.addStepperControl( "myControl", "My Control", [10,20,30,40,50], 20, false );

gs.addEventListener( amaya.api.events.GameSettingChangeEvent.CHANGE, onChange );

function onChange ( event ) {

switch ( event.data.id ) {

case "myControl":

console.log(event.data.newValue, event.data.oldValue);

break;

}

}

### Setting the Control Value

Typically, you will not often need to force the control to a specific value after it has been created. However, there are two scenarios where games will need to do this.

First, the control may need to be set if a change should be disallowed. For example, if a player changes a bet control to an amount that is larger than their current balance, the game may want to disallow the change. To do this, simply catch the problem inside the GameSettingChangeEvent handler and reset the control to the event.data.oldValue value.

var gs = new amaya.GameServices();

gs.addStepperControl( "myControl", "My Control", [10,20,30,40,50], 20, false );

gs.addEventListener( amaya.api.events.GameSettingChangeEvent.CHANGE, onChange );

function onChange ( event ) {

switch ( event.data.id ) {

case "myControl":

// Disallow the change

gs.gameSettings.setGameSetting( "myControl", event.data.oldValue );

break;

}

}

Second, when resuming an unfinished game, you may need to force controls to the values that were set when the game was interrupted.

gs.gameSettings.setGameSetting( "myControl", valueFromPreviousSession );

# The Spin Button

The Casino 5 Framework provides a Spin button which games can use to help provide a more consistent user experience.

The spin button is displayed when a game wants to use the standard Spin control. The action taken when the spin button is pressed must be implemented by the game.

There are two methods and two events associated with the spin button:

* gameServices.showSpinButton() : Displays and enables the spin button on the screen
* gameServices.hideSpinButton() : Removes the spin button from the screen
* SpinButtonEvent: Dispatched by Game Services when the user clicks the spin button. The game should listen for this event initiate the game play.

# Auto Spin

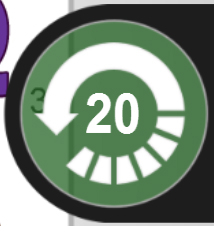
The Casino 5 Framework provides a control and state management for incorporating auto spin functionality into a game. There are three main components that work in tandem to provide this functionality; the Auto Spin Game Settings control, the GameServices.autoSpin API and the game code itself. The framework provides the first two components.

## Auto Spin Flow

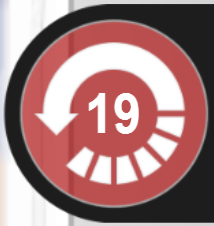
1. The game begins by creating an Auto Spin control using the GameServices.gameSettings.addAutoSpinControl method. This adds the auto spin UI to the game settings panel.



1. The user opens the game settings panel and chooses to enable auto spin by setting the on/off switch to the "on" position. This enables the radio group which allows the user to select the number of spins (default is five).
2. The user can now see that the Spin button is changed to the Auto Spin Start button displaying the number of spins selected.



1. The user clicks the auto spin button to start the auto spin series. The framework sends the AutoSpinStartEvent to the game.
2. The game receives the AutoSpinStartEvent and initiates the first spin by starting the spin animation and then calling the GameServices.autoSpin.next() method. The "next" method instructs the framework to decrease the number of auto spins remaining by one. The auto spin button changes to the auto spin "stop" button displaying the remaining number of spins.



1. Once the first auto spin is completed, the game can check if additional auto spins remain by calling GameServices.autoSpin.hasSpins() which returns a boolean. If true, there are more auto spins to complete. The game should then pause momentarily to display the results of the spin to the user and move on to the next auto spin (essentially repeat step 5 by starting a spin and then calling GameServices.autoSpin.next() to decrement the number of auto spins remaining).
2. Auto spinning continues until:
   1. GameServices.autoSpin.hasSpins() returns false. In this scenario, the game should simply re-enable the normal betting UI and spin button. GameServices will also dispatch the AutoSpinCompleteEvent when the auto spin counter is decremented to zero. This occurs immediately after calling GameServices.autoSpin.next() for the last time.
   2. A bonus round is triggered. Bonus rounds, including free spins, should cancel the remaining auto spins. This is done by calling GameServices.autoSpin.cancel(). When called, the AutoSpinStopEvent is dispatched by GameServices.
   3. The user presses the Auto Spin Stop button. When the user presses this button, the AutoSpinStopEvent is dispatched by GameServices.

## Code Required in the Game to Support Auto Spin

In general, games will need to implement code similar to the following in order to properly implement auto spins. Keep in mind that this code is not exact (i.e. do not simply copy/paste) due to the fact that every game is coded different and has different game-specific considerations.

1. Add the Auto Spin control

// Create an autoplay control  
gameServices.gameSettings.addAutoSpinControl( "MyAutoSpinControlID", "Auto Spin" );

2. Add listeners to GameServices for important auto spin events

gameServices.addEventListener( amaya.api.events.AutoSpinStartEvent.START, onAutoSpinStart );

// Optional

gameServices.addEventListener( amaya.api.events.AutoSpinStopEvent.STOP, onAutoSpinStop );  
gameServices.addEventListener( amaya.api.events.AutoSpinCompleteEvent.COMPLETE, onAutoSpinComplete );

3. Add the AutoSpinStartEvent handler

function onAutoSpinStart ( event ) {

// Typically this handler will do two things:  
 // 1. Call the same function that pressing the regular spin button does (ie. start a new spin)

mySpinFunction();

// 2. Call the GameServices.autoSpin.next() function to decrement the counter.

gameServices.autoSpin.next();

}

4. Add code to the game's "end of spin" functionality to determine if additional auto spins are available. Often a good place to do this is in the handler for the confirmHandEndRequest.

function handleConfirmHandEndRequest () {

// ... other handler code ...

// Check if more auto spins exist  
 if (gameServices.autoSpin.hasSpins()) {

// Set a short delay to let the player see the results.

// Use a slightly longer delay if the spin had payline wins.  
 setTimeout(autoSpin,(model.spinHasPaylineWins()) ? 1500 : 200 );  
 } else {

// Enable the game's regular (non-auto-spin) UI.  
 gameServices.showSpinButton();  
 }  
}

5. Add code to cancel the auto spin in the event of a bonus round or free spins being triggered.

function onSpinComplete () {

// ... other spin complete code ...

// Check if this spin has triggered a bonus round or free spins

if ( model.getFreeSpinsTriggered() ) {

// If the auto spin system has spins remaining, they must be cancelled.

if ( gameServices.autoSpin.hasSpins() ) {

gameServices.autoSpin.cancel();

}

// ... other bonus round / free spin triggering code

}

}

## General Auto Spin Notes

* Always call GameServices.notifyGameStart() before GameServices.autoSpin.next(). Calling notifyGameStart places the framework in the correct state before decrementing the counter
* Handlers for AutoSpinStopEvent and AutoSpinCompleteEvent are not strictly required but are provided in case the game needs to do additional tasks at these points.
* Call GameServices.autoSpin.next() just after the spin starts. This decrements the counter as the spin is starting. During the spin, the player will see a number indicating how many spins remain \*after\* this spin completes.
* The framework provides a configuration to disable auto-pay features as certain jurisdictions disallow players from using them. Typically, it is the game's responsibility to check this setting (GameServices.session.gameInfo. autoPlayEnabled). In the case of auto spin, it is not necessary to check this value before creating the control. Instead, simply code the game to always add the auto spin control and the framework will silently ignore it if auto play features are disabled.
* Games can only have a single auto spin control. Attempts to add additional auto spin controls will be silently ignored by the framework.

# Creating a Paytable for the Game

The Casino 5 Framework provides a menu item within the game menu that loads a paytable.html file into an iFrame.

The framework will always load the paytable from a game's casino5/paytable/paytable.html file. A default placeholder file is automatically generated by the game template at this location.

## Parameters

The paytable.html file is sent the following values on the query string. These can be used to customize the page.

* gameId: The gameId of the game being played
* lang: The 2-char language code in use by the player
* quality: The quality setting (low, standard, high) the game is currently running under
* ratio: The aspect ratio (1.33, 1.6, 1.77) the game is currently running in

## Developing a Paytable

A paytable can be as simple as a single-page document used by all quality/ratio combinations or it can be a more complex multi-page "mini-site" with navigation and customizations based on the parameters passed to it. The only requirement is that the main page be the paytable.html file. Any additional files, images or libraries should be included within the paytable directory.

It is recommended to keep the paytable as simple and light as possible. A good practice is to use responsive design so that the a paytable elements reflow based on the size of the viewport (The size of the iframe will be different depending on the quality and ratio settings).

# Sound

Implementing sound in HTML5 is currently a challenging task given the differences between browsers and the restrictions imposed by mobile OS's. This section will discuss the options available to developers.

## HTML5 Audio Limitations

These are some, but not necessarily all, of the known limitations of audio in HTML5.  
  
**Safari**

* Safari requires Quicktime to be installed for audio playback.

**iOS6**

* Sound is initially muted and will only unmute through play being called inside a user initiated event (touch/click).

**Android**

* Audio can only be played inside a user event (touch/click).

**Android Chrome**

* Can only play 1 sound at a time.
* Sound is not cached
* There is a delay before a sound is played, presumably while the src is loaded.

## Sound Formats

Currently, there is no consensus among browsers on a single audio file format. This means that in order to provide audio to as many browsers as possible, the sound files for your game will need to be provided in multiple formats. Amaya requires that games provide the following formats:

|  |  |
| --- | --- |
| Format/Codec | Extension |
| OGG Vorbis | .ogg |
| AAC (Lossy) | .m4a |
| MP3 | .mp3 |

## Using the SoundSystem API

In order to make playback of basic sounds as simple as possible, the Amaya SDK library provides a class called SoundSystem that you can use in your game. Usage of the SoundSystem API imposes the following requirements on the game:

* Sounds are encoded in OGG Vorbis, AAC (Lossy), and MP3
* Sounds are located in the games webapp/casino5/sound directory
* Sounds are named following the naming convention prescribed in this section

### Encoding

There are many free and commercial encoders available to encode sound files into the various formats required by the game. There are no hard and fast rules about settings for the encoding. Instead, use your best judgment to balance the quality of the encoded sound with the resulting file size. Keep the sounds as small as possible while still maintaining an acceptable level of audio fidelity.

Encode your sounds into each of the required formats (OGG Vorbis, AAC (Lossy) and MP3).

### Naming and Storing the Files

Each sound file that is encoded must be given a common name that will be used as the sound's identifier value later on.   
  
As an example, assume there is a sound to be played during the reel spin of a slot machine. You would encode the sound and name each resulting file as follows:

* reelspin.ogg
* reelspin.m4a
* reelspin.mp3

Be sure to name each file exactly the same, including the case of the letters. The extension must be all lowercase.

In order to correctly preload and register the sound files, they must be named and located in a specific location within the games webapp.

|  |  |  |
| --- | --- | --- |
| Format/Codec | Location | Filename |
| OGG Vorbis | webapp/casino5/sound | <sound\_id>.ogg |
| AAC (Lossy) | webapp/casino5/sound | <sound\_id>.m4a |
| MP3 | webapp/casino5/sound | <sound\_id>.mp3 |

Using the reel spin example, the final paths would be:

* webapp/casino5/sound /reelspin.ogg
* webapp/casino5/sound /reelspin.m4a
* webapp/casino5/sound /reelspin.mp3

The unique identifier for the sound would be **reelspin**. This represents the portion of the path following the sound directory, excluding the extension.

### Preloading the Sounds

Provided that you follow the naming convention and locate the files in the correct place, the Casino 5 framework will automatically preload the sound files required for the given device. It will NOT preload all the sounds. Instead, it determines which formats the current browser/device supports and preloads only the sounds in that format. For example, if the browser is Mobile Safari on iOS, the sounds in webapp/casino5/sound/m4a (AAC encoded) would be loaded.  
  
In the event that a browser supports more than one format, only one format is selected according to a predefined order of preference. That order is M4A, then OGG and finally MP3.

As each sound is loaded, it is registered with the sound system using the unique id of the sound as determined by its name and path.

### The SoundSystem Object

In order to work with sounds, the game code will create instances of the amaya.sdk.sound.SoundSystem object. This object is instantiated using the "new" keyword. Internally, the SoundSystem object is backed by a singleton instance. Therefore, a game may have multiple SoundSystem instances but they each operate on the same set of sounds.

Example

To create a SoundSystem instance:

var soundSystem = new SoundSystem ();

### Playing Sounds

To play sounds, the game code simply needs to create an instance of the SoundSystem object and call it's "play" method, passing the unique identifier as an argument

Example

Assuming a sound with a unique identifier of "reelspin":

var soundSystem = new SoundSystem ();

soundSystem.play("reelspin");

### Sound Instances

When called, the "play" method will return an instance of amaya.sdk.sound.SoundInstance. This object allows you to continue to work with the sound once it has been played (for example, stopping the sound before it completes, listening for an event to know when the sound has finished playing etc). For complete information, please refer to the Amaya SDK Library documentation.

Example

Assuming a sound with a unique identifier of "reelspin":

var soundSystem = new SoundSystem ();

var sndReelSpin = soundSystem.play("reelspin");

sndReelSpin.stop();

## Using Your Own Sound Solution

If the SoundSystem implementation is not sufficient to provide sound for the game, developers can implement their own solution. This can be done by adding additional JavaScript libraries our working directly with the HTML5 audio interface. This requires that the game handle all the details of sound formats, browser/feature detection, preloading and playback. For this reason, Amaya recommends that the amaya.sdk.sound.SoundSystem API be used as often as possible.

## Working Around Audio Limitations on Mobile Devices

The iOS and Android operating systems only allow sounds to play within the handler of a user interaction (click/touch). This creates many problems when implementing sound for game development. For example, sounds cannot be played in the following scenarios (unless these scenarios occur as a direct result of the user touching the screen):

* Intro music when a game first loads
* Background music during a bonus round
* Notification sounds when special icons appear during a spin (Bonus icons, Scatter icons)
* Reel stop sounds when a reel finishes animating
* Win sounds when awards are presented to the player

### 

### The Sound Selection Screen

To work around this, the Casino 5 platform detects whether the game is being played on an affected browser (i.e. Mobile Safari on iOS, Chrome on Android) and presents a "sound selection" screen to the player before game loading commences. The user must choose to enable sounds ("Yes" or "No" buttons) to dismiss it and load the game. When the user clicks either button, the game is loaded inside the handler of the event. Due to the nature of closures in JavaScript, the entire game is "trapped" within the scope of the click event. This allows sound to be played at any time within the game, making all of the above scenarios possible.  
  
A generic sound selection screen is provided by the framework and will be loaded when required.

# Game Notification Events

All games built on the Casino 5 Framework must call the following notification methods on Game Services at the correct time within the game. These events are used to assist with the integration of the game into the framework. When these methods are called, the framework can react accordingly. In addition, the framework may need to communicate with other integrated gaming systems based on the type of notification event.

## notifyGameStart

The GameServices method "notifyGameStart" tells the framework that the game is now in progress. By default, the menu and game settings panels are disabled at this time

## notifyGameEnd

The GameServices method "notifyGameEnd" tells the framework the game is no longer in progress. By default, the menu and game settings panels are re-enabled at this time. The notifyGameEnd method should only be called once an entire "round" is completed. Some examples:

* At the end of a simple spin: A spin is started by pressing spin (notifyGameStart is called). At the end of the spin, the players winnings are presented. At this time, notifyGameEnd would be called.
* At the end of a multi-spin round: A spin is started by pressing spin (notifyGameStart is called). At the end of the spin, the player is awarded a bonus round or free spins. At this point, notifyGameEnd is NOT called. Instead, the game should complete all the additional actions (spins, progress through bonus round etc) before calling notifyGameEnd.
* At the end of a hand of cards: A hand is started when the first cards are dealt to the player (notifyGameStart is called). At the end of the hand, after all intervening decisions and additional cards have been dealt, notifyGameEnd is called. This would typically coincide with the completion of any payouts.

# Mobile Device Information

## Application Icons

The Casino 5 framework employs a dynamic system for providing application icons for users that wish to make a “home screen” shortcut.

“lInk” tags are written into the page at runtime, in the following order:

|  |  |  |
| --- | --- | --- |
| Dimensions | URL | Used By |
| 152x152 | <game webapp name>/casino5/apple-touch-icon-152x152.png | Retina iPad with iOS7 |
| 144x144 | <game webapp name>/casino5/apple-touch-icon-144x144-precomposed.png | Retina iPad with iOS6 |
| 120x120 | <game webapp name>/casino5/apple-touch-icon-120x120.png | Retina iPhone with iOS7 |
| 114x114 | <game webapp name>/casino5/apple-touch-icon-114x114-precomposed.png | Retina iPhone with iOS6 |
| 76x76 | <game webapp name>/casino5/apple-touch-icon-76x76.png | Non-Retina iPad with iOS7 |
| 72x72 | <game webapp name>/casino5/apple-touch-icon-72x72-precomposed.png | Non-Retina iPad with iOS6 |
| 57x57 | <game webapp name>/casino5/apple-touch-icon-precomposed.png | Non-retina iPhone with iOS6, iPod Touch, Android 2.1+ |

## Generating App Icons using the Photoshop Template

A Photoshop template and script for generating all of the icons required with the proper names is provided in the Game Development Platform in the "tools/icons" directory. Copy the template to your game's src directory.

The template (icon.psd) is a 1024x1024 document with 3 main folders in the layers panel

* **Guides:** Contains layers with guide masks for iOS6 and iOS7. These can be used to see how your artwork will cropped in each OS.
* **Amaya Overlay:** Contains the Amaya logo that must appear over the lower-right corner of the icon.
* **Icon Art:** Contains template artwork that should be removed and replaced with the artwork for the game icon

Once the artwork for the icon is complete, select File > Scripts > Browse... which will open a dialog to select a script. Navigate to the Game Development Platform's tools/icons/icon.jsx file a select it. The script will run, resizing the image several times and saving out a PNG file sized and named according to the list above. The files will be automatically be saved alongside the PSD. At this point, the files can be moved to the game's webapp/casino5 directory.

## Running the Game as a Web Application

The iOS platform allows a webpage to be run as web application to more closely simulate a native application. The Casino 5 launch page adds the required meta-tag information to enable this mode. There is no action required by game developers.

# Localization

Localization is the process of adding functionality to game to load and display text translated into the player's chosen language.

In general, there are two high-level approaches to implementing localized text into a game: using translated strings inserted into text fields or using translated images that are inserted into the game display. In many games, a combination of both approaches will need to be used.

## Supported Languages

The Casino 5 Framework supports the following languages:

|  |  |  |
| --- | --- | --- |
| Language | Language Code | Orientation |
| Catalan | ca | Left-to-Right |
| Chinese (Simplified) | cn | Left-to-Right |
| Czech | cz | Left-to-Right |
| Danish | da | Left-to-Right |
| German | de | Left-to-Right |
| Greek | el | Left-to-Right |
| English | en | Left-to-Right |
| Spanish | es | Left-to-Right |
| French | fr | Left-to-Right |
| Hebrew | he | Right-to-Left |
| Hungarian | hu | Left-to-Right |
| Italian | it | Left-to-Right |
| Japanese | ja | Left-to-Right |
| Korean | ko | Left-to-Right |
| Dutch | nl | Left-to-Right |
| Norwegian | no | Left-to-Right |
| Polish | pl | Left-to-Right |
| Portuguese | pt | Left-to-Right |
| Romanian | ro | Left-to-Right |
| Russian | ru | Left-to-Right |
| Slovak | sk | Left-to-Right |
| Swedish | sv | Left-to-Right |
| Turkish | tr | Left-to-Right |
| Chinese (Traditional) | tw | Left-to-Right |

## Using Strings

Using strings to localize text involves:

* loading data containing the translations
* coding the game to read the required translation from the file and insert it into a text field in the game

The data can be loaded from any source but in practice, Amaya recommends using JSON files. The process of creating, formatting, positioning and inserting text into the text field is dependent on the technologies used to program the display tier of the game.

**Note:** Before continuing, please be sure that you have read the "*Using Preloaded Localized Files: The "lang" Keyword*" section of this document. It contains important information on the preloading mechanism and how asset id's are defined.

Amaya's process requires making use of the preloading mechanism built into the Casino 5 framework. A JSON file is created for each language, following a prescribed naming convention. The framework will load the file matching the user's chosen language and will then store it as an asset with a generalized id.

Example:

Create a "lang" folder under "preload" and create the following files

+ webapp

+ casino5

+ preload

+ lang

mygamename\_ca.json

mygamename\_cn.json

mygamename\_cz.json

mygamename\_da.json

mygamename\_de.json

mygamename\_el.json

mygamename\_en.json

mygamename\_es.json

mygamename\_fr.json

mygamename\_he.json

mygamename\_hu.json

mygamename\_it.json

mygamename\_ja.json

mygamename\_ko.json

mygamename\_nl.json

mygamename\_no.json

mygamename\_pl.json

mygamename\_pt.json

mygamename\_ro.json

mygamename\_ru.json

mygamename\_sk.json

mygamename\_sv.json

mygamename\_tr.json

mygamename\_tw.json

Each file will have the exact same structure, with different translation values.

English file contents:

{

"myToken": "My English Value.",

"myToken2": "Another English phrase."

}

French file contents:

{

"myToken": "My French Value." ,

"myToken2": "Another French phrase."

}

The same would be performed for each file in the series.

By following the naming convention described in the "*Using Preloaded Localized Files: The "lang" Keyword*" section of this document, only the file matching the current language will be downloaded. It will be stored in the "assets" object using a generalized index of "lang/mygamename\_lang.json". The code in the game can safely utilize this asset and its declared variables and the correct translation will be used.

var langFile = gameServices.getAsset("lang/mygamename\_lang.json");  
myTextFile.text = langFile.myToken;

### Text Encoding

Many of the supported languages use character sets other than Latin (ex. Russian, Chinese, Japanese, Korean, Greek etc). In order to ensure the correct display of translations, any files containing the translated values must be encoded in UTF-8. When encoded incorrectly, some or all of the characters in the translation may appear missing or corrupted.

Depending on the operating system and editor used to create the files, the default encoding may be listed as ISO-8859-1, ANSI, CP1252, UTF-8 or another character set. Characters may even display correctly in the editor but appear corrupted when rendered in the browser. Check to ensure that your editor is saving the files correctly. This varies in each editor. Notepad++ is a free editor that makes it simple to change the encoding of a file using the "Encoding" menu.

To avoid corruption of any data, it is good practice to ensure the encoding is set correctly before inserting translations into the file. When opening the files at a later time (for viewing or editing), be careful they do not become corrupted by an editor which does not support the encoding settings. In some cases, the editor may change the encoding upon opening the file and saving it may result in corruption of the data.

## Using Images

In certain instances, using a string for localized text does not provide the graphic quality required for the game. In these cases, images can be used to localize the text.

For example, the game requires buttons that use a special font with a gradient color applied. In this case, it may not be feasible (or even possible) to create the text using a string and a text field. Instead, the labels can be created as images and included in the game.

The process works exactly as it does for the language files in the previous section.

Example

+ webapp

+ casino5

+ preload

+ standard\_1.33

button\_ca.png

button\_cn.png

button\_cz.png

button\_da.png

button\_de.png

button\_el.png

button\_en.png

button\_es.png

button\_fr.png

button\_he.png

button\_hu.png

button\_it.png

button\_ja.png

button\_ko.png

button\_nl.png

button\_no.png

button\_pl.png

button\_pt.png

button\_ro.png

button\_ru.png

button\_sk.png

button\_sv.png

button\_tr.png

button\_tw.png

At runtime, the file matching the current language will be loaded and stored using the generalized id "device/button\_lang.png". The code of the game is safe to use the generic id which will contain the translated image.

var localizedImage = gameServices.getAsset("device/button\_lang.png");

// Insert localized image into game display.

# Developer Tools

## Query String Parameters

The following table defines query string parameters supported by the Casino 5 framework that are useful for developers.

|  |  |
| --- | --- |
| Key | Value |
| ratio | Any positive floating point value, defining the ratio of width to a height of 1, that indicates the desired viewport ratio.  Examples: 1, 1.33, 1.5, 1.6, 1.77, 2.0  Ratios expressed as 4:3, 16:9 etc must be divided out first. For example, 16:9 is 16/9 which is 1.77.  Specifying this value overrides the measured viewport ratio. Example, set the value to 1.33 to force a 4:3 (1.33) viewport ratio.  Note that the value of ratio does not have to be exactly 1.33, 1.6 or 1.7. For example, to test how content would appear on a device with a 1.5 ratio, pass "ratio=1.5" and the system will respond as though the viewport had measured a 1.5 ratio. |
| quality | One of the values "low", "standard" or "high". Specifying this value overrides the measured quality based on the view size. |
| fallback | Either 0 or 1. Any value other than 1 is treated as 0. Default is 1.  When 0 is specified, the system will not fall back to other game implementations when the ideal implementation is not provided. For example, if a game implements "standard\_1.33", "standard\_1.6" and "standard\_1.77" but the framework determines the best game implementation for the viewport is "low\_1.6", the framework will NOT use "standard\_1.6" instead of "low\_1.6". Instead an error will be displayed.  This value is useful when used in conjunction with the "ratio" and "quality" parameters to force a specific game implementation on a specific screen (for example, viewing a phone-centric layout on a desktop development browser). |
| console | Either 0 or 1. Any value other than 1 is treated as 0. Default is 0.  Displays a semi-transparent bar over the top of the game containing useful information about the display as well as a button to open a positioning grid.  In the top-left, a button display "+ Grid" will appear. Clicking this button causes the positioning grid to appear over the game. This grid is useful when positioning items though X and Y coordinates. Clicking anywhere on the grid causes a red horizontal and vertical guide to be drawn at the point of the click. The X and Y location of the intersection is displayed for reference. Note that when open, the positioning grid's canvas blocks mouse events to the game beneath. To interact with the game, click the "- Grid" button in the top-left to close the grid.  In the top-right, a series of numbers are displayed.  Example: standard\_1.77 | 1366x768 | 1253x705 @ 0.92x/0.92x | DPR: 1  Beginning from the left, these values are:   1. The quality/aspect ratio combination currently in use. 2. The "world" size of the game. That is, the un-scaled width and height used within the game code. 3. The actual rendered dimensions of the game area after being scaled to fit the viewport. 4. The scaling factor applied to the width and height. When the "fit to window" option is off, these numbers are constrained to the same value. When "fit to window" is on, the values may be different as the framework will scale the content in both directions without constraining the aspect ratio. 5. 5. The Device Pixel Resolution (DPR) of the device. In most cases this value is 1. On retina displays such as the iPad 4, the DPR may be a value greater than 1. In the case of the iPad 4, the DPR is 2. |

## Resizing The Browser Window in a Desktop Development Environment

In addition to using the query string values to force specific quality and aspect ratio settings, the browser window can also simply be resized. Resizing will NOT change aspect ratio or quality settings but will simply scale the content. However, reloading the window after resizing will cause the framework to recalculate and apply new quality and aspect ratio settings.

# Advanced Topics

## Google Closure Compiler

The Casino 5 framework is built using the Google Closure Compiler (GCC). Unlike other JavaScript minifiers, the GCC is a true compiler which analyzes and can even optimize code while removing whitespace and combining files into a single resource.

The GCC also provides some very helpful features to assist developers to write better higher quality code. By using standard API-doc comments, the GCC can better understand the code it compiles. For example, simply adding the following comment tells the GCC that the variable is a numeric data type and it will fail if a different data type is assigned to it elsewhere in the code.

/\*\*  
\*  
\* @type {number}  
\*/  
var myNumber;

The more annotations you add in your code, the better the GCC becomes at finding errors and warnings that might otherwise go unnoticed.

The GCC also provides a mechanism for assigning order to your scripts. Using the goog.require() and goog.provide() calls at the top of a file allow the compiler to ensure that required files are placed ahead of dependent files in the final output.

The GCC is not required to be used by game developers, it is simply mentioned here as a helpful recommendation. A Maven plugin that wraps the compiler has been developed by Amaya and included for use in projects. The pom file in the “casino5” directory of the template provided includes this plugin as part of an optional build profile that can be enabled. Please refer to the section on building the game for detailed information.

For more information, please refer to the GCC documentation at <https://developers.google.com/closure/compiler/>

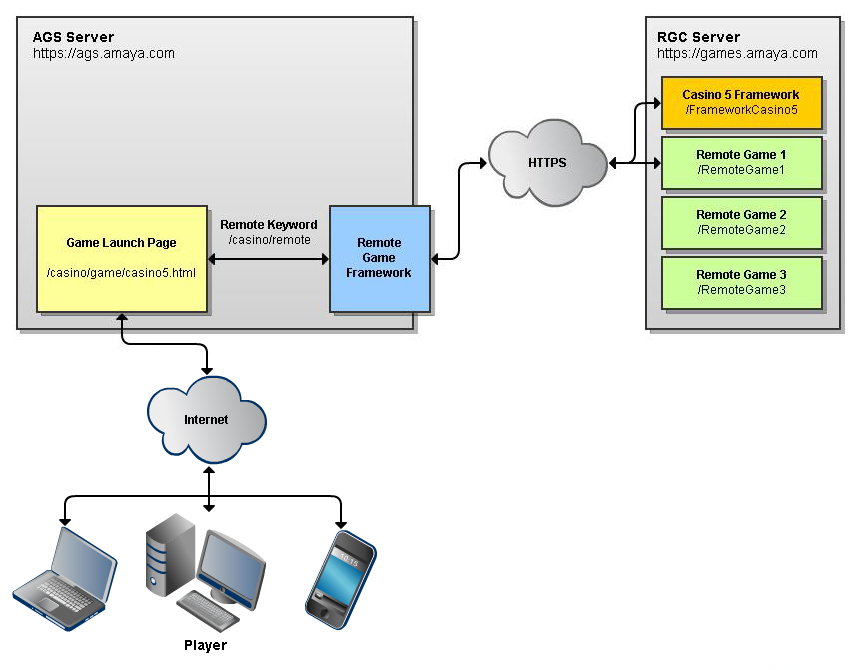
## Postloading and the “Remote” Keyword

While preloading files simplifies game development and easily provides users with a view of the loading progress, sometimes a game can benefit from manual control of loading some of its assets. Perhaps you want to play a short video clip when a player enters a bonus round but do not want to force the user to preload the file. In these instances, a file can be postloaded (aka lazy loading).

The first thing to know about postloading is that the Casino 5 Framework provides no mechanism for postloading files on behalf of a game. Games must initiate and manage the loading process directly.

The structure of a Casino 5 game provides a special “postload” directory for files you wish to prevent from preloading. To access the file, you must first understand how the game webapp is deployed.

All game webapps are deployed to a special server, separate from the AGS server, called the Remote Game Container (RGC). The page that hosts the game resides on the AGS server but loads much of its content from the game on the RGC. Therefore the URL of the page (and the URL which relative requests are resolved against) points to the AGS, not the RGC or your game’s webapp.



In the image above, the player loads “https://ags.amaya.com/casino/game/casino5.html”. This page loads the Casino 5 Framework and the requested game from the RGC by way of the RGF. In order for a request to the AGS to be routed to the RGC server, the “remote” keyword must be used as follows:

https://ags.amaya.com/casino/remote/<remote webapp name>/path/to/file.png

The AGS server filters requests to the “casino” webapp looking for the “remote” keyword. Once found, it assumes the next element of the URL is the name of a webapp hosted on the RGC. Any portion of the URL following the webapp name is assumed to be a path within the remote webapp.

**Note:** The game webapp name is always the “Start Game Name” value of the game within the AGS database. This is usually the game name with no spaces and capitalized first letters. For example, the Start Game Name for “Sample Slot Game” is “SampleSlotGame”. Therefore the webapp name is also “SampleSlotGame”.

Example:

You want to postload a file called “myPostloadFile.png”. The file is located within your game’s webapp in the “casino5/postload/images” directory.

Your webapp structure:

MyGame

+ casino5

+ postload

+ images

myPostloadFile.png

To load this file from your game, you would need to make a request to:

https://ags.amaya.com/casino/remote/MyGame/casino5/postload/images/myPostloadFile.png

Since the game is rooted at “https://ags.amaya.com/casino/game”, you can (and should) make a relative request instead:

/remote/MyGame/casino5/postload/images/myPostloadFile.png

You should ALWAYS make requests relatively from the game. This prevents needing to know the leading portion of the URL including information such as the protocol, domain and webapp on the AGS.

**Note:** In the example above, the relative URL would resolve against “https://ags.amaya.com/casino/game” which would result in a URL of “https://ags.amaya.com/casino/game/remote/MyGame/casino5/postload/images/myPostloadFile.png”. The “remote” keyword is filtered anywhere within the “casino” webapp so it will work identically from “/casino”, “/casino/game” or “/casino/game/some/other/path”. All the matters is that the remote keyword is detected after “casino”.

Finally, using the “remote” keyword, any file within the remote webapp can be accessed, not just those within the “postload” directory. However, all the other directories are preloaded so typically there would be no reason to access anything outside of the postload directory.

# Coming In Future Releases

***The following information is provided for information purposes and is subject to change.***

The Alpha release of the Remote Game Framework Game Development Platform has been prioritized to provide as many features directly related to the development of HTML5-based mobile games as possible. Several features of the framework itself are intended to be developed in parallel with the first games. Additional updates to this package will be provided as new features are completed.

**Casino 5 Framework**

* Social media integration
* Player bonus integration

**Template Games**

* Unit/Integration Testing support

**Documentation**

* Expanded documentation of the Casino 5/HTML5 development platform
* Complete documentation of the Casino 4/Flash development platform