Vanguard GGi Interface

Distribution: All Clients

Version: 23.00
Document Number: 008-419

18 January 2016

Contents

Figures

Introduction

This document provides an overview of the methods that the Generic Gaming Interface (GGi) API must support to enable the Microgaming gaming system to authenticate a player and update the player's balance.

Related Documents

- 008-591 - Poker Vanguard Integration
- 008-163 - Vanguard Generic API Developer Guide
- 009-019 - Orion Third Party Integration for Quickfire

Expected API Architecture

API Process Flow

Figure 1 - API Process

Method Invocation via Vanguard

The Microgaming (MGS) system uses XML to communicate with your API via an HTTPWebRequest. The structure of this XML is defined in the document. The XML changes according to the method the service invokes.

XML Packet Specifications

All packets to and from the Microgaming system must be wrapped in a *pkt* tag as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<pkt>
</pkt>
```

Method Request

The request parameters are wrapped in the method call XML element. All requests that are sent to the API should be authenticated. The authentication credentials are contained in the **auth** element. Method-specific parameters are contained in the call XML element and these could change depending on the method. The **seq** and **token** attributes exist in all packets.

```
<pkt>
  <methodcall name="login" timestamp="2011/01/18 14:33:03.000"
system="casino">
    <auth login="test" password="test" />
    <call seq="24971455-aecc-4a69-8494-f544d49db3da"
          token="someothertoken" />
  </methodcall>
</pkt>
```

Method	Description
methodcall	The **methodcall** XML element indicates that the packet is a request.
name	The **name** attribute identifies the method name that is called.
timestamp	The **timestamp** attribute represents the date and time that Vanguard sent the request, in the format: *year/month/day hour:minute:second.millisecond* The time zone may vary depending on the hosting location.
system	The **system** attribute identifies the gaming system that the game is played on. Possible values include **casino** and **poker**. Currently, bingo games are identified as casino, but in a future update the system field will identify bingo as its own system.
auth	The **auth** XML element contains the API login credentials.
login	The credentials for API authentication. You must send this information to Microgaming prior to integration testing.
password	The credentials for API authentication. You must send this information to Microgaming prior to integration testing.
call	The **call** XML element contains the parameters required for method execution on your system.
seq	The **sequence** attribute is generated by the Microgaming system and uniquely identifies the request. This value should be handled as a string with a maximum length of 100 characters.

Method	Description
token	The **token** is generated by your system and passed in from the Microgaming client. Your system is responsible for authenticating the player.
clienttypeid	This indicates the client that triggered the login request. The field is optional and is only included if configured by Microgaming. Possible values are: • 1 - Flash • 5 - Viper • 38 - Android APK • 40 - Mobile
extinfo	The **extinfo** element is used to pass data between the Microgaming client and your system.

Method Response

Every method call must return a result XML packet that is wrapped in the **methodresponse** element.

```
<pkt>
     <methodresponse name="login" timestamp="2011/01/18 14:33:00.000">
      <result seq="24971455-aecc-4a69-8494-f544d49db3da" token="2fd9f9d4-012a-
      4b54-a4a9-6b3bf78d73bb" loginname="someothertoken" currency="USD"
      country="" city="" balance="0" bonusbalance="0">
            <extinfo/>
      </result>
     </methodresponse>
</pkt>
```

Method	Description
methodresponse	This XML element indicates that the packet is a response. This element is mandatory.
name	The **name** attribute identifies the method call and must match the corresponding request. This attribute is mandatory.
timestamp	The **timestamp** attribute represents the date and time the API sent the response, in the format: *year/month/day hour:minute:second.millisecond* This must be in UTC. This attribute is mandatory.

Method	Description
result	This XML element contains the result of the method execution. The attribute list will change depending on the method call. This element is mandatory.
seq	The **sequence** attribute must match what was sent in the corresponding request. This value should not be interpreted, as the type can change. This is mandatory.
token	The **token** attribute is mandatory and must always contain a valid token that uniquely identifies the player. For more information, refer to the Token Management section.
errorcode	The **errorcode** attribute contains a code associated with the error. In some cases the GGi interface defines these codes, otherwise your system defines them. Where the error code is defined by your system, the code can be represented by a string value. For a list of GGi defined codes, see the Error Codes section. If the method execution did not result in an error, then this attribute should be excluded. The error code value must have a maximum length of 10 characters.
errordescription	The **errordescription** attribute contains a description of the error. If the method execution did not result in an error, then this attribute should be excluded. The error description value has a maximum length of 255 characters.

Process Flows

Game Launch

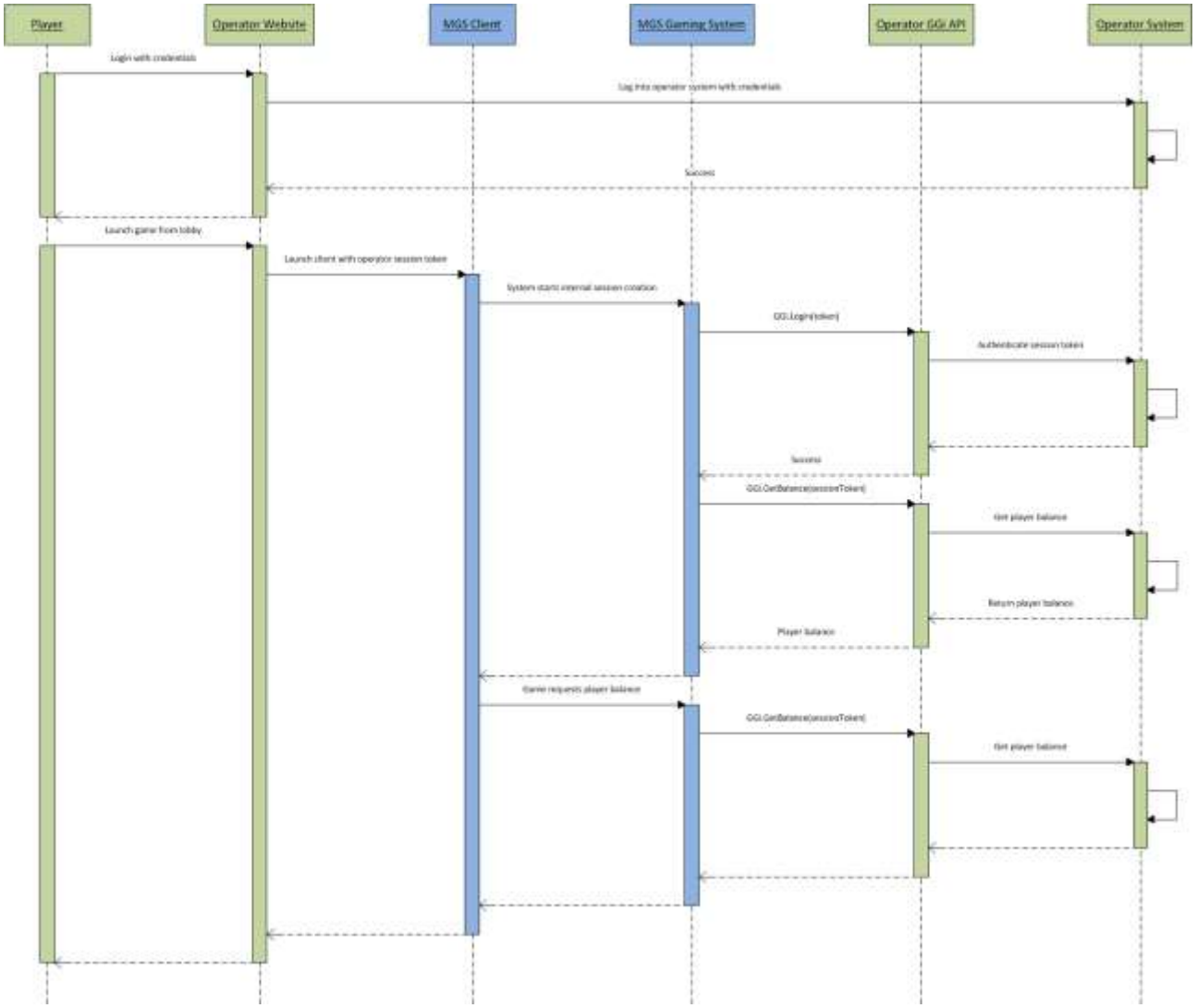

Figure 2 - Game Launch Process

Gameplay

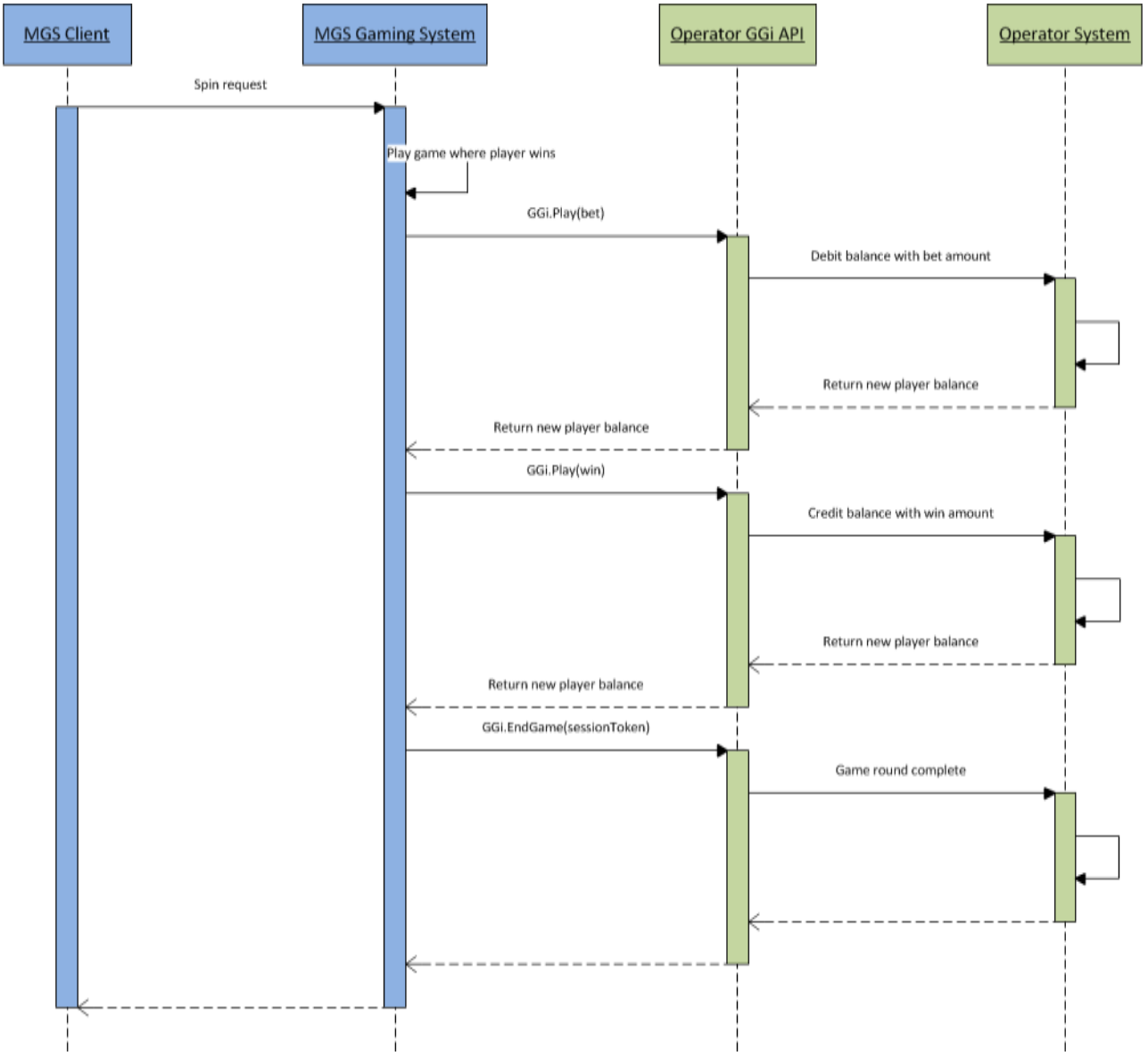

Figure 3 - Gameplay Process

Method Schemas and Descriptions

Login

The **Login** method is used to authenticate players. The authentication method varies depending on the game client that is used. The method supports both the token-based and user credential (log in name and password) authentication models.

The method returns the player's details, if the player is successfully authenticated.

Request

```
<pkt>
  <methodcall name="login" timestamp="2011/01/18 14:33:00.000"
system="casino">
    <auth login="" password="" />
    <call seq="24971455-aecc-4a69-8494-f544d49db3da"
          token=" AAD8EE30-8C43-11DC-9755-668156D89593">
      <extinfo/>
    </call>
  </methodcall>
</pkt>
```

The following attributes are additional to the standard attributes described in the Expected API Architecture section.

Method	Description
loginname	If the client does not support token-based authentication, it is possible to authenticate the player using their login name and password. This attribute is optional and depends on the client authentication model.
password	The **password** attribute represents the player's password and is used only if the client does not support the token-authentication model. This is mandatory only when used in conjunction with the **loginname**.
ipaddress	The **ipaddress** attribute represents the player's IP address. It is not included in all requests as currently only the Microgaming Poker client supports this attribute.

Success Response

```
<pkt>
  <methodresponse name="login" timestamp="2011/01/18 14:33:00.000">
    <result seq="24971455-aecc-4a69-8494-f544d49db3da"
            token="2fd9f9d4-012a-4b54-a4a9-6b3bf78d73bb"
            loginname="johnc"
            currency="USD"
            country="USA"
            city="NY"
            balance="100"
            bonusbalance="0"
            wallet="vanguard">
      <extinfo/>
    </result>
  </methodresponse>
</pkt>
```

Success Response DTD

```
<!DOCTYPE pkt [
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (result)>
<!ATTLIST methodresponse name (login) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (responsiblegaming*, regulatedmarket*, extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result token CDATA #REQUIRED>
<!ATTLIST result loginname CDATA #REQUIRED>
<!ATTLIST result currency CDATA #REQUIRED>
<!ATTLIST result country CDATA #REQUIRED>
<!ATTLIST result city CDATA #REQUIRED>
<!ATTLIST result balance CDATA #REQUIRED>
<!ATTLIST result bonusbalance CDATA #REQUIRED>
<!ATTLIST result wallet CDATA #IMPLIED>
<!ELEMENT responsiblegaming (add+)>
<!ELEMENT add ANY>
<!ATTLIST add key CDATA #REQUIRED>
<!ATTLIST add value CDATA #REQUIRED>
<!ELEMENT regulatedmarket (add+)>
<!ATTLIST add key CDATA #REQUIRED>
<!ATTLIST add value CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

Method	Description
loginname	The **loginname** attribute is mandatory and represents the unique identifier for the player in your system. The Microgaming system limits the length of the login name to 17 characters. This is handled as a string value in the Microgaming system. If you are required to support Microgaming's Generic SAFE API for regulated markets, it is essential that the returned **loginname** is the player's unique identifier assigned by your SAFE.
currency	The **currency** attribute is mandatory and must match your wallet currency. This value cannot change between player logins. You must use the relevant ISO currency code, such as **GBP** for United Kingdom Pound.
country	The **country** attribute is mandatory although the value can be left blank. You must use the ISO country code.
city	The **city** attribute is mandatory although the value can be left blank.
balance	The **balance** attribute is mandatory and must represent the player's real time balance. It is essential that this value is always in cents. It is handled as an Int32 in the Microgaming system.
bonusbalance	The **bonusbalance** attribute is mandatory. However, it is possible to default this to zero, as currently it is not supported by our casino interface.
wallet	The **wallet** attribute is an optional parameter used to indicate where the player's balance is kept. Only the Microgaming Poker client supports this attribute. Current values are **vanguard** or **local**.

responsiblegaming	The **responsiblegaming** XML element is optional and can contain specific player protection limits that are supported by the Microgaming system. You can add each limit to this element as a key-value pair.
	For example:
	`<add key="sessiondurationreminder" value="20" />`
	`<add key="sessiontimeout" value="60" />`

Method	Description
regulatedmarket	The **regulatedmarket** XML element is optional and can contain information specific to a regulated market. You can add additional information to this element as key-value pairs. The known keys are: **All Regulated Markets** • **idnumber** The player's unique identifier defined by their government. This value is handled as a string in the Microgaming system. This attribute is optional. **United Kingdom Regulated Market** • **status** The possible statuses returned are: – **Provisional** ▪ The player's age is not yet verified in the required period after registration. **Note:** This period is 72 hours for the United Kingdom regulated market. ▪ The player can log in to the casino and play games. – **Active** ▪ The player's age is successfully verified. ▪ The player can log in to the casino and play games. • **remainingprovisionaltimeinsec** The time, in seconds, remaining for the Provisional account status. **Note:** A value of **-1** denotes that the timer has not started. If this field is not returned, the system assumes that the timer has not started and performs accordingly.
type	The **type** attribute in the **regulatedmarket** element defines the regulated market type. Typical values are **Denmark**, **Spain** or **UK**. This attribute is mandatory when the **regulatedmarket** element is included.
extinfo	The **extinfo** attribute is reserved for client- or system-specific information. This is currently only used in the context of the Poker system integration. For more information on extended information, see the **Login** section of the **008-591 - Poker Vanguard Integration** document.

Failure Response

```
<pkt>
  <methodresponse name="login" timestamp="2011/01/18 14:33:00.000" >
    <result seq="24971455-aecc-4a69-8494-f544d49db3da"
            errorcode="6102"
            errordescription=" Account is Locked">
      <extinfo/>
    </result>
  </methodresponse>
</pkt>
```

Failure Response DTD

```
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (login)>
<!ATTLIST methodresponse name (play) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result errorcode CDATA #REQUIRED>
<!ATTLIST result errordescription CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

GetBalance

The **GetBalance** method is used to retrieve a player's real time balance.

This method is called after the player has logged in and normally before every bet. Depending on the game that is played, there is a possibility of multiple bets with a single **GetBalance** call

Request

```
<pkt>
  <methodcall name="getbalance" timestamp="2011/01/19 15:31:00.000"
system="casino">
    <auth login="test" password="test" />
    <call seq="d8382c1d-6ce0-433d-9233-e7358b10a70b"
          token="7699e837-3488-41f2-8ce5-f2816f633339">
      <extinfo />
    </call>
  </methodcall>
</pkt>
```

Success Response

```
<pkt>
  <methodresponse name="getbalance" timestamp="2011/01/19 15:31:00.000">
    <result seq="d8382c1d-6ce0-433d-9233-e7358b10a70b"
            token="9800710f-51e1-4906-aa85-641bfaf41ef3"
            balance="25000"
            bonusbalance="0">
      <extinfo />
    </result>
  </methodresponse>
</pkt>
```

Success Response DTD

```
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (result)>
<!ATTLIST methodresponse name (getbalance) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result token CDATA #REQUIRED>
<!ATTLIST result balance CDATA #REQUIRED>
<!ATTLIST result bonusbalance CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

Method	Description
balance	The **balance** attribute is mandatory and must represent the player's real time balance. It is essential that this value is always in cents. It is handled as an Int32 in the Microgaming system.
bonusbalance	The **bonusbalance** attribute is mandatory. However, it is possible to default it to zero as it is not currently supported by our casino interface.

Failure Response

```
<pkt>
  <methodresponse name="getbalance" timestamp="2011/01/19 15:31:00.000">
    <result seq="d8382c1d-6ce0-433d-9233-e7358b10a70b"
            errorcode="6508"
            errordescription="The player exceeded their game play duration.">
      <extinfo />
    </result>
  </methodresponse>
</pkt>
```

Failure Response DTD

```
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (getbalance)>
<!ATTLIST methodresponse name (play) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result errorcode CDATA #REQUIRED>
<!ATTLIST result errordescription CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

Play

The **Play** method is used for any balance updates. The **playtype** attribute defines the type of balance update. It is possible to have multiple bets and wins in a single game round depending on the game type, such as Blackjack.

Any play types that credit a player's balance can be sent as offline transactions. For example, **win**, **progressivewin**, **transferfrommsg**, **admin** and **refund**. Support must be provided for bingo, poker and multi-player tournaments where this behaviour is prevalent.

Request

```
<pkt>
  <methodcall name="play" timestamp="2011/01/19 14:53:00.000" system="casino">
    <auth login="test" password="test" />
    <call seq="df829722-9826-4540-8a58-1d1c0feada93"
          playtype="bet"
          token="96b1f112-cd21-4e9e-a29a-75a1e454d7df"
          gameid="20"
          gamereference="MGS_Thunderstruck"
          actionid="12158"
          actiondesc=""
          amount="225"
          start="false"
          finish="false"
          offline="false"
          currency="zar"
          freegame="freegameoffer1">

      <extinfo />
    </call>
  </methodcall>
</pkt>
```

Method	Description
seq	The **sequence** code generated by Vanguard. This must be sent back in the response.
token	The **token** generated by your system that is used for authentication. This is also used to send an offline transaction identifier.
playtype	The **playtype** attribute indicates the type of balance update. This may differ across the various Microgaming game platforms. For more information, see the Gaming Platform Types section. The possible values for this attribute are: • **bet** Debit the player's balance. • **win** Credit the player's balance. • **progressivewin** Credit the player's balance. • **refund** Credit the player's balance. This is triggered by the rollback queue. • **transfertomgs** Debit the player's balance. • **transferfrommgs** Credit the player's balance. • **tournamentpurchase** Debit the player's balance. • **admin** Debit or credit the player's balance. The **actiondesc** attribute provides more details for this balance update.
gameid	The **gameid** attribute identifies the player's game round. The **gameid** is used to link the various game events, such as bets and wins, to a single game round. In the case where a bet is rolled back, this value is re-used for the next game that the player triggers. You should not use this value to uniquely identify a transaction or game event. It is seen as a 64-bit integer.
gamereference	The **gamereference** attribute identifies the game that is played. For example, the identifier for a game, such as Thunderstruck, is **MGS_Thunderstruck**. Microgaming issues you with a list of all the game references and what game names they map to.

Method	Description
actionid	The **actionid** attribute uniquely identifies each game event, such as a bet or win, in the Microgaming system. This reference is linked to the **gameid** described above.
actiondesc	The **actiondesc** attribute can contain any additional information referring to the balance update. For example, tournament win or major for the progressive win. For poker-specific examples, see the Play section of the **008-591 – Poker Vanguard Integration** document.
amount	The **amount** attribute refers to the amount the player's balance is updated by. This value is always in cents.
start	The **start** attribute indicates if this is the first event of the game round. This is either **true** or **false**.
finish	The **finish** attribute indicates if the game round is complete. Currently, the casino system does not support it and it defaults to **false**. It is there for future use.
offline	The **offline** attribute is used to indicate an offline transaction. It is possible for the system to send a transaction when a player is not logged in. In this case, Vanguard is not aware of the player's token, so an offline transaction is sent. The offline parameter is set to **true** and an alternate value is sent in the token parameter. Depending on the Microgaming game platforms and **playtype**, this offline token may consist of different data. For more information, see the Gaming Platform Types section.
currency	The **currency** attribute is optional and used to indicate the currency of the transaction. Currently, only the Microgaming Poker system supports this attribute.
extinfo	The **extinfo** attribute is reserved for client- or system-specific information. This is currently only used in the context of the Poker system integration. For more information, see the Play section of the **008-591 - Poker Vanguard Integration** document.

Success Response

```
<pkt>
  <methodresponse name="play" timestamp="2011/01/19 14:53:00.000">
    <result seq="df829722-9826-4540-8a58-1d1c0feada93"
            token="847ab65e-82f9-4ee5-a7be-242a5e1d1b80"
            balance="24250"
            bonusbalance="0"
            exttransactionid="1763178465">
      <extinfo />
    </result>
  </methodresponse>
</pkt>
```

Success Response DTD

```
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (result)>
<!ATTLIST methodresponse name (play) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result token CDATA #REQUIRED>
<!ATTLIST result balance CDATA #REQUIRED>
<!ATTLIST result bonusbalance CDATA #REQUIRED>
<!ATTLIST result exttransactionid CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

Method	Description
balance	The **balance** attribute is mandatory and must represent the player's real time balance. It is essential that this value is always in cents. It is handled as an Int32 in the Microgaming system.
bonusbalance	The **bonusbalance** attribute is mandatory. However, it is possible to default it to zero as currently our casino interface does not support it.
exttransactionid	The **exttransactionid** attribute is mandatory and must uniquely identify the transaction onn your system. This includes **bet**, **win**, **refund**, **admin**, **transfertomgs** or **transferfrommgs**. This attribute must contain a value and the value is handled as a string in the Microgaming system. If a **refund** transaction is sent but the bet is not successfully processed in your system, we recommend the API responds successfully with an **exttransactionid** value of **DEBIT-NOT-RECEIVED**, as the transaction does not exist.

Failure Response

```
<pkt>
  <methodresponse name="play" timestamp="2011/01/19 14:53:00.000">
    <result seq="df829722-9826-4540-8a58-1d1c0feada93"
            errorcode="6503"
            errordescription="Player has insufficient funds.">
      <extinfo />
    </result>
  </methodresponse>
</pkt>
```

Failure Response DTD

```
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (result)>
<!ATTLIST methodresponse name (play) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result errorcode CDATA #REQUIRED>
<!ATTLIST result errordescription CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

EndGame

The **endgame** method is used to indicate when a particular game round is complete.

There are various endgame triggers:

- After the last bet or win in a game round.
- If a player has one or multiple transactions in the commit queue, an endgame is automatically sent once the last win for the game round is processed through one of the admin systems.
- When a game is auto-completed by the Microgaming system. Typically, this occurs sometime after the player has closed the client window or even logged out of your website.

Request

```
<pkt>
  <methodcall name="endgame" timestamp="2011/01/21 14:33:00.000"
system="casino">
    <auth login="test" password="test" />
    <call seq="633b0a18-3cd8-4e59-8c1b-16fc301a5f53"
          token="48546f57-68c8-46e0-9994-bd3451d5a5ee"
          gamereference="MGS_BigTop"
          gameid="18_2cf92c89-8b08-4e8e-ab41-709f7f622090">
      <extinfo />
    </call>

  </methodcall>
</pkt>
```

Method	Description
gamereference	The **gamereference** attribute identifies the game identifier.

Method	Description
gameid	The **gameid** attribute identifies the player's game round that closes.
offline	The **offline** attribute indicates if the **token** sent in the **endgame** call is generated from offline information. For more information about offline transactions, see the Play, Request and Gaming Platform Types, Offline Tokens sections.

Success Response

```
<pkt>
  <methodresponse name="endgame" timestamp="2011/01/21 14:33:00.000">
    <result seq="633b0a18-3cd8-4e59-8c1b-16fc301a5f53"
            token="07f926a0-156a-4da0-b70a-4107add1b39b"
            balance="25275"
            bonusbalance="0">
     <extinfo />
    </result>
  </methodresponse>
</pkt>
```

Method	Description
balance	The **balance** attribute is mandatory and must represent the player's real time balance. It is essential that this value is always in cents. It is handled as an Int32 in the Microgaming system.
bonusbalance	The **bonusbalance** attribute is mandatory. However, it is possible to default it to zero, as currently our casino interface does not support it.

Success Response DTD

```
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (result)>
<!ATTLIST methodresponse name (endgame) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result token CDATA #REQUIRED>
<!ATTLIST result balance CDATA #REQUIRED>
<!ATTLIST result bonusbalance CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

Failure Response

```
<pkt>
  <methodresponse name="endgame" timestamp="2011/01/21 14:33:00.000">
    <result seq="633b0a18-3cd8-4e59-8c1b-16fc301a5f53"
            errorcode="#"
            errordescription="">
      <extinfo />
    </result>
  </methodresponse>
</pkt>
```

Failure Response DTD

```
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (result)>
<!ATTLIST methodresponse name (endgame) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result errorcode CDATA #REQUIRED>
<!ATTLIST result errordescription CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

Refresh Token

It is recommended that the current token lifespan is extended when the **refreshtoken** call is made. This method provides a better user experience for our download client that can be open for extended periods of time.

The **refreshtoken** method activates when the system detects that the current token is two minutes from expiry. For more information on tokens, see the Token Management section.

The system keeps refreshing the current token until the inactivity threshold is reached. The threshold is typically set to twelve minutes, but is configurable at an operator level.

The token's lifespan determines the frequency of the **refreshtoken** method call - the longer the lifespan, the longer the interval between calls.

This method is not mandatory and can be disabled.

Request

```
<pkt>
  <methodcall name="refreshtoken" timestamp="2011/01/19 15:00:00.000">
    <auth login="test" password="test" />
    <call seq="f6fbaf29-19ca-4c9a-955b-709e11570f99"
          token="7205e83f-974c-48ab-881c-2f4ffdaa7e2b">
     <extinfo />
     </call>
  </methodcall>
</pkt>
```

Success Response

```
<pkt>
  <methodresponse name="refreshtoken" timestamp="2011/01/19 15:00:00.000">
    <result seq="f6fbaf29-19ca-4c9a-955b-709e11570f99" token="f3d608b5-531d-
43f7-ae6d-a7637594cca7">
      <extinfo />
    </result>
  </methodresponse>
</pkt>
```

Success Response DTD

```
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (result)>
<!ATTLIST methodresponse name (refreshtoken) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result token CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

Failure Response

```
<pkt>
  <methodresponse name="refreshtoken" timestamp="2011/01/19 15:00:00.000">
    <result seq="f6fbaf29-19ca-4c9a-955b-709e11570f99" errorcode="6002"
            errordescription="The player token expired.">
        <extinfo />
     </result>
  </methodresponse>
</pkt>
```

Failure Response DTD

```
<!ELEMENT pkt (methodresponse)>
<!ELEMENT methodresponse (refreshtoken)>
<!ATTLIST methodresponse name (play) #REQUIRED>
<!ATTLIST methodresponse timestamp CDATA #REQUIRED>
<!ELEMENT result (extinfo)>
<!ATTLIST result seq CDATA #REQUIRED>
<!ATTLIST result errorcode CDATA #REQUIRED>
<!ATTLIST result errordescription CDATA #REQUIRED>
<!ELEMENT extinfo ANY>
```

Error Codes

To enable the Microgaming system to display informative error messages to players, we require that the following error codes are returned under the prescribed conditions:

General Errors

Operator Code	Description	Applicable Methods
6000	Unspecified Error	All
6001	The player token is invalid.	All (Excl. Ping)
6002	The player token expired.	All (Excl. Ping)
6003	The authentication credentials for the API are incorrect.	All

Login Errors

Operator Code	Description	Applicable Methods
6101	Login validation failed. Login name or password is incorrect.	Login (with credentials)
6102	Account is locked.	Login
6103	Account does not exist.	Login
6104	Player is self-excluded.	Login
6105	Player must accept the T&Cs.	Login
6106	Must Show Player Protection.	Login
6107	The IP address is restricted.	Login
6108	The password expired.	Login
6109	Self-exclusion is over and the player must contact the operator to lift the exclusion. The cooling period is effective once this is done.	Login
6110	Self-exclusion is over but the player is in the cooling period.	Login

Operator Code	Description	Applicable Methods
6111	Account is blacklisted.	Login
6112	Player is deactivated. This applies to regulated markets only.	Login

Game Play Errors

Operator Code	Description	Applicable Methods
6501	Already processed with different details.	Play (Type = Refund)
6503	Player has insufficient funds.	Play (Bet / TransferToMgs)
6505	The player exceeded their daily protection limit.	Play (Bet / TransferToMgs)
6506	The player exceeded their weekly protection limit.	Play (Bet / TransferToMgs)
6507	The player exceeded their monthly protection limit.	Play (Bet / TransferToMgs)
6508	The player exceeded their game play duration.	Play (Bet / TransferToMgs)
6509	The player exceeded their loss limit.	Play (Bet / TransferToMgs)
6510	The player is not permitted to play this game.	Play (Bet / TransferToMgs)
6511	The external system name does not exist (**gamereference**).	Play (Bet / TransferToMgs)

Token Management

The Microgaming system stores only a single token per player, per **serverid**. This is always the last known token. For more information on the **serverid**, refer to the Glossary section. The temporary mapping of a player and their token is initiated from the login call. Thereafter, this token is overwritten by any successful request or response call triggered by either the player or on behalf of the player.

The Microgaming system requires you to keep two tokens alive for a player at any given time. It is recommended that **Token A**, used in the login request, expires once **Token B** is generated.

As the transfer of a token is server-to-server, **Token B** can be used for the duration of the session.

The requirements for token management are:

Description
A token must uniquely identify the player.
A token must be included in each response.
A new token must be sent in the login response. This is because only the login token is ever visible in a client. **Note:** All other method responses can include the current token.
The minimum recommended lifespan of the token is greater than six minutes.
The token expiry must be extended when the token is in use, to avoid expiry during gameplay.
The **refreshtoken** method must also extend the lifespan of the current token.
You must notify us if the token lifespan has changed. Our **refreshtoken** call is dependent on this setting.
Error packets returned to Microgaming must differentiate between tokens not generated by your system and expired tokens.

Note:
It is only the login token that is ever visible in a client, so we request that it is expired as soon as it is validated. After this, the transfer of tokens takes place only between the Microgaming gaming server and your API - ideally HTTPS.

Operator Hosted Web Applications

External or operator hosted applications are launched using the latest token for a player, where applicable. The token must be used to perform an authentication lookup for the player, but you must not expire or invalidate the token in the authentication process. This is because it may be used for the next Vanguard or GGi **GetBalance** or **Gameplay** call.

Some examples of applications hosted externally are Banking, the My Account pages, and player session and transaction history.

Note:
This functionality is only applicable to launching Banking from the Viper download client.

Gaming Platform Types

Casino Games and Tournaments

We offer casino games with our desktop Viper client (download), Flash client and Mobile Web HTML5 client. For information specific to Flash and Mobile Web HTML5 games, see the Flash and Mobile Web HTML5 Games section.

Authentication

All of our clients use the token-based authentication model. The token is either generated from your system, or the client uses the MGS **session-auth** token, if supported.

If the client is launched from an external system, such as your lobby page, you must use your own token for authentication.

Viper supports a web-based login dialog that you create and host. This application must be able to accept the player's login credentials, authenticate the player and return a token identifying the player. For more information on the integration required for the client, see the **Generic Viper Integration Spec** document.

Cross-platform Authentication

If the client is launched from the Microgaming Bingo client, the last known token is passed into the client. This is typically a token generated by your system. This triggers a second external authentication call for the player against your system. You must be able to handle multiple sessions for the player in this scenario. If you prefer to differentiate between authentication and game play tokens, then you must be able to handle the possibility of cross-platform authentication.

If the Microgaming client supports the MGS **session-auth** token, you can use this as an alternative to using a token generated by your system. Using this token means that no external call is made to your system for authentication. Internally, the flash session uses the current Microgaming session created by the parent client.

Game Play

Single Player and Multi-player Games

Any bet placed by the player triggers a play request with a **playtype** of **bet**.
Any payouts awarded to the player trigger a play request with a **playtype** of **win**.

Progressive Games

Progressive wins are identified by a **playtype** of **progressivewin** in a play request. The **actiondesc** attribute contains additional information about the progressive win.

Tournament-based Games

The behaviour of tournament games is very different to other games as money is transferred to the game and then transferred back to the player's balance once the tournament has ended.

The playtype for tournaments is typically **transfertomgs**, **transferfrommgs**, **tournamentpurchase** or **admin**.

The following table displays the mapping between the tournament action and the API action:

Tournament Action	API Action
Registration	play with **transfertomgs**
Addon	play with **tournamentpurchase**
Rebuy	play with **tournamentpurchase**
Payout	play with **transferfrommgs**
Prize	play with **transferfrommgs**
Cancellation with Refund	play with **admin**
Cancellation with Payout	play with **admin**
End of Tournament	endgame

End Game Notification

Single Player and Multi-player Games

For single-state games, the end game notification is sent immediately after the wager or the payout, depending on the outcome of the game.

For multi-state games, the end game notification is only sent when the player has completed the game. This could be hours or months after the last wager or payout was made.

There are also multi-state games that can be auto-completed, such as slot games with a gamble feature. So, when the player closes the client without clicking **Collect**, the end game notification is triggered minutes later. As the player could already be offline, the end game notification is sent with an offline token.

Tournament-based Games

Tournaments only trigger the end game notification when the last player is paid out. As the player could already be offline, the end game notification is sent with an offline token. In this game, the end game notification is sent for each player in the tournament, so the external system can receive the same end game multiple times.

Offline Tokens

Offline tokens are used if a player token does not exist in the Microgaming system.

Single and multi-player casino games make use of offline tokens for refunds, payouts and end game calls. The offline token differs depending on the context.

API Action	Offline Token
play with **win**	**loginname_gameid_actionid**
play with **refund**	**loginname_gameid_actionid**
endgame	**loginname_gameid**

Tournament-based games make use of offline tokens for refunds, payouts and end game calls. The offline token differs depending on the context.

API Action	Offline Token
play with **transferfrommgs**	**loginname_gameid**
play with **admin**	**loginname_gameid**
endgame	**serverid_gameid**

Error Behaviour

Bet

Deterministic Error

If an error, such as an insufficient balance, is included in a bet response, the Microgaming casino does not commit the transaction to the player's bet log and an error is displayed in the client.

In-deterministic Error

If the error is in-deterministic, such as a time out, the Microgaming casino marks the transaction as unresolved and adds it to the **retry queue**.

Win

Deterministic Error

If an error is included in a win response, the Microgaming casino adds a win message to the **commit queue** and an error is displayed in the client.

The transaction is marked as unresolved, but in this instance it is not resolved by the system.

The player is not able to continue playing the game until you have resolved the transaction using Orion integration. You are expected to resolve the transaction by either crediting the player's balance, or ignoring the unresolved transaction.

Although this is an unlikely scenario, it can be caused by a software bug in either of the two integrating systems.

In-deterministic Error

If the error is in-deterministic, such as a time out, the Microgaming casino marks the transaction as unresolved and adds it to the **retry queue**. An error is displayed in the client.

EndGame

Deterministic Error

If an error is included in the end game response, the Microgaming casino marks the notification as unresolved and adds it to the **retry queue**. An error is not displayed in the client.

In-deterministic Error

The same process is followed for in-deterministic errors as deterministic errors.

Bingo Games

Authentication

The desktop Flash client uses the token-based authentication model. The token is created by your system and must uniquely identify the player. The token is passed to the Bingo client when the client is launched from your website.

Cross Platform Authentication

The Bingo client supports access to casino games using the Microgaming Flash client, known as Casino in Bingo (CiB).

The Bingo client has access to the last known token and uses this to launch the Flash client. This triggers a second authentication request to your system. In this scenario, your system must be able to handle multiple sessions for the player in this scenario.

Game Play

Bingo makes use of the standard game play calls, so play with **playtype** of **bet**, **win** and **refund**, as well as the **getbalance** call.

Offline Tokens

If a player token does not exist in the Microgaming system, bingo games make use of offline tokens for refunds and payouts. The offline token consists of the player identifier, **gameid** and the **actionid** separated by an underscore: **loginname_gameid_actionid**.

Live Dealer Games

Authentication

The desktop Flash client uses the token-based authentication model. The token is created by your system and must uniquely identify the player. The token is passed to the client when the client is launched from your website.

Cross-platform Authentication

Casino games cannot currently be accessed from the Live Dealer client.

Game Play

The Live Dealer client makes use of the standard game play calls, including play with the **playtype** of **bet**, **win** and **refund**, as well as the **getbalance** call.

Live dealer games are also considered multi-state as you have multiple wagers and payouts per game round.

End Game Notification

The Live Dealer system only triggers the end game when the last player has been paid out.

Offline Tokens

Live Dealer games make use of offline tokens for refunds and payouts if a player token does not exist in the Microgaming system. The offline token consists of the player identifier, **gameid** and **actionid**, separated by an underscore: **loginname_gameid_actionid.**

Error Behaviour

Bet

Deterministic Error

If an error, such as an insufficient balance, is included in a bet response, the Microgaming casino does not commit the transaction to the player's bet log and an error is displayed in the client.

In-deterministic Error

If the error is in-deterministic, such as a time out, the Microgaming casino marks the transaction as unresolved and adds it to the **rollback queue** for processing. An error is displayed in the client. For more information on the **rollback queue**, see the Queue Processes section.

Win

Deterministic Error

If an error is included in a win response, the Microgaming casino commits the transaction to the player's bet log. A win message is added to the **commit queue** for processing. For more information on the **commit queue**, see the Queue Processes section.

The win message is retried until the threshold of five tries is reached.

In-deterministic Error

The same process is followed for in-deterministic errors as deterministic errors.

EndGame

Deterministic Error

If an error is included in the end game response, the Microgaming casino marks the notification as unresolved and adds it to the **endgame queue** for processing. An error is not displayed in the client. . For more information on the **endgame queue**, see the Queue Processes section.

In-deterministic Error

The same process is followed for in-deterministic errors as deterministic errors.

Flash and Mobile Web HTML5 Games

This section provides an overview of the changes to the Vanguard API for Flash and Mobile Web HTML5 games.

Orion

If you have a direct integration with Orion, three of the methods on the interface are updated to accommodate **RowID** as an int64 or long data type. This was done by adding a distinct **RowIdLong** parameter onto the following methods:

- **CompleteGameRequest** - on the request
- **GetCommitQueueData** - on the response
- **GetRollbackQueueData** - on the response

The **RowID** parameter is still populated where possible, but once the Int32 limit is exceeded the value defaults to **0** and only **RowIdLong** is valid. You must update your integration to accommodate this. For more information on how to update your integration, see the **009-019 - Orion Third Party Integration for Quickfire** document.

With the updates to Vanguard API, Orion also returns negative values for **RowIdLong** and **RowID** in the calls. Where possible, this only affects **RowID**.

This means that you must pass these negative **RowID**'s on the calls:

- **ManuallyValidateBet**
- **ManuallyCompleteGame**

Note:

- If you do not integrate directly with Orion and only use Vanguard Admin, your functionality remains the same.
- Applying the changes should not cause any issues as the **RowID** is defined as a signed integer. However, as this is new behaviour in the API, you must confirm that the changes are supported by your system.

Refund Behaviour

Vanguard API now manages indeterministic errors differently to before. Previously, a bet was rolled back for any error, regardless if it is indeterministic or deterministic. Now, successful responses and failure responses are handled differently.

In the previous Vanguard API, Vanguard sent a refund call for indeterministic errors, such as an API timeout on the bet. For deterministic errors, such as insufficient funds on a bet, Vanguard rolled back the bet and there was no need to send a refund. The Vanguard API assumed that the bet was never placed in your system.

With the updated Vanguard API, the system behaviour continues to resolve deterministic errors in the same way, but indeterministic errors are managed differently. With an indeterministic error, the bet is retried until the Vanguard API receives a deterministic response. In the case of a successful

response, Vanguard commits the bet and gameplay continues. In the case of a failure response, Vanguard rolls back the bet, no refund call is made, and gameplay continues.

GameID Re-use Behaviour

The Vanguard API previously re-used the **GameID** when a bet was rolled back, so the next bet players placed had the same **GameID** as the previously rolled back bet.

GameID's are no longer re-used, and the next bet players place after a rolled back bet has a new **GameID**. This simplifies **GameID** handling, as the complex logic required to manage re-use is no longer required. From a reporting perspective, this means that there could be gaps in the sequence of **GameID**'s for players when reports are viewed.

Note:
The **GameID** used in GGi integrations is called **TransactionID** in the generic integration.

End Game Behaviour

If you support the **EndGame** call but do not require it, we now support suppression to optimise performance.

This means that if the **EndGame** provides no function to your system and is only implemented to satisfy the interface requirement, we are now able to disable it. This reduces the number of calls we make to your API.

If you do not require the **EndGame** call, please contact your Service Manager to have it disabled.

Poker Games and Tournaments

For more information on poker games and tournaments, see the **008-591 Poker Vanguard Integration** document.

Queue Processes

This section provides descriptions of the queue processes included in various error behaviour scenarios.

Retry Queue Process

The retry queue process enables the Microgaming casino to retry either the play (win), play (bet) or the endgame request.

Players with a bet or win transaction in the retry queue are not able to continue playing the game until the system has resolved the transaction against the external account system. However, players can continue playing a different game.

An endgame request in the retry queue will not affect the player's gameplay.

Player-triggered Retry

When the player launches the game, the system immediately attempts to retry any transaction in the queue for the associated game. An error is displayed on the next wager until the transaction on the queue has been resolved.

System-triggered Retry

An asynchronous process running in the background also attempts to clear all transaction on the retry queue, regardless of whether the associated player is online or not.

The endgame requests on this queue are only retried when triggered by the system.

System-triggered retries are sent with an offline token.

Rollback Queue Process

The rollback bet process enables the Microgaming system to refund a player's bet. The rollback is only initiated when the Microgaming system does not receive a response for a play (bet) request. Players with a transaction in the rollback queue are locked and unable to continue playing.

In this event, the bet is not committed to the player's bet log, and an error is displayed in the client. The system adds a bet message to the rollback queue where it is picked up and processed.

If the play (refund) response is received, the transaction is flagged as processed and the player is unlocked.

If the play (refund) response is not received, the request is retried. If a response is not received after five refund attempts, the transaction is flagged as failed. You need to process the transaction through Orion.

If your system never received the original bet related to the refund, based on game ID and action ID, the refund response still returns a success. However, the player's account must not be credited.

No endgame request is sent after a refund request.

Commit Queue Process

The commit win process enables the Microgaming system to resent a player's win. This can happen when a response is not received for a play (win) request, or when your system responds with an error packet.

If the Microgaming system is unable to process the message in the commit queue through the retry mechanism, the transaction is flagged as failed. Your system needs to process the transaction through the Orion integration.

Please note this retry logic (retry threshold) is slightly different, depending on the gaming system.

For more information, see the win error behaviour of the relevant Gaming Platform Types section.

EndGame Queue Process

The endgame process enables the Microgaming system to resent the **EndGame** request for a player's game round. This can happen when a response is not received for the original **EndGame** request sent at the end of the game round.

If the Microgaming system is unable to process the message in the endgame queue after five attempts, the transaction is flagged as failed. Your system needs to process the transaction through the Orion integration.

A player is not locked if they have an **EndGame** request in the queue.

Glossary

Term	Description
Admin system	QuickFire offers two systems to enable you to unlock a player and process the refund or win – Orion or Vanguard Admin. Using Orion, you can build a GUI or at least automate the process. Vanguard Admin is a web application providing you with a GUI. Documentation on both of these applications is included in your welcome pack from Microgaming.
Game event	Either a bet or win.
Game round	At the very minimum, a game round consists of a bet. However, it can include multiple bets and wins depending on the type of game. The **gameid** is used to identify a player's game round.
Single state game	This is a basic slot game without any additional features, including gamble, double, bonus rounds or free spins.
Multi-state games	This is a slot game with features like gamble, double, free spins bonus rounds and table games. This category covers any game where there is additional player action required before the game is completed.
Single-state game	A single state game is a basic slot game without any additional features like gamble, double, bonus rounds or free spins.
Multi-state game	A multi-state game is a slot game with features like gamble, double, free spins bonus rounds and table games. This category covers any game where there can be additional player action required before the game can be completed.
Serverid	This is the unique identifier allocated to your casino and is considered a brand identifier. It is also referred to as the **Casinoid**. You receive multiple **serverid**'s for your different brands, but you can also be assigned a **serverid** per platform, such as Viper, Flash, bingo or poker.