Contents

[Overview 2](#_Toc380154411)

[Glossary 2](#_Toc380154412)

[Diagram Legend 2](#_Toc380154413)

[Game Integration Types 3](#_Toc380154414)

[Expectations of Game Integrations 3](#_Toc380154415)

[Integrating Flash Games via the External Game Interface 5](#_Toc380154416)

[Generic Integration 5](#_Toc380154417)

[Custom Integration 6](#_Toc380154418)

[Integrating HTML5 Games via the External Game Interface 7](#_Toc380154419)

[Generic Integration 7](#_Toc380154420)

[Custom Integration 8](#_Toc380154421)

[Platform Integration Types 9](#_Toc380154422)

[Expectations of Platform Integrations 9](#_Toc380154423)

[Integrating the Flash Gaming Platform 10](#_Toc380154424)

[Generic Integration 10](#_Toc380154425)

[Custom Integration 11](#_Toc380154426)

[Integrating the HTML5 Gaming Platform 12](#_Toc380154427)

# Overview

This document describes various methods used for client (front-end) integrations. It covers both Flash and HTML5 platforms and provides information that game vendors can use to integrate games into the system. It also provides information for integrating the entire platform into an existing vendor product.

This document only covers the client portion of the integration. Additional integration work is required at the server level and is covered in another document.

In addition, this document focuses on the conceptual models supported and/or provided by Amaya's platforms. Additional technical documentation such as API documents that describe classes and events are provided in separate documents with the relevant library (if applicable).

# Glossary

**Flash Games:** ActionScript 3.0 Flash games designed for browser-delivery to desktops, laptops and embedded systems.

**HTML5 Games:** JavaScript HTML5 games designed for browser-delivery to mobile devices such as phones and tablets. These games can be played on desktop, laptop or embedded systems as well.

**Casino 4.0:** Amaya's Flash gaming framework that is responsible for all the common functionality across the suite of games.

**Casino 5.0:** Amaya's HTML5 gaming framework that is responsible for all the common functionality across the suite of games.

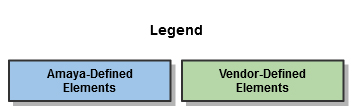
**CGS:** Amaya's gaming platform. The CGS refers to all the components of the system except the direct implementation of the games themselves. This would include the database, transactional system etc.

**RGF:** Remote Game Framework. The RGF is a component of the CGS that allows games to be built "natively" for the platform and operate as plug-ins to the CGS itself. In the context of integrations, an RGF "game" is often developed as a lightweight adaptor to sit between the Casino 4.0 and 5.0 frameworks and the external game.

**EGI:** External Game Interface. The EGI is a component of the CGS that allows integration of external games, both on the server (transactions) and client (presentation).

# Diagram Legend

All diagrams within this document color elements based on who is responsible for the development of the component.



# Game Integration Types

* **Flash Game via the External Game Interface (EGI):** This method refers to integrating an existing Flash game within the Casino 4.0 Framework used to run Amaya's Flash games. This is the most complete method of integrating Flash games and typically is integrated in one of two ways:

* + Generic Integration: In a generic integration, an adaptor built by Amaya is used to load and present the external game. An API is provided to allow the external game to interact with the framework.
  + Custom Integration: In a custom integration, an adaptor is built to facilitate communication between the game and framework. This method is typically used when the external game already has an integration API or when additional customizations are required.
* **HTML5 Game via the External Game Interface (EGI):** This method refers to integrating an existing HTML5 game within the Casino 5.0 Framework used to run Amaya's HTML5/Mobile games. This is the most complete method of integrating HTML5 games and typically is integrated in one of two ways:
  + Generic Integration: In a generic integration, an adaptor built by Amaya is used to load and present the external game. An API is provided to allow the external game to interact with the framework.
  + Custom Integration: In a custom integration, an adaptor is built to facilitate communication between the game and framework. This method is typically used when the external game already has an integration API or when additional customizations are required.
* **Direct Launch (Flash or HTML5):** The Direct Launch method is a less complete form of integration, particularly on the client side. This method bypasses the Amaya gaming frameworks and instead redirects from Amaya's game launch page to a vendor-defined page. This method still requires server-side integration.

# Expectations of Game Integrations

With the exception of the Direct Launch method (which bypasses the client framework), integrated games should perform the following:

* Manage the display of the loading progress of the external game (the framework provides no mechanism to display the progress of the external game load).
* Keep the balance field in the Casino 4.0 framework correctly set at all times
* Keep the wagered field in the Casino 4.0 framework correctly set at all times
* Send the amount won at the end of the hand/spin
* Send the net amount won at the end of the hand/spin
* Notify the Casino 4.0 framework when a hand/spin starts
* Notify the Casino 4.0 framework when a hand/spin ends
* Set the game play id when applicable
* Clear the game play id when no longer applicable
* If the game has a built-in sound on/off switch, the state of the switch must be communicated to the Casino 4.0 framework when changed. In addition, the game must also set the state of the switch accordingly when the sound on/off preference in the framework is changed.

In addition to the integration points above, additional information can be useful:

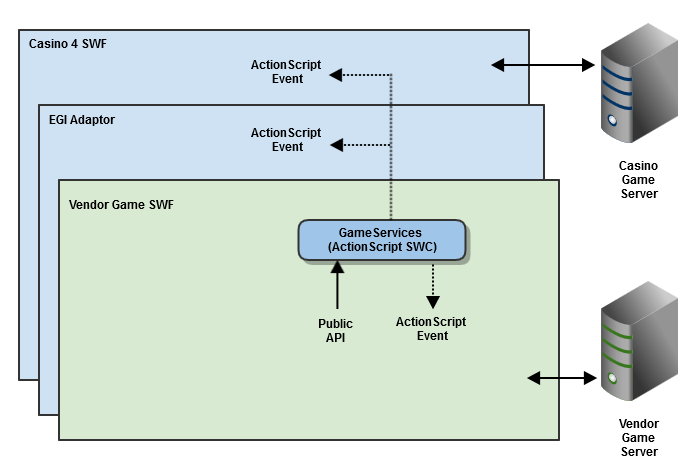
* Information about the loading progress of the external game.

# Integrating Flash Games via the External Game Interface

When integrating Flash Games via the EGI, there are two options, either a generic integration (using a prebuilt adaptor) or a custom integration (using a customized adaptor). In either case, an adaptor SWF is built to facilitate communication between the Casino 4.0 framework and the external game.

## Generic Integration

In a generic integration, the prebuilt adaptor loads a single SWF defined by the vendor for that game. That SWF is given access to a number of properties, methods and events through the GameServices object. Definitions for GameServices and its related classes are provided by a SWC file that the external game SWF can link against during compilation.



Note: The SWF loaded as the external game can download and use any additional resources required. From the perspective of the Amaya platform, all that matters is which SWF to load first. Also, in many cases, the initial SWF to load may be the same "common" file used by all games.

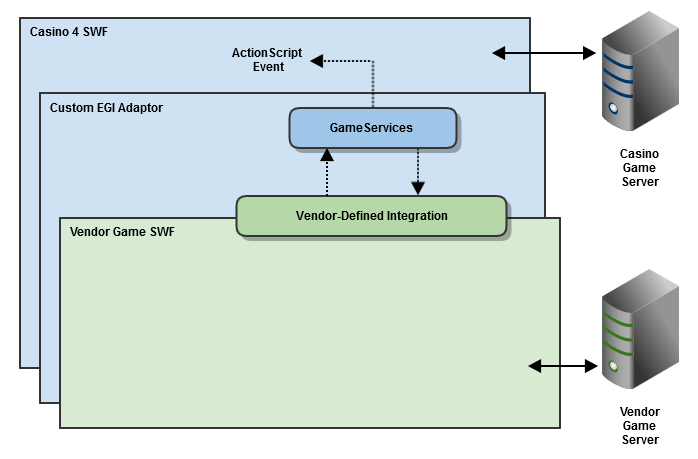
The complete API documentation for GameService.as and its related classes are available but not covered by this document. However, at a high level, GameServices would provide access to:

* game-related information such as the game's display name, game id, language, player information etc.
* methods that can be called to interact with the framework such as setting the balance and wagered fields, showing dialog boxes, displaying errors etc.
* events that will tell a game when important actions have occurred within the framework such as opening of a modal dialog box, change in the state of a user preference, display of an error etc.

As with all EGI games, game communication is performed directly between the external game SWF and the vendor's game server. In other words, game communication does not flow through GameServices, the Casino 4.0 Framework or the CGS.

## Custom Integration

Custom integrations are very similar to generic integrations, with a few notable exceptions. As with generic integrations, an adaptor SWF loads a single SWF defined by the vendor for that game. However, in the case of a custom integration, the external game does not use the GameServices class. Instead, the vendor's own API is used and the custom adaptor maps the vendor's calls and events to those of GameServices (and the Casino 4.0 framework by extension). In this model, the custom adaptor contains more logic and does more of the work directly. As each vendor has different methods of communicating data (and the nature and amount of data communicated is itself different), the custom adaptor is usually vendor-specific.

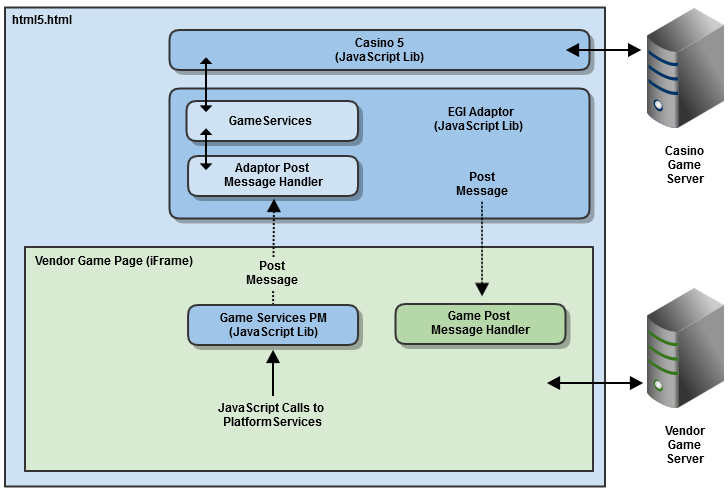


# Integrating HTML5 Games via the External Game Interface

When integrating HTML5 Games via the EGI, there are two options, either a generic integration (using a prebuilt adaptor) or a custom integration (using a customized adaptor). In either case, an adaptor JavaScript library is built to facilitate communication between the Casino 5.0 framework and the external game.

## Generic Integration

In a generic integration, the prebuilt adaptor creates an iFrame and sets its source to a URL defined by the vendor for that game. That iFrame is given access to a number of properties, methods and events through the GameServicesPM object contained within a small JavaScript library that is added to the game's page. GameServicesPM utilizes the browser's Post Message capability behind the scenes to communicate between the main page and the iFrame.



The complete API documentation for GameServicePM are available but not covered by this document. However, at a high level, GameServicesPM would provide access to:

* game-related information such as the game's display name, game id, language, player information etc.
* methods that can be called to interact with the framework such as setting the balance and wagered fields, showing dialog boxes, displaying errors etc.
* events that will tell a game when important actions have occurred within the framework such as opening of a modal dialog box, change in the state of a user preference, display of an error etc.

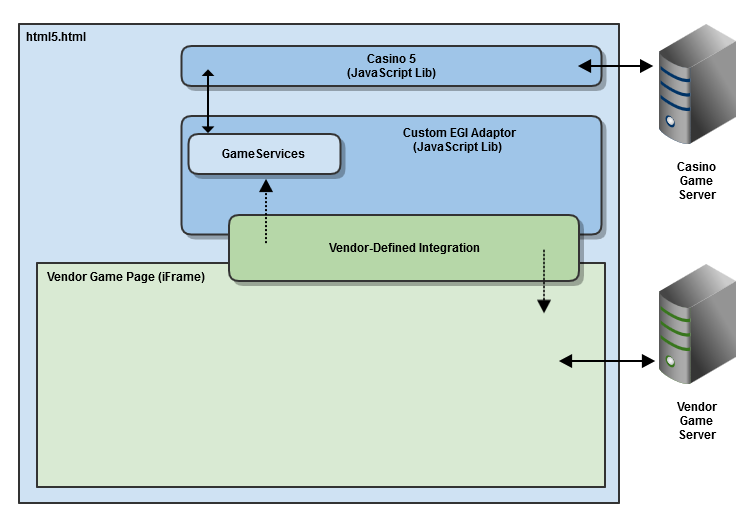
Since all communication between the Casino 5.0 Framework and the external game is accomplished via Post Message, it is inherently asynchronous. This means that when calls are made to Casino 5.0, it cannot directly return information. Instead, any call that should return information does so by sending a "ReturnEvent" Post Message to the game. The message will contain the return value as well as information about the original call that generated it.

In order to decrease the amount of Post Message traffic that is generated, the game must call the GameServicesPM.enableEvent() method to register to receive the specified event. This must be done once for each message the game cares to receive. If no longer needed, the game can call GameServicePM.disableEvent() to stop those events from being sent. The GameServicePM class provides constants for all event types it can register.

As with all EGI games, game communication is performed directly between the external game and the vendor's game server. In other words, game communication does not flow through GameServicesPM, the Casino 5.0 Framework or the CGS.

## Custom Integration

Custom integrations are very similar to generic integrations, with a few notable exceptions. As with generic integrations, an adaptor library is created to generate an iFrame whose source is pointed a URL defined by the vendor. It also serves to facilitate the interaction between the Casino 5.0 framework and the external game. In the case of a custom integration, the adaptor is designed to work with the vendor's API and map it to the GameServices API. In this model, the custom adaptor contains more logic and does more of the work directly. As each vendor has different methods of communicating data (and the nature and amount of data communicated is itself different), the custom adaptor is usually vendor-specific.



# Platform Integration Types

* **Flash Gaming Platform:** This method of integration is used when a vendor has their own gaming platform and wishes to integrate some or all of the suite of Amaya Flash games into it. This integration can be completed in one of two ways.
  + Generic Integration: In a generic integration, An interface is provided that allows bi-directional communication between the host and the gaming platform.
  + Custom Integration: In a custom integration, an adaptor is built to facilitate communication between the host and framework. This method is typically used when the vendor already has an integration API or when additional customizations are required.
* **HTML5 Gaming Platform:** This method of integration is used when a vendor has their own gaming platform and wishes to integrate some or all of the suite of Amaya HTML5 games into it. An interface is provided that allows bi-directional communication between the host and the gaming platform.

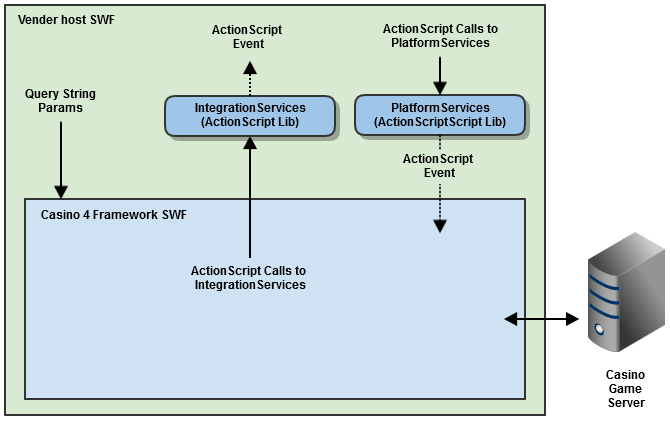
# Expectations of Platform Integrations

The nature of a platform integration is for the Amaya gaming library to be utilized within the context of a vendor's own platform. Therefore, the expectations of the integration lie primarily with the vendor and the needs of its platform. In typical platform integrations, the vendor platform usually requires information about the state of loading progress, game play information such as balances and game events, and system information such as when errors have occurred.

# Integrating the Flash Gaming Platform

## Generic Integration

Integrating the Flash gaming platform is accomplished by loading Amaya's Casino 4.0 Framework SWF into either an HTML page or another SWF. A "Flash-in-Flash" integration is typical. This SWF is loaded first for all games and requires parameters be passed to it to configure the framework.



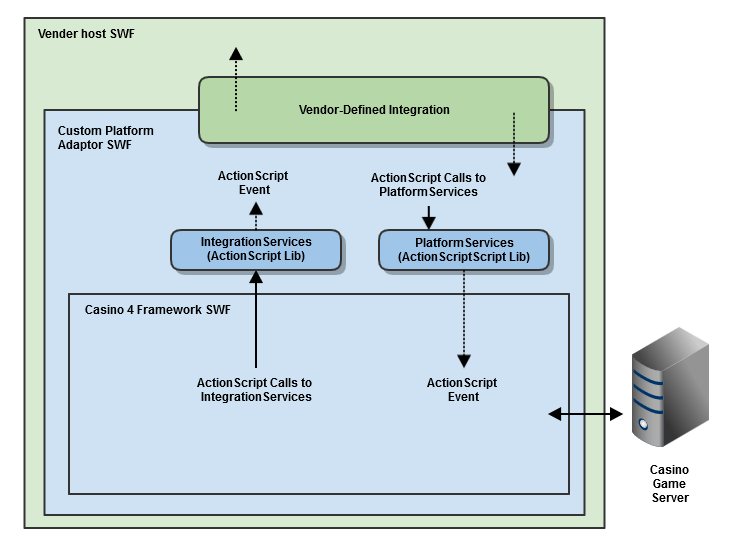
In order to load the framework, the PlatformServices object contains two important methods that perform calls to the CGS to gather startup information. The "login" method can be used to obtain a playerHandle (CGS unique player session id) by passing a vendor's session token to the server to perform the login. The "getGameLaunchURL" takes a set of parameters, calls the server to obtain additional information about the game, checks for unfinished games, and finally returns the url needed to load the Casino 4 Framework.

Once loaded, the framework dispatches ActionScript 3.0 events through an object called IntegrationServices. The host SWF simply creates a new instance of this class and assigns listeners for any events it wants to consume. These events are documented in the API docs for the library. They include information about loading progress, messages sent and received, state of user preferences, errors etc.

Often the host SWF needs to not only receive messages from the framework but also send messages to it. Sending messages is accomplished using the PlatformServices object. The host SWF simply creates a new instance of this class and then calls its public methods to send messages to the framework. The public methods are documented in the API docs for the library. These include changing the state of user preferences, displaying errors etc.

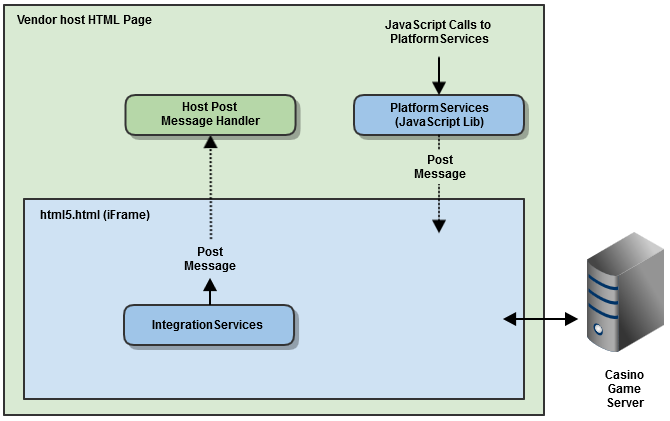
## Custom Integration

In some cases, the vendor platform already provides an integration interface to be used. In this situation, a custom adaptor can be built to manage loading the Casino 4.0 framework and can then serve to map its interface to the vendor's. In this model, the custom adaptor contains more logic and does more of the work directly. As each vendor has different methods of communicating data (and the nature and amount of data communicated is itself different), the custom adaptor is usually vendor-specific.



# Integrating the HTML5 Gaming Platform

Integrating the HTML5 gaming platform is accomplished by using an iFrame and pointing its source to a URL provided by Amaya. Once loaded, interaction between the framework and the host page is facilitated by leveraging the browser's PostMessage API.



Sending messages to the Casino 5.0 Framework is accomplished using the PlatformServices object provided via a small JavaScript library. The host page simply creates a new instance of this class and then calls its public methods to send messages to the framework. The public methods are documented in the API docs for the library. These include changing the state of user preferences, displaying errors etc.

When the Casino 5.0 framework needs to inform the host of a message or a change in state, the framework will invoke a postMessage command containing a JSON object. This object will always be in a consistent format:

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| api | String | A unique identifier for the API. All messages from the Casino 5.0 framework will have a value of "integrationServices" |
| event | String | The name of the event. This value is used to determine how to handle the message. |
| params | Object | An object containing 0 or more key/value pairs of data relevant to the event. See each event's description for information about the keys and values. |

The events and their specific parameters are not covered by this document but can found in the PlatformServices API document provided by Amaya.

In order to decrease the amount of Post Message traffic that is generated, the host must first register for each event it cares to receive. This is done by calling the PlatformServices.enableEvent() method and specifying the name of the event. If the event is no longer needed, the host can stop receiving the event by calling the PlatformServices.disableEvent() method.

There are two slight exceptions to this rule. The "integrationServicesReady" and "frameworkReady" events are always sent without having to be registered for first. The "integrationServicesReady" event tells the host when the framework has loaded far enough that "enableEvent" can now be called to register listeners. The "frameworkReady" event tells the host that methods of the PlatformServices object can now be reliably called.