|  |
| --- |
| Amaya Gaming |
| Remote Game Framework v2.2 |
| Developer Guide |

|  |
| --- |
| Office of Software Development  6/8/2013 |

Contents

[Overview 3](#_Toc323660925)

[How it Works 3](#_Toc323660926)

[The Flash Game Client 3](#_Toc323660927)

[The Remote Game Services 6](#_Toc323660928)

[SWF and Other Static Resources 7](#_Toc323660929)

[Figure 2. High Level Game Messages 7](#_Toc323660930)

[Technology and Design Constraints 8](#_Toc323660931)

[Game Container 8](#_Toc323660932)

[Game Client 8](#_Toc323660933)

[Game Service 8](#_Toc323660934)

[Game Service Reference 10](#_Toc323660935)

[get\_game\_descriptor 10](#_Toc323660936)

[initialize\_game 13](#_Toc323660937)

[replay\_game 16](#_Toc323660938)

[resume\_game 16](#_Toc323660939)

[gaming\_guide 17](#_Toc323660940)

[about\_info 18](#_Toc323660941)

[get\_profile\_parameters (New Profile) 19](#_Toc323660942)

[get\_profile\_parameters (Copy Profile) 20](#_Toc323660943)

[validate\_profile\_parameters 21](#_Toc323660944)

[Other Game Specific Requests 22](#_Toc323660945)

[manifext.xml 26](#_Toc323660946)

[logo.png 27](#_Toc323660947)

[Figure 3. High Level Deployment Scenario 28](#_Toc323660948)

# Overview

The Remote Game Framework is a powerful game development framework and platform. It provides a means to develop games for the Amaya Gaming System (AGS) such that the game code remains simple, loosely coupled, and largely independent of the AGS. The Remote Game Framework leverages a simple HTTP based web architecture that allows game logic to be separated from the game messaging and transaction infrastructure. Not only is it possible to build games remotely, but also to deploy and run the games remotely as well. In addition, the Remote Game Framework leverages modern standards that allow for rapid deployment and upgrades of games into live, running production environments with zero downtime.

# How it Works

A Remote Game Framework game is made up of two key parts:

* The Flash Game Client
* The Remote Game Services

## The Flash Game Client

The Flash Client is implemented using Amaya’s Casino 4 Framework. This framework is broken into three key parts:

* The GameServices API – which provides access to the features and functions of Amaya’s game wrapper. This includes the top and bottom bars surrounding the game, game menus, dialog boxes and other common components required to interact with Amaya’s Gaming System. It is also the means by which the game can send and receive messages to and from the server.
* The Core SDK – provides low level access to key features required by games such as keyboard interaction, sounds, currency formatting and messaging classes.
* The Language Library – Provides features and functions that assist with internationalization of games.

The Flash Game Client is loaded into the framework as a child application.

### Figure 1. Conceptual Game Client / Framework Relationship

Menu

Menu

Balance

Dialog

Your Game

### Game.as

Menu

Menu

Balance

Dialog

Your Game

The game client is implemented using a main class, called Game.as. This class is then compiled as game.swf (located in web/game/controller/). Game.as is stored in the src/actionscript3 (classpath) directory in the com.cwh.<gamename> package. When using the supplied Ant build, the Game.as class is automatically found within the classpath directory and is compiled as game.swf in the correct location.

A partial game implementation is provided in the Core SDK library (com.cwh.casino4.sdk.singleplayer.game.V2Game). It is recommended that your Game.as class inherit from V2Game as this class takes care of a number of standard startup procedures. Simply override the “startMain” and “resumeMain” functions which provide entry points for starting new games and resume unfinished games respectively. When inheriting from V2Game, the constructor of Game.as is typically left empty.

### External Assets

External assets (graphics, sound, configuration) can be preloaded by placing them within the web/game directory. Due to the nature of how the Casino 4.0 framework loads these files, only .swf, .jpg, .gif, .png and .xml files can be included in this directory for preloading. These files are then available to the game at runtime through GameServices using the path to the file relative to the web/game directory.

The Game.as class typically adds graphics by accessing preloaded swf’s via the GameServices class. Since the game.swf is itself a Sprite and is added to the display list by the framework during the loading phase, graphics can be added as a child of Game.as directly.

Sound files are typically published as library items of a swf file contained within the web/game directory. Due to a limitation of the Flash Player API, .mp3 files cannot be preloaded by the Casino 4.0 framework.

### Internationalization

Internationalization is the process of translating the game into other languages at runtime. There are many ways of implementing this type of feature but Amaya provides some classes that can assist in this process.

The Amaya Language library (chartwell\_lang\_<version>.swc) provides access to common strings through classes such as CommonLang.as and SlotsLang.as. These classes are pre-populated with a number of phrases translated into many languages. Refer to the documentation of this library for more information.

Most games utilize common strings from the library in addition to their own game-specific phrases. These game-specific phrases are typically added to a language class that inherits from com.cwh.lang.dynamiclang.AbstractDynamicLang. Simply add public static variables (String type) to this class that have no values. Override each languages method (for example, the “en” method for English) and set the value for each static variable. In the game, you can now insert the correct translation at runtime by statically referencing YourLangClass.YourVariable. For more information, refer to the library’s documentation.

Sometimes, a game requires text that cannot be properly implemented using dynamic text fields (for example, text that uses custom fonts, strokes, drop-shadows etc). This text, referred to as Static Text, can be internationalized using a movieclip that conforms to an exact format and references the com.cwh.lang.staticlang.StaticTextUtils class. For more information, refer to the library’s documentation.

## The Remote Game Services

The server side of a Remote Game is developed as a web service that must respond to specific web requests and receive and return game messages using the Remote Game Protocol. This callback design ensures that the Remote Game never needs to request a connection to the Amaya Gaming System (AGS), but instead only needs to respond to request from it. This simplifies both implementation and deployment.

The web service request format is simply:

* http://<domain>/<GameName>/<RequestName>

Game messages will be described in full detail later. However, the standard requests that the game service must respond to are:

* get\_game\_descriptor – This request is made only once, upon installation of the game. The AGS will request the Game Descriptor which contains all information required to initialize and install the game. This includes the Game Name, Game ID, Game Profile Schema and Default Game Profiles. This service is only requested once upon installation of the game.
* initialize\_game – This request is used to configure each request that the AGS will make to the game. The configuration includes the random number pools and different game state scopes required by each request. This service is requested in two cases: First, when the AGS starts up. Second, when new game profiles are added.
* replay\_game – This request renders historical game data in a format that a player or operator can easily read and understand. Games are required to do this as the interpretation of the data is fully dependent upon the game rules.
* resume\_game – This request handles a case whereby historical game state must be used to re-initialize an unfinished game so that the player can continue where they left off (for example, in the case of an unexpected network interruption that causes the game to fail).
* gaming\_guide – Returns the game rules for the game in any of the supported languages.
* about\_info – Returns fine grained elements that contain game information. These can be used by other systems to use for generating content based on game details. Currently only RTP values are supported.
* All Other Requests: All other requests are game specific and thus defined by the game in the result of the initialize\_game request.

|  |  |  |  |
| --- | --- | --- | --- |
| **Service** | **Method** | **Request Content Type** | **Response Content Type** |
| get\_game\_descriptor | GET | N/A | text/xml |
| initialize\_game | POST | text/xml | text/xml |
| replay\_game | POST | text/xml | text/html |
| resume\_game | POST | text/xml | text/xml |
| gaming\_guide | GET | N/A | text/html |
| about\_info | GET | N/A | text/xml |
| manifest.xml | GET | N/A | text.xml |
| Other Game Requests | POST | text/xml | text/xml |

## SWF and Other Static Resources

In addition to responding to requests, the Remote Game Service serves up the Flash Game Client files (SWF) and other static content required by the Flash Game Client itself. These files must be placed in specific locations to ensure proper loading. In addition, a manifest.xml file must be maintained (or generated by the standard build file included). The manifest.xml lists all resources that the Flash Framework will load, and that are required for your game to run properly.

The following diagrams depict the various components and basic interactions.

## Figure 2. High Level Game Messages

menu

balance

dialog

Flash Framework

Framework Events

Flash Game Client

Framework Functions

Game Response

Game Request

Remote Game Services

Amaya Gaming System

/swfs + manifest.xml

/get\_game\_descriptor

/...game specific requests...

/resume\_game

/replay\_game

# Technology and Design Constraints

The following technology constraints should be respected when implementing a Remote Game.

## Game Container

* The “Game Container” is a fancy term for a “web server”. In production, Amaya will use a typical Java Servlet container such as Resin or Tomcat. Sometimes these are fronted by Apache HTTPD.
* For development, we recommend using the same, but it should be safe to use pretty much any HTTP server solution. Games should work behind firewalls and with or without HTTPS.

## Game Client

* The Remote Game Framework requires that the game client is developed using Flash 10 compatible SWF binaries, and that all source code be developed using ActionScript 3.
* The Flash Client must not be directly aware of the location of the game container or the resources that it provides. All Flash client messaging still routes through the AGS.

## Game Service

* Technically speaking, the Remote Game Framework does not require any specific technology on the server side, as long as it can be implemented behind a HTTP web server. Any technology such as Java Servlets, ASP.NET, Ruby on Rails or even PHP can be used to implement the server side logic.
* However, for games developed as work for hire for Amaya, for which ownership, stewardship and maintenance will be owned by Amaya, they must be implemented using the following technologies:
  + Java 6+
  + Java Web Application Archive (WAR file deployable)
  + Java Servlets preferred over JSP
  + Frameworks we recommend include:
    - Stripes (action based web framework)
    - Guice (dependency injection)
    - XStream (Java/XML marshalling)
    - Ant build system
  + For convenience, Amaya may provide a template that includes the scaffolding for the above server side technologies
* Unidirectional Messaging: The AGS must call into the game to process game rules and requests. Neither the Game nor the Game Container may call into the AGS. This is critical to ensure simplicity and security both in the development and deployment of games.
* HTTP (including HTTPS) Only Messaging: The only protocol or communication mechanism allowed to make requests to the Game or Game Container is HTTP. HTTPS will also be supported, but the application must remain unaware and independent of whether HTTP or HTTPS is in use.
* No Access to System Resources: The Games and Game Container may not access the database directly, nor may they access the file system, network sockets, native libraries (JNI) or the System console directly (including System.out). This is another way of saying that the Games and Game Container may only respond to requests made by the AGS.
* No Threads: Games may not spawn threads, nor may plug-ins, filters or other components deployed to the Game Container spawn threads. This also implies that there should never be a need to use the “synchronized” keyword (or other locking mechanisms).
* Stateless: Games and the Game Container must remain stateless. This means no static mutable fields, no use of the Servlet session or context attributes and no caching logic.
* Location Transparency: Games may not know where they are deployed. They must not depend on their server name, their directory, the protocol or their location on the file system. The Game Container and the AGS may know this information.
* Currency Independent: Games must be unaware of the currency that they are using. However, the AGS may send the rounding scale to the game, so that the game can appropriately round partial win values.
* Multi-vendor: A single Game instance within a Game Container should be reusable by any and all clients and affiliates. While this may not be practical from a legal or jurisdictional perspective, as a design goal it is important, especially for testing and staging environments.
* One Step: Game installation must consist of only one step. Namely this step is the deployment of a WAR file to a JEE compatible application server such as Resin.
* Convention over Configuration: Wherever possible, including for all game URLs, names, and requests, convention should be preferred over deployment-time configuration files.
* Complete Game: The entire game and all of its resources must be included in the WAR file. This includes (but is not limited to):
  + The Flash Client SWF files with the base skins
  + A manifest file that contains all information about client side (Flash) resources to be downloaded
  + All other client skins for a particular deployment
  + The gaming guide in all supported languages.
  + The replay rendering logic.
  + All server side game logic (Java code)
  + Any 3rd party libraries or frameworks (including, in some cases, internally developed frameworks).
  + An XML descriptor or service that contains enough information for the AGS to appropriately populate database tables, including Games, Gdk\_GameSettings, Game\_Suite, Game\_Group, and Game\_Profile. This includes the Game Profile XSD template and the default profiles themselves. The AGS will be responsible for loading this information into the database.

# Game Service Reference

The following sections detail the web service APIs for games. Remember, that the Remote Game Framework does not require the games to make these calls, but rather implement and respond to these calls.

## get\_game\_descriptor

This request is made only once, upon installation of the game. The AGS will request the Game Descriptor which contains all information required to initialize and install the game. This includes the Game Name, Game ID, Game Profile Schema and Default Game Profiles.

The game descriptor XML response is returned by the Game Service which the AGS calls upon initial installation of the game. It returns game information used to register the game and default game profiles in the system.

### Request URL

http://<domain>/<game\_name>/get\_game\_descriptor

### Response Body

1. **<gameDescriptor>**
2. **<gameInfo** gameId="" suiteName="" groupName="" gameName="" displayGameName="" pjMode="" bonusMoneyAllowed=""**>**
3. **<platformModes>**
4. **<platformMode** platform="" mode=""/**>**
5. **</platformModes>**
6. **<minorGameInfo** gameId="" gameName="" displayGameName="" pjMode="" bonusMoneyAllowed=""**/>**
7. **<paramXSD>**
8. **</paramXSD>**
9. **</gameInfo>**
10. **<gameTemplates>**
11. **<gameTemplate** curClass="" suffix="" webEnable="" downloadEnable=""**>**
12. **</gameTemplate>**
13. **<gameTemplate** curClass="" suffix="" webEnable="" downloadEnable=""**>**
14. **</gameTemplate>**
15. **</gameTemplates>**
16. **</gameDescriptor>**

#### gameDescriptor

This is the root element of the get\_game\_descriptor response.

Child Elements:

* gameInfo
* gameTemplates

#### gameInfo

Contains elements and attributes that describe the identity, type and system features supported.

|  |  |
| --- | --- |
| Attribute | Description |
| gameId | The unique numeric identifier for the game. This ID will be supplied by Amaya. |
| suiteName | The game suite that this game should belong to. Amaya will provide this string. |
| groupName | The game group that this game should belong to. Amaya will provide this string. |
| gameName | The name of game that the system will use. No spaces or special characters should be used. The technical significance of the game name is that it will be used in all URL requests – thus the path that the game is deployed to (i.e. the web application context) should match the game name. |
| displayGameName | This is a more human friendly version of the game name that will be used for display in menus. It may contain spaces and other special characters. |
| pjMode | This is a 0 or 1 setting that determines if the game can be included in a Progressive Jackpot. |
| bonusMoneyAllowed | This is a 0 or 1 setting that determines if the game allows bonus money to be used for betting during real play. |
| ~~mode~~ | **(*DEPRECATED – Mode will still be available for legacy needs.)***  This setting identifies the version of the client side framework that should be used. Currently the only valid setting for remote games is “casino4”. |

Child Elements:

* platformModes
* minorGameInfo
* paramXSD

#### platformModes

A RGF based game may support multiple platforms out of the box. To support this, different frameworks were developed for these platforms. Each platform that the RGF game supports, must be associated with a framework, the platformModes element will defined this.

Currently, valid platforms and modes are:

* flash
  + casino1
  + casino2
  + casino3
  + casino4
  + casino4\_800x600
* html5
  + casino5
* flashmini
  + casino3mini
  + casino4mini

|  |  |
| --- | --- |
| Attribute | Description |
| platform | Identifies the platform, for which a mode is assigned to. |
| mode | Defines the version of the client side framework that should be used. |

#### minorGameInfo

A minor game is like a “game within a game”. Many games will simply implement such games as features of the main game, but some games require separate reporting or contain a feature that is big enough to warrant the treatment of a full game. Like its parent element, the minorGameInfo element contains elements and attributes that describe the identity, type and system features supported by the minor game. However a minor game cannot have a suite, group and needs not specify the mode, as all are inherited from the main game.

|  |  |
| --- | --- |
| Attribute | Description |
| gameId | The unique numeric identifier for the game. This ID will be supplied by Amaya. |
| gameName | The name of game that the system will use. No spaces or special characters should be used. The technical significance of the game name is that it will be used in all URL requests – thus the path that the game is deployed to (i.e. the web application context) should match the game name. |
| displayGameName | This is a more human friendly version of the game name that will be used for display in menus. It may contain spaces and other special characters. |
| pjMode | This is a 0 or 1 setting that determines if the game can be included in a Progressive Jackpot. |
| bonusMoneyAllowed | This is a 0 or 1 setting that determines if the game allows bonus money to be used for betting during real play. |

#### paramXSD

The paramXSD element contains an XML Schema Descriptor (XSD) as its body element. This XSD will describe the constraints used to validate the <param> elements that are discussed below as part of the gameTemplates section.

Example Body Content

1. **<paramXSD>**
2. **<xs:schema** xmlns:xs="http://www.w3.org/2001/XMLSchema"  elementFormDefault="qualified"**>**
3. **<xs:element** name="param"**>**
4. **<xs:complexType>**
5. **<xs:attribute** name="coinValue" type="coinValue"
6. use="required" **/>**
7. **<xs:attribute** name="maxGambleWinnings" type="winnings"
8. use="required" **/>**
9. **</xs:complexType>**
10. **</xs:element>**
11. **<xs:simpleType** name="coinValue"**>**
12. **<xs:restriction** base="xs:string"**>**
13. **<xs:pattern** value="([0-9]{1,}[.][0-9][0-9])(([,][ ]\*([0-9]{1,}[.][0-9][0-9]))\*)" **/>**
14. **</xs:restriction>**
15. **</xs:simpleType>**
16. **<xs:simpleType** name="winnings"**>**
17. **<xs:restriction** base="xs:string"**>**
18. **<xs:pattern** value="[0-9]{1,}(\.[0-9][0-9])?" **/>**
19. **</xs:restriction>**
20. **</xs:simpleType>**
21. **</xs:schema>**
22. **</paramXSD>**

See below for examples of <param> elements that are validated by the XSD that is defined in the paramXSD element.

#### gameTemplates

This is a collection element that contains a number of templates used to create the default game profiles for this game upon installation.

Child Elements:

* gameTemplate

#### gameTemplate

A game template is used to create default game profiles for the game upon installation. Game profiles are used to configure game instances that are available for players to play. Each game profile defines parameters like coin values and limits. Some settings are standard and requires, these are defined in the attributes of the gameTemplate element. Custom attributes can be defined in the param child element, which is validated by the paramXSD element described above.

|  |  |
| --- | --- |
| Attribute | Description |
| curClass | A currency class defines a relative weighting of the value of a currency that a particular game profile should apply. For example, because GBP is worth hundreds of times more than KRW, the system will ensure that fair rules are applied to contributions to progressive jackpots. Valid values are 1 – 20. |
| suffix | The suffix is used in creating the name of the default profile. |
| webEnable | This is a 0 or 1 option that indicates whether this profile is enabled for play in a web browser. |
| downloadEnable | This is a 0 or 1 option that indicates whether this profile is enabled for play in the download client. |

Child Elements:

* param

#### param

The param element contains the fully customizable attributes of a game profile. The param element will be validated by the XSD defined in the body of the paramXSD element described above. An example of a gameTemplate with a param element is as follows:

1. **<gameTemplate** curClass="1" suffix="0.02-5.00" webEnable="true" downloadEnable="false"**>**
2. **<param** coinValue="0.02,0.05,0.10,0.25,1.00,2.00,5.00" maxGambleWinnings="100000.00"**/>**
3. **</gameTemplate>**

The attributes in param element should adhere to the following restrictions:

The client frameworks support displaying a profile selection menu based on the profile data returned by the game profile list servlet. This only occurs when launching by gameId AND two or more profiles exist for the same account (i.e. USD). There are six unique profile parameter formats that can be detected by the frameworks. If the profile doesn’t conform to one of these six formats, the menu can’t be displayed and the player can’t choose which one to play.

Any number of additional parameters can be added to any of these. However, since the framework detects which type of menu to create based on the existence of these key attributes, it would be possible to confuse it by adding parameters that would make it match the wrong type. For example, if you add “coinValue” to a table game, the menu will present it the profiles as slot profiles.

Table Game 1 (Min bet and Max bet apply to the total of all bets)

Required attributes:

* minBet (Number)
* maxBet (Number)

Table Game 2 (Min bet and Max bet apply per bet)

Required attributes:

* minPerBet (Number)
* maxPerBet (Number)

Roulette

Required attributes:

* maxStraightUpBet (Number)
* minInsideBet (Number)
* minOutsideBet (Number)
* max12OutsideBet (Number)
* max18OutsideBet (Number)
* minBet (Number)
* maxBet (Number)

Slot Machine/Video Poker (Multiple credit values)

Required attributes:

* coinValue (Comma-delimited list of Numbers)

Slot Machine/Video Poker (Single credit value)

Required attributes:

* singleCoinValue (Number)

Instant Games (Pull tabs and scratch cards)

Required attributes:

* InstantGameCoinValue (Comma-delimited list of Numbers)

## initialize\_game

The initialize\_game request is requested by the AGS to configure all game specific requests.

### Request URL

http://<domain>/<game\_name>/initialize\_game

### Request Body

1. <!-- Sent to http://.../<gameName>/initialize\_game -->
2. **<gdkRequest>**
3. **<gameHeader>**
4. **<historicalState>**
5. <!-- This content is specified by the Game -->
6. **</historicalState>**
7. **</gameHeader>**
8. **</gdkRequest>**

#### historicalState

The body of the initialize\_game request is designed to leave room for future extension. Currently the only relevant element is the historicalState element that is a child to the gameHeader and inherently the root element for all suh respects: gdkRequest.

The body of the historicalState element is entirely game specific and is set in responses to other game requests. Historical state is a long term game state feature that allows games to maintain state between game sessions and can be shared between players (e.g. for an in-game progressive jackpot feature). It is most commonly represented as XML.

### Response Body

1. **<gdkResponse version=”2”>**
2. **<gameInitialize>**
3. **<request** name=""**>**
4. **<randomPools>**
5. **<pool** id="" size="" min="" max="" **/>**
6. **<pool** id="" size="" min="" max="" increment=""**/>**
7. **</randomPools>**
8. **<historicalState** required="" **/>**
9. **<persistentState** required="" **/>**
10. **</request>**
11. **</gameInitialize>**
12. **<gameError>**
13. **</gameError>**
14. **</gdkResponse>**

#### gameInitialize

The gameInitialize element contains one <request> element or each game specific request that the game should expect. For example, if a game supports requests such as “bet”, “split”, “double”, “stand”, then there would be one <request> element included for each.

Child Elements:

* request

#### request

The request element is used to configure each known game request. For example, if a game supports requests such as “bet”, “split”, “double”, “stand”, then there would be one <request> element included for each.

|  |  |
| --- | --- |
| Attribute | Description |
| Name | The name of the request. This name must match the last part of the request URL format: http://<domain>/<gameName>/**<requestName>** |

Child Elements:

* randomPools
* historicalState
* persistentState

#### randomPools

The randomPools element is collection of <randomPool> elements. A game can define a number of random pools per request.

Child Elements:

* pool

#### pool

A pool element defines a named pool of random numbers. Each pool is identified by a name, and specifies how many random numbers should be generated and what range those numbers should be in.

|  |  |
| --- | --- |
| Attribute | Description |
| Id | This is the identity of the random pool. When this request is sent to the game, the random numbers available will be organized by these identifiers. |
| Size | The number of random numbers to include in this pool. |
| min | The smallest number that should possibly be returned by this pool. |
| max | The largest number that should possibly be returned by this pool. |
| increment | (*Optional*) The amount that the *max* pool size will be incremented on each drawn number after the first. This number may be a positive (increment) or negative (decrement) value. |

#### historicalState

This element configures the request to include historical state. Historical state can be expensive to retrieve, so it should only be included if absolutely necessary. Historical state is available across game sessions and can be shared between players (e.g. for an in-game progressive jackpot feature).

|  |  |
| --- | --- |
| Attribute | Description |
| Required | This is a “true” or “false” setting to determine if historical state should be returned. |

#### persistentState

This element configures the request to include persistent state. Many games will require persistent state, but if a certain request does not need it, then it can be disabled to improve performance of this request. Persistent state is available across sessions for the same user.

|  |  |
| --- | --- |
| Attribute | Description |
| required | This is a “true” or “false” setting to determine if persistent state should be returned. |

#### gameError

This element contains critical error messages in the event the Remote Game could not respond appropriately to the initialize\_game request.

## replay\_game

The replay\_game request is responsible for transforming/rendering a replayed game state object (usually XML) into a human readable HTML form.

### Request URL

http://<domain>/<game\_name>/replay\_game

### Request Body

1. <!-- Sent to http://.../<gameName>/replay\_game -->
2. **<gdkRequest>**
3. **<gameRequest>**
4. <!--
5. Replay Data from AGS
6. -->
7. **</gameRequest>**
8. **</gdkRequest>**

The body of the replay\_game request will contain the original game play data, including the request, response, accumulator state and other game results. This data can be quite large, so rather than provide an example here, one can be provided by Amaya in a separate file.

### Response Body

1. **<table>**
2. **</table>**

The response to the replay\_game request can be almost any XHTML content that can be rendered. The only constraint is that this content must be rendered inside of an XHTML <table> tag, and therefore appropriate <tr> and <td> tags must be used to create well formed XHTML.

## resume\_game

The resume game request is used to restore a game to a playable state in the event that the game was interrupted (e.g. power loss), before it was able to complete.

### Request URL

http://<domain>/<game\_name>/resume\_game

### Request Body

1. **<gdkRequest>**
2. **<gameHeader>**
3. **<gameState>**
4. **</gameState>**
5. **<persistentState>**
6. **</persistentState>**
7. **</gameHeader>**
8. **</gdkRequest>**

The body of the resume game state includes only the gameHeader element, that includes both the gameState for the resumed game session, as well as the persistent state for that game and user. Game state and persistent state are both game specific. The gameState element must have only one child as a root for all data elements you store in the game state. Examples can be provided by Amaya.

### Response Body

1. **<gdkResponse version=”2”>**
2. **<gameResponse>**
3. **<responseToClient>**
4. **<msgdata** reqName="ResumeGameReq"**>**
5. <!-- game specific response here -->
6. **</msgdata>**
7. **</responseToClient>**
8. **</gameResponse>**
9. **<gameError></gameError>**
10. **</gdkResponse>**

The response to the resume\_game response is almost exactly like that of a normal game response, but does not change the game state (instead assuming the current resume state). Also, the resume\_game response cannot contain financial instructions.

## gaming\_guide

The gaming\_guide request returns the game rules for the game in any of the supported languages.

### Request URL

http://<domain>/<game\_name>/gaming\_guide?lang=<language>

|  |  |
| --- | --- |
| Parameter | Description |
| lang | ISO 639-1 (2 character) language code. |

### Response Body

1. **<html>**
2. **<head>**
3. **<title>**A Gaming Guide**</title>**
4. **</head>**
5. **<body>**
6. **<h2>**Gaming Guide**</h2>**
7. **<img** src="images/blank.png" **/>**
8. **</body>**
9. **</html>**

The gaming guide can be any valid and complete HTML. Any resource may be referenced, as long as the resource exists relative to the root of the web application (see the image example above).

There are a few rules that must be followed:

1. The HTTP contentType must have the proper character encoding set. The character set used for the gaming guide text, must be reflected in the contentType. For example, english guides that used UTF-8, the contentType will be: text/html;charset=utf-8.
2. Resources (images, audio, CSS, etc) that are referenced in the Gaming Guide, must be relatively referenced.

## about\_info

About info is a feature that will evolve over time to support fine grained game information. This information can be used by other systems to generate content based on game details. Currently only RTP values are supported.

### Request URL

http://<domain>/<game\_name>/about\_info?level=<rtp\_level>&lang=<language>

|  |  |
| --- | --- |
| Parameter | Description |
| level | The level can be set to 1, 2, 3 or 4. Values 1 through 3 are the normal RTP levels supported by the game, ordered by lowest to highest. For example 1 = 92%, 2 = 95%, 3=97% -- the RTP for your game may be different, or may only support one level. The fourth RTP level is reserved for games that are configured with a progressive jackpot. A progressive jackpot contribution will increase the RTP, so this level is intended to reflect a lower RTP because the combined RTP with the jackpot will be higher. For example, RTP level 4 may be 91.5% to achieve an overall game + jackpot RTP of 95%. Again, your RTP levels may be different, and you may only support one level, and may or may not support a progressive jackpot. |
| lang | ISO 639-1 (2 character) language code. |

### Response Body

1. **<about>**
2. **<game** name="MyGameName" type="0" desc="My Game Name"**>**
3. **<rtp>**
4. **<level** id="1"**>**92%**</level>**
5. **</rtp>**
6. **</game>**
7. **</about>**

#### game

The game element contains attributes that describe basic game details.

|  |  |
| --- | --- |
| Attribute | Description |
| name | The system game name (matches the URL and game name in the game descriptor. |
| description | The human readable display name for the game. |
| type | Legacy. This may be removed in the future, but for now, it’s a required field that only allows for a value of 0. |

#### level

The body of the level element contains a string representing a descriptive Return to Player percentage (e.g. 95%).

|  |  |
| --- | --- |
| Attribute | Description |
| id | The RTP level. This should match the input level parameter. |

## get\_profile\_parameters (New Profile)

Get Profile Parameters is a call made by CB5 to the game to get the information about game profile.

### Request URL

http://<domain>/<game\_name>/get\_profile\_parameters?lang=<language>

|  |  |
| --- | --- |
| Parameter | Description |
| lang | ISO 639-1 (2 character) language code. |

### Response Body

1. **<profileParams>**
2. **<paramAttribute** name="GameParam1"**>**
3. **<displayName>**Game Param 1**</displayName>**
4. **<description>**Describing Game Param 1**</description>**
5. **<value>** Default value for Game Param 1**</value>**
6. **</paramAttribute>**
7. **</profileParams>**
8. <!—Game specific example Empty new profile -->
9. <profileParams>
10. <paramAttribute name="coinValue">
11. <displayName>Coin Value</displayName>
12. <description>One/more coin value delimited by comma
13. (e.g. '1.00,2.00,5.00,10.00,50.00,....')</description>
14. <value />
15. </paramAttribute>
16. <paramAttribute name="maxGambleWinnings">
17. <displayName>Maximum Gamble Winnings</displayName>
18. <description>The maximum a player can win through a gamble (e.g.'100000.00')</description>
19. <value />
20. </paramAttribute>
21. </profileParams>

#### paramAttribute

The paramAttribute element contains id attribute.

|  |  |
| --- | --- |
| Attribute | Description |
| name | Unique ID for the profile parameter |

#### displayName

The body of the displayName element contains parameter name displayed by CB5 on the profile page (e.g. Coin Value).

#### description

The body of the description element contains detail about the parameter usage and valid values

(e.g. One/more coin value delimited by comma (e.g. '1.00,2.00,5.00,10.00,50.00,....'))

#### value

The body of the value element contains default values for profile parameters and can be returned empty as well.

## get\_profile\_parameters (Copy Profile)

Get Profile Parameters is a call made by CB5 to the game to copy any existing profile into a new one.

### Request URL

http://<domain>/<game\_name>/get\_profile\_parameters?lang=<language>

|  |  |
| --- | --- |
| Parameter | Description |
|  |  |
| lang | ISO 639-1 (2 character) language code. |
| param | Posted Parameter whose value is profile to be copied. |

### POSTED Parameter : ‘param’

1. **<param** GameParam1="Value for GameParam1" GameParam2="Value for GameParam2"/**>**

<!— Game specific example -- >

1. <param coinValue="0.10, 1.00, 2.00, 5.00" maxGambleWinnings="10000.00" />

#### param

The param element contains the game specific profile parameters as attributes.

### Response Body

1. **<profileParams>**
2. **<paramAttribute** name="GameParam1"**>**
3. **<displayName>**Game Param 1**</displayName>**
4. **<description>**Describing Game Param 1**</description>**
5. **<value>** Value copied over from profile passed in**</value>**
6. **</paramAttribute>**
7. **</profileParams>**
8. <!—Game specific example Copy profile -->
9. <profileParams>
10. <paramAttribute name="coinValue">
11. <displayName>Coin Value</displayName>
12. <description>One/more coin value delimited by comma
13. (e.g. '1.00,2.00,5.00,10.00,50.00,....')</description>
14. <value>0.10,1.00,2.00,5.00</value>
15. </paramAttribute>
16. <paramAttribute name="maxGambleWinnings">
17. <displayName>Maximum Gamble Winnings</displayName>
18. <description>The maximum a player can win through a gamble (e.g.'100000.00')</description>
19. <value>10000.00</value>
20. </paramAttribute>
21. </profileParams>

#### paramAttribute

The paramAttribute element contains id attribute.

|  |  |
| --- | --- |
| Attribute | Description |
| name | Unique ID for the profile parameter |

#### displayName

The body of the displayName element contains parameter name displayed by CB5 on the profile page (e.g. Coin Value).

#### description

The body of the description element contains detail about the parameter usage and valid values.

(e.g. One/more coin value delimited by comma (e.g. '1.00,2.00,5.00,10.00,50.00,....'))

#### value

The body of the value element contains the value of the parameter extracted from the profile sent in as ‘param’.

## validate\_profile\_parameters

Validate Profile Parameters is a call made by CB5 once the user fill the profile values and click ‘Save’. The game should validate the values entered and send back error messages if any or the game returns the exact string to be inserted into db for profile.

### Request URL

http://<domain>/<game\_name>/validate\_profile\_parameters?lang=<lang>

|  |  |
| --- | --- |
| Parameter | Description |
| profileParams | Posted Parameter whose value is profile to be validated |
| lang | ISO 639-1 (2 character) language code. |

Request sent in is exactly what the response was sent for get\_profile\_parameters with all the ‘paramAttribute’ elements and each one with ‘value’ in it.

### POSTED Parameter: ‘profileParams’

1. **<profileParams>**
2. **<paramAttribute** name="GameParam1"**>**
3. **<value>** Entered Value for Game Param 1**</value>**
4. **</paramAttribute>**
5. **< paramAttribute**  name="GameParam2"**>**
6. **<value>** Entered Value for Game Param 2**</value>**
7. **</paramAttribute>**
8. **</profileParams>**
9. <!—Game specific example -- >
10. <profileParams>
11. <paramAttribute name="coinValue">
12. <value>0.10,1.00,2.00,5.00</value>
13. </paramAttribute>
14. <paramAttribute name="maxGambleWinnings">
15. <value>10000.00</value>
16. </paramAttribute>
17. </profileParams>

Response will differ based upon if there were any validations errors or not.

### Response Body (No Validation Errors)

1. **<response** status="ok"**>**
2. **<param** GameParam1="Value for GameParam1" GameParam2="Value for GameParam2"/**>**
3. **</response>**
4. **<response** status="ok"**>**
5. **<param** coinValue="0.10, 1.00, 2.00, 5.00" maxGambleWinnings="10000.00" /**>**
6. **</response>**

#### response

The response element contains status attribute.

|  |  |
| --- | --- |
| Attribute | Description |
| status | Ok (means no validation errors) |

The game can specify any attributes in the ‘param’ element.

### Response Body (With Validation Errors)

1. **<response** status="validationErrors"**>**
2. **<errors>**
3. **<paramAttribute** name="GameParam1"**> Descriptive error message** **</param>**
4. **<errors>**
5. **</response>**
6. **<response** status="validationErrors"**>**
7. **<errors>**
8. **<paramAttribute** name="coinValue"**>**Coin value entered is not a number[1.AA]**</param>**
9. **<paramAttribute** name="maxGambleWinnings"**>**Only 2 digits allowed after decimal[10000.005]**</param>**
10. **<errors>**
11. **</response>**

#### response

The response element contains status attribute.

|  |  |
| --- | --- |
| Attribute | Description |
| status | validationErrors |

#### paramAttribute

The paramAttribute element contains name attribute.

|  |  |
| --- | --- |
| Attribute | Description |
| name | Unique ID for the profile parameter |

The body of the paramAttribute element contains a descriptive error message (e.g. Coin value entered is not a number[1.AA])

## Other Game Specific Requests

In addition to the standard requests that must be implemented, your game will no doubt have a number of request types specific to the game itself – otherwise it would be a pretty boring game.

Game specific requests have standard elements and attributes that provide management of game state and instructions to the Amaya Gaming System. However, there are sections reserved for data specific to your game that may use any suitable structure depending on the needs of your game.

### Request URL

http://<domain>/<game\_name>/<request\_name>

### Request Body

1. **<gdkRequest>**
2. **<gameHeader>**
3. **<player** id="" **/>**
4. **<gameState>**
5. <!-- This content is specified by the Game -->
6. **<myGameState** **/>**
7. **</gameState>**
8. **<historicalState>**
9. <!-- This content is specified by the Game -->
10. **<myHistoricalState** **/>**
11. **</historicalState>**
12. **<persistentState>**
13. <!-- This content is specified by the Game -->
14. **<myPersistentState** **/>**
15. **</persistentState>**
16. **<profile** profileId="" accId="" currCode="" currScale="" chatChannelId="" pjEnabled="" mpjEnabled="" **>**
17. **<param** **/>**
18. **</profile>**
19. **<randomPools>**
20. <!-- Random pools of comma delimited numbers, identified as configured -->
21. **<pool** id=""**>**2,13,4,5,23,45,1,2,34,22,10**</pool>**
22. **<pool** id=""**>**1,34,4,5,23,45,1,2,34,22,10**</pool>**
23. **</randomPools>**
24. <!-- Defines if this request is a free game action -->
25. **<freeGameMode></freeGameMode>**
26. **</gameHeader>**
27. **<gameRequest>**
28. <!-- Game Request Sent by Client -->
29. **<msgdata** reqName=""**>**
30. <!-- Game specific defined data -->
31. **</msgdata>**
32. **</gameRequest>**
33. **</gdkRequest>**

#### gameState

The game state element contains state information that is relatively volatile and is meant to be used from request to request. It is not preserved across game sessions, but may be resumed. The body of the gameState element is game specific, and thus can be anything.

Because this is the request, this state represents the last state set by the last response received from the Remote Game. The Amaya Gaming System acts as the storage mechanism for game state.

#### persistentState

The persistentState element contains state information that is persisted across game sessions. Thus it is useful for things such as accumulated results for game features that should preserve date for the player between gaming sessions. The body of the persistentState element is game specific, and thus can be anything.

Because this is the request, this state represents the last state set by the last response received from the Remote Game. The Amaya Gaming System acts as the storage mechanism for game state.

#### historicalState

The historicalState element contains state information that is persisted across game sessions and is shared between all players of that game. This is useful for persistent features shared between players, such as the state of an in-game progressive jackpot. The body of the historicalState element is game specific, and thus can be anything.

Because this is the request, this state represents the last state set by the last response received from the Remote Game. The Amaya Gaming System acts as the storage mechanism for game state.

#### profile

The profile element contains a single <param> element that matches that which is defined by the Game Profile. The attributes of the <param> element will match the XSD defined in the game descriptor (see the get\_game\_descriptor request above).

|  |  |
| --- | --- |
| Attribute | Description |
| profileId | This is the numeric identifier of the game profile described in this element. |
| acctId | This is the system account ID that the player is currently playing against. In the Amaya Gaming System, an account can be thought of as a “configured currency”. Bets and wins from this game play activity will be rolled up under this account. |
| currCode | This represents the currency that is assigned to the acctId. Its value is the ISO standard Currency Code associated with the acctId. |
| currScale | This represents the currency scale that is associated with the currency code given. The currency scale should be taken into consideration when rounding financial values. |
| chatChannelId | If the game supports a chat feature, this is the chat channel ID that should be used. |
| pjEnabled | If the game supports Progressive Jackpots. |
| mpjEnabled | If the game supports Mystery Progressive Jackpots. |

#### randomPools

The random pools are named sets of random numbers as configured in the initialize\_game response described above. Each pool child element is identified by the ID that was configured in initialize\_game, as well as the number of random numbers in the ranges specified. The game code can refer to these pools by name, and sequentially draw upon the numbers to preserve the order in which the random number generator created them.

#### freeGameMode

Free games/spins are a promotional tool. This node is only valid for games that are designed to support free games.

#### gameRequest

The game request element contains data sent from the Flash Client. The structure and data contained within this element is game specific, and is not manipulated by the Amaya Gaming System. It is as it was sent from the Flash Client. Because it is game specific, it can be almost anything, but is usually represented as XML.

The game specific data must be enclosed within the <msgdata> element.

|  |  |
| --- | --- |
| Attribute | Description |
| reqName | This is the name of the incoming request. This request name will be used to complete the request URL. |

### Response Body

1. **<gdkResponse version=”2”>**
2. **<gameState>**
3. <!-- This content is specified by the Game -->
4. **<myGameState** **/>**
5. **</gameState>**
6. **<historicalState>**
7. <!-- This content is specified by the Game -->
8. **<myHistoricalState** **/>**
9. **</historicalState>**
10. **<persistentState>**
11. <!-- This content is specified by the Game -->
12. **<myPersistentState** **/>**
13. **</persistentState>**
14. **<gameResponse>**
15. <!-- Game Response: Includes GDKInstructions, ClientResponse, etc -->
16. **<instructions>**
17. <!-- This part is game specific and will hold various gdk instructions -->
18. **<financial>**
19. **<transaction** playerId="1000" handId="1" action="1" amount="100" **/>**
20. **</financial>**
21. **</instructions>**
22. **</infoOnly>** <!—only define infoOnly if the request does not need to be saved (rare)-->
23. **<responseToClient>**
24. **<msgdata reqName=””>**
25. <!-- game specific response here -->
26. **</msgdata>**
27. **</responseToClient>**
28. **</gameResponse>**
29. **<gameError>**
30. <!--
31. Error Message. Only defined if there is an error/exception,
32. that the GDK command do not handle, and an exception is required to be thrown in the AGS
33. -->
34. **</gameError>**
35. **</gdkResponse>**

The response to the game requests is a similar format to the request. It includes the gameState, persistentState and historicalState elements. The obvious difference is that because this is the response, it should represent the state that you wish to preserve. This state is what will be returned in the next game request in the elements described above in the request body section.

In addition to game state manipulation, the response contains instructions to the AGS to manipulate the player’s balance.

#### transaction

The transaction element is a financial instruction used to manipulate the player balance based on the outcome of game rules.

|  |  |
| --- | --- |
| Attribute | Description |
| playerId | The ID of the player currently playing |
| handId | (*Optional*) The hand index that is being played. This has more relevance to games such as blackjack, where a player can play multiple hands at the same time. |
| action | The type of transactional action to perform. Valid parameters:   * “*bet*” or “*1*” A debit to the player’s account. * “*win*” or “*2*” A net-win credit to the player’s account. Positive/Negative amounts allowable. * “*return*” or “*3*” A credit to the player’s account. Positive amounts only. |
| amount | The amount of the action (bet, win or return) |

#### infoOnly

The infoOnly element can be used to indicate to the AGS that a request which contains this element does not need to be saved into the game state. The AGS does not save this request in the game state and so therefore this request cannot be viewed in replay or used to resume a game but can be used to send information along to the client. An example in which this request could be used is sending the reels for the game to the client so that spins can use the correct ordering of the reels without having to store this information directly in the client code.

#### responseToClient

The responseToClient element is a game specific section that is used to communicate with the Flash Client. Recall that in the request body, the Flash Client was able to communicate with the server via the gameRequest element. The responseToClient element is the other side of that communication. Thus, between these two elements, is the means through which the client and server communicate with each other.

The game response data must be enclosed within the <msgdata> element. The body of the element is entirely game specific, and thus can contain anything. However it is usually represented as XML.

|  |  |
| --- | --- |
| Attribute | Description |
| reqName | This is the name of the request that was responded to. |

#### gameError

The gameError element can return details regarding any errors that occurred during the processing of the request. It should be reserved for severe system errors, not for normal game play errors that are recoverable.

## manifext.xml

The manifest file is not a game service (but could be), rather it is a descriptor file for the Flash Game Client that contains a list of all of the files required by the game. This is necessary because we do not allow directory browsing or other means to determine which files the game needs to load. This approach is faster and more secure.

The manifest file can be manually created. However, if you use the Ant build script provided by Amaya, it will create the manifest for you by automatically including all files in the client/web/game directory. All files listed in the manifest will be preloaded by the Casino 4.0 framework during the loading phase of the game. These files can then be accessed by the game client using the GameServices class.

### Request URL

http://<domain>/<game\_name>/manifest.xml

### Response Body

1. **<manifest** type="game"**>**
2. **<file>**game/controller/game.swf**</file>**
3. **<file>**game/sound/game\_sounds.swf**</file>**
4. **<file>**game/ui/3dcards.swf**</file>**
5. **<file>**game/ui/game\_ui.swf**</file>**
6. **</manifest>**

#### manifest

The manifest element contains a number of file child elements, each which describe a relative file path to an asset required by the game client. All paths are relative to the root directory of the game web application.

|  |  |
| --- | --- |
| Attribute | Description |
| type | For most games, this should always be set to “game”. |

## logo.png

The logo image file is the logo image in PNG graphic format for the game.

### Request URL

http://<domain>/<game\_name>/logo.png

### Response

The PNG image as the game’s logo

## Figure 3. High Level Deployment Scenario

