NetEnt CasinoModule™ 10.10

Seamless Wallet REST API Integration Guide

Revision Table

Version	Description	Date
1.0	NetEnt CasinoModule™ 10.10 Seamless Wallet REST API Integration Guide	18 April 2017

Table of Contents

Introduction

NetEnt provides a REST API specification for operators to implement on their Wallet Server to enable NetEnt's remote gaming system to retrieve player balances, perform monetary transactions, etc., on their players' accounts.

REST, or Representational State Transfer, lets the Client and Server to exchange representations of a resource, where the representations reflect the resource's current state or its desired state.

Uniform Resource Identifiers (URIs) specify the resource to access. The URI contains path parameters and query parameters, which specify the variable parts of the resources. The HTTP method (verbs such as GET, POST, DELETE) specifies the action you want to perform on the resource.

All methods in this specification are idempotent; that is, multiple requests with the same 'transactionRef' must have the same effect on your system as if the requests were issued only once. For example, multiple deposit calls with the same 'transactionRef' must result in money being deposited only once. For successful transactions, the Client assumes that any subsequent requests yield the same response.

Payload format is JavaScript Object Notation (JSON).

Note:

Henceforth, this document uses the terms 'Client' for NetEnt's system and 'Server' for the operator's wallet server.

Integration Process

1. Create your environment, see Create Environment.
2. Optional: Create server token. See Server Tokens for information on how the Client supports server tokens.
3. Implement the REST API according to our specification, see API Reference.
4. Confirm your implementation by making the requests, by combining the HTTP method, the full URI to the resource, HTTP headers (if required) and the JSON-formatted payload. See examples in API Reference.

Create Environment

Set up the wallet server environment according to these requirements for a standard integration:

- **REST service:** You provide a REST service, implementing the representations described in API Reference.
- **Methods:** All requests are made using HTTP GET/POST/DELETE methods performed over HTTPS/SSL using standard HTTP protocol.
- **Authentication:** All the API methods must use HTTP Basic authentication. Authentication is handled with user name and password parameters in the HTTP request headers combined with HTTPS/SSL encryption and IP address restrictions on the Server. No user names and passwords are to be included in the JSON request payload.
- **Certificates:** An SSL certificate from a major Certification Authority (CA) must be installed on the Server. The Client does not support SSL client certificates.

- **Load balancing:** Your servers and load balancers must support HTTP Keep Alive, in order to reduce the number of connections made between the networks.
- **Error handling:** The Client expects API response codes and HTTP status codes as described in Codes.

Server Tokens

For security reasons (such as authentication of the player on all requests), the Client supports inclusion of token strings that are unique per player session. According to NetEnt's session management, which is one session per channel, this means that a player playing simultaneously on mobile and desktop will have two separate tokens active. Therefore it is important to always return separate tokens for different channels.

The token is received from the Server as part of the initial Player Account Balance response, and then passed back by the Client in the game context wallet requests. However, the Client supports receiving a new token with any response from the Server, which means that you can switch a player’s token any time.

Once a token is set for a player, the Client will send this token with each withdraw and deposit request made to the Server. The same token will be re-used, until a new one is set by the Server, or existing one is removed. If the Server sends an empty string as a token, the token is automatically removed, so that future requests do not contain any tokens.

Note:

This feature is supported only for calls which are made in a game round context, as opposed to for example tournament prize payouts.

Server Token Scenarios and Rules

1. The player starts a game.
2. Upon the first game request, the Client gets the player's balance from the Server.
3. Server returns balance and the new token in the response.
4. Client stores the token.
5. In the next requests, the token is retrieved for the particular player and session.

   The Client manages the token according to these rules:

   - If token is the same as before, then no action is done. The existing token is used.
   - If new token is received, then the token is updated.
   - If token is empty or NULL, then the token will be omitted in the future requests.

Configuration

The following cases for the withdraw and deposit requests are by default not enabled for integrations. Please contact NetEnt integration manager to enable them, if required.

- Withdraw requests with zero amount to withdraw (useful if you wish to get information about game rounds played with bonus money only).
- Deposit requests with zero amount to deposit (useful if you wish to get information about game rounds that only have bonus winning).
- Free round reporting (withdraw and deposit requests with reason FREE_ROUND_PLAY or FREE_ROUND_FINAL).

Note:

Enabling these requests leads to a big increase in the number of wallet requests, so enable them only if required.

API Reference

This reference describes all the resources with sample requests and responses.

Important:

This API will be extended with more parameters over time, so it is recommended that you enable the ‘ignore unknown json’ property if allowed by the framework being used. This will make the Server 'forwards compatible' with respect to clients and NetEnt will be able to extend the API without requiring an immediate server upgrade from you.

Player Account Currency

This resource represents the currency of a player's account.

Note:

The currency is fetched only once; the first time a player is logged in. So, any later change in player currency will not be honoured by the Client.

Request Data for Player Account Currency

Type	Request Data for Player Account Currency
Resource	Player Account Currency
HTTP method	GET
URI path	`https://<hostname>/walletserver/players/{player}/account/currency?session=<sessionId>`
URI path parameter	■ **player** (string, mandatory): The player identifier, whose account information is being fetched.
Query parameter	■ **session** (string, optional): The current player session identifier, if the request is being made in the context of a game round. Note that the Client may request the resource out of a game round context too, in which case there will not be any session identifier sent.

Response Data for Player Account Currency

Response Body for Player Account Currency
■ **responseCode** (int, mandatory): Status of the request. Zero (0) indicates success. For other response codes, see API Response Codes.   ■ **currencyISOCode** (string, mandatory): The code of player’s account currency, according to ISO 4217.   ■ **responseMessage** (string, optional): Any response message string (related to the **responseCode**), according to ISO 8859-1 standard (Latin-1). See API Response Codes for more information.

Example Request for Player Account Currency

Example:

```
GET /walletserver/players/player1/account/currency?session=1476270388070-45-
9QAWXB5EBP6BA HTTP/1.1
Host: wallet.operator.com
Charset: utf-8
Content-type: application/json
Authorization: Basic Auth
```

Example Response for Player Account Currency

Example:

```
{
 "responseCode":0,
 "currencyISOCode":"EUR",
 "responseMessage":"Success"
}
```

Player Account Balance

This resource represents the balance of a player's account.

Request Data for Player Account Balance

Type	Request Data for Player Account Balance
Resource	Player Account Balance
HTTP method	GET
URI path	`https://<hostname>/walletserver/players/{player}/account/balance?currency=<currencyISOCode>&game=<gameId>&session=<sessionId>`
URI path parameter	■ **player** (string, mandatory): The player identifier, whose account information is being fetched.
Query parameters	■ **currency** (string, mandatory): The currency in which the player balance is expected. It is the same currency as fetched from the Server when requesting the currency upon the first login of the player. ■ **game** (string, optional): The game identifier if the request is being made in the context of a game round. If the request is made outside of game round context, this parameter will be omitted. ■ **session** (string, optional): The current player session, if the request is being made in the context of a game round. If the request is made outside of game round context, this parameter will be omitted.

Response Data for Player Account Balance

Response Body for Player Account Balance
■ **responseCode** (int, mandatory): Status of the request. Zero (0) indicates success. For other response codes, see API Response Codes. ■ **serverToken** (string, optional): A token which will be included in all the future requests (until a new token is issued). ■ **balance** (double, mandatory): The player account balance. Double value range, with a maximum of 6 decimal places. ■ **responseMessage** (string, optional): Any response message string (related to the **responseCode**), according to ISO 8859-1 standard (Latin-1). See API Response Codes for more information.

Example Request for Player Account Balance

Example:

```
GET /walletserver/players/player1/account/balance?
currency=EUR&game=blacklagoon_sw&session=1476269821869-212-QQGS2WK7Y1750
HTTP/1.1
Host: wallet.operator.com
Charset: utf-8
Content-type: application/json
Authorization: Basic Auth
```

Example Response for Player Account Balance

Example:

```
{
 "responseCode":0,
 "serverToken":"token1101476867934846-247-EBI040JUU3E24-1476867945571",
 "balance":100.123456,
 "responseMessage":"Success"
}
```

Player Account Withdraw

This resource represents the withdrawal from of a player's account.

Note:
All parameters containing monetary amounts are rounded off to a maximum of 6 decimals.

Request Data for Player Account Withdraw

Type	Request Data for Player Account Withdraw
Resource	Player Account Withdraw
HTTP method	POST
URI path	`https://<hostname>/walletserver/players/{player}/account/withdraw`
URI path parameter	■ **player** (string, mandatory): The player identifier, whose account information is being modified.

Continued on next page

Type	Request Data for Player Account Withdraw
Request body	■ **withdrawRequest** (JSON object, mandatory) ■ **session** (string, mandatory): The current player session identifier, as the request is being made in the context of a game round. ■ **serverToken** (string, optional): A server token, if set previously as part of a response, or through some other mechanism. See Server Tokens for more information. ■ **currency** (string, mandatory): The currency in which the amount is being withdrawn. It is the same currency as fetched from the Server when requesting the currency, upon first login of the player. ■ **game** (string, mandatory): Game identifier, in whose context the request is being made. ■ **gameRoundRef** (long integer, mandatory): Game round reference, as the request is being made in the context of a game round. The request is never made outside of a game round context. ■ **transactionRef** (long integer, mandatory): A reference identifier, uniquely identifying this request. Multiple requests with the same transactionRef values must be considered duplicate, and to honour the idempotency contract, must return the same response as the original (first) request, without actually modifying the account balance (after the first one). ■ **amountToWithdraw** (double, mandatory): The amount to be withdrawn from the player account in the Server. ■ **bonusBet** (double, optional): The amount of money withdrawn from the bonus account in the Client. ■ **bonusBalance** (double, mandatory if **bonusBet** >0, Optional otherwise): The resulting bonus account balance in the Client. ■ **freeRoundBet** (JSON object, optional): Present only if a free round is played; that is, with reason FREE_ROUND_PLAY or FREE_ROUND_FINAL. If present, **amountToWithdraw** and **bonusBet** will be 0. This is the bet amount in a free round and also the free round bonus program details. Since it is a free round, no money is ever withdrawn from the player's account. This is only for reporting purposes. Note that messages sent to the game client in the response are not supported for free round calls. The **freeRoundBet** attributes are: ■ **amount** (double): The bet of a free round for which the call was issued. ■ **bonusProgramId** (long integer): Identifier of the bonus program by which the free rounds were issued. ■ **externalReferenceId** (string): Identifier for free rounds that were awarded via API. ■ **bonusProgramName** (string): Name of the free rounds bonus in case it was issued by a bonus program. ■ **bonusProgramDescription** (string): Description that was added when bonus program was created. ■ **groupId** (string): Identifier that was manually added when awarding free rounds via API or Back Office, for the purpose of identifying the bonus in a summary report.

-
    - **promotionCode** (string): Promotion code, if such was added when bonus program was set up.
    - **remainingRounds** (integer): Indicates the number of free rounds remaining for the player in this bonus program.
- **jackpotContributions** (JSON object, optional): List of jackpots (object **jackpotContributions**) and player's contributions made as part of this game round (if any). A jackpot game can have one or more jackpot contributions.
  The **jackpotContributions** attributes are:
    - **jackpotId** (string): Identifier of the jackpot that was contributed to. Examples from the 'Hall of Gods' game: 'hog_small', 'hog_medium', 'hog_large'.
    - **contribution** (double): Amount contributed by the player for this jackpot.
- **reason** (string, mandatory): String representing the reason for this particular withdraw. See Reason Codes for list of allowed reason strings, and their relevance.
- **transactionDate** (date in UTC format, according to ISO 8601 standard, optional): The time at which current transaction was initiated. This parameter will be made available in an upcoming release of the CasinoModule.

Response Data for Player Account Withdraw

Response Body for Player Account Withdraw

- **responseCode** (int, mandatory): Status of the request. Zero (0) indicates success. For other response codes, see API Response Codes.
- **serverTransactionRef** (string, mandatory if responseCode=0, optional otherwise): A value identifying the transaction on the Server, used for tracking/debugging purposes.
- **serverToken** (string, optional): A token sent by the Server, which will be included in all the future requests. See Server Tokens for more information.
- **balance** (double, mandatory): The player account balance after the transaction. Double value range, with maximum of 6 decimal places.
- **serverBonusWithdraw** (double, optional): Indicates how much of the withdrawn amount that is bonus money. This parameter is reserved for future use.
- **responseMessage** (string, optional): Any response message string (related to the **responseCode**), according to ISO 8859-1 standard (Latin-1), which is stored for audit purposes. However, messages for codes 990 or 991 will as well be displayed to the player. See API Response Codes for more information.
- **messagesToPlayer** (JSON object, optional): A list of messages (object **messages**) to be displayed to the player after the game round outcome has been displayed, according to ISO 8859-1 standard (Latin-1), see Message Codes.
  The **messages** attributes are:
    - **code** (string)
    - **message** (string)

Example Request for Player Account Withdraw (normal game play)

Example:

```
POST /walletserver/players/player1/account/withdraw HTTP/1.1
Host: wallet.operator.com
Charset: utf-8
Content-type: application/json
Authorization: Basic  Auth

{
"session":"1476867934846-247-EBI040JUU3E24"
"serverToken":"token1101476867934846-247-EBI040JUU3E24-1476867945571",
"currency":"EUR",
"game":"hallofgods_sw",
"gameRoundRef":33,
"transactionRef":4,
"amountToWithdraw":10.0,
"bonusBet":1.0,
"bonusBalance":23.5,
"jackpotContributions":{
             "jackpotContributions":[
                      {"jackpotId":"jp1","contribution":0.01375},
                      {"jackpotId":"jp2","contribution":0.005},
                      …
                      ]
             },
"reason":"GAME_PLAY",
"transactionDate":"2016-11-02T09:41:46.000+0000"
}
```

Example Request for Player Account Withdraw (with free rounds, reason FREE_ROUND_PLAY)

Example:

```
POST /walletserver/players/token74/account/withdraw HTTP/1.1
Host: wallet.operator.com
Charset: utf-8
Content-type: application/json
Authorization: Basic Auth

{
"session":"1477931198945-26419-A75A1V7UXY04G"
"serverToken":"player_1111477931198945-26419-A75A1V7UXY04G-1477931218214",
"currency":"EUR",
"game":"flowers_sw",
"gameRoundRef":12458,
"transactionRef":14567,
"amountToWithdraw":0.0,
"freeRoundBet":{
             "amount":75.0,
             "bonusProgramId":1781,
             "externalReferenceId":"ExtRefId153",
             "bonusProgramName":"*** System-created manual Free Rounds ***",
             "remainingRounds":0
             }
"reason":"FREE_ROUND_PLAY",
"transactionDate":"2016-11-02T09:41:46.000+0000"
}
```

Example Request for Player Account Withdraw (with free rounds, reason FREE_ROUND_FINAL)

Example:

```
POST /walletserver/players/token74/account/withdraw HTTP/1.1
Host: wallet.operator.com
Charset: utf-8
Content-type: application/json
Authorization: Basic Auth

{
"session":"1477931198945-26419-A75A1V7UXY04G"
"serverToken":"player_1111477931198945-26419-A75A1V7UXY04G-1477931218214",
"currency":"EUR",
"game":"flowers_sw",
"gameRoundRef":12461,
"transactionRef":14575,
"amountToWithdraw":0.0,
"freeRoundBet":{
             "amount":0.3,
             "bonusProgramId":1784,
             "externalReferenceId":"ExtRefId156",
             "bonusProgramName":"*** Created by Admin Tool. ***",
             "groupId":"",
             "remainingRounds":0
             }
"reason":"FREE_ROUND_FINAL",
"transactionDate":"2016-11-02T09:41:46.000+0000"
}
```

Example Response for Player Account Withdraw

Example:

```
{
"responseCode":0,
"serverTransactionRef":4589,
"serverToken":"token1101476867934846-247-EBI040JUU3E24-1476867945571",
"balance":12.123456,
"serverBonusWithdraw": 5.0,
"responseMessage":"Success",
"messagesToPlayer":{
             "messages": [ {"code":993,"message":"Happy Birthday"},
                         {"code":993,"message":"You have 100 bonus"},
                         …
                         ]
             }
}
```

Player Account Deposit

This resource represents the deposit to a player's account.

Note:
All parameters containing monetary amounts are rounded off to a maximum of 6 decimals.

Request Data for Player Account Deposit

Type	Request Data for Player Account Deposit
Resource	Player Account Deposit
HTTP method	POST
URI path	`https://<hostname>/walletserver/players/{player}/account/deposit`
URI path parameter	■ **player** (string, mandatory): The player identifier, whose account information is being modified.

Continued on next page

Type	Request Data for Player Account Deposit
Request body	■ **depositRequest** (JSON object, mandatory) ■ **session** (string, optional): The current player session identifier, if the request is being made in the context of a game round. ■ **serverToken** (string, optional): A server token, if set previously as part of a Server response, or through some other mechanism. See Server Tokens for more information. ■ **currency** (string, mandatory): The currency in which the amount is being deposited. It is the same currency as fetched from the Server when requesting the currency, upon first login of the player. ■ **game** (string, optional): Game identifier, in whose context the request is being made. Deposit requests may be made outside of game context (such as tournament win with zero wagering). For all other cases, such as WAGERED_BONUS deposit requests for free rounds and other bonuses, this parameter is mandatory. ■ **gameRoundRef** (long integer, optional): Game round reference, if the request is being made in the context of a game round. The request may be made outside of a game round context (such as tournament win with zero wagering with reason AWARD_TOURNAMENT_WIN). For all other cases, such as reason WAGERED_BONUS for free rounds and other bonuses, this parameter is mandatory. ■ **transactionRef** (long integer, mandatory): A reference identifier, uniquely identifying this request. Multiple requests with same transactionRef values must be considered duplicate, and to honour the idempotency contract, must return the same response as the original (first) request, without actually modifying the account balance (after the first one). ■ **amountToDeposit** (double, mandatory): The amount to deposit into the player account in the Server. ■ **bonusWin** (double, optional): The amount of money deposited into the bonus account in the Client. ■ **bonusBalance** (double, mandatory if **bonusWin**>0, optional otherwise): The resulting bonus account balance in the Client. ■ **source** (string, optional): This is sent only if the **reason** parameter is CLEAR_HANGED_GAME_STATE. Possible trigger types: ■ finishGameRoundJob: Game round is finished by a scheduled job. ■ api: Game round is finished through API ■ manual: Game round is finished manually through Back Office ■ cancelGameRound: Game round is cancelled through NetEnt Live ■ **freeRoundWin** (JSON object, optional). Present only if a free round is played; that is, with reason FREE_ROUND_PLAY or FREE_ROUND_FINAL. This is the win amount in a free round and also the free round bonus program details. Since it is a free round, no money is deposited into the player's account at this point. This is only for reporting purposes. Note that messages sent to the game client in the response are not supported for free round calls. The **freeRoundWin** attributes are: ■ **amount** (double): The win of a free round for which the call was issued. ■ **accumulatedWinnings** (integer): Indicates the total winnings

	that the player has received so far, for this particular bonus. ▪ **bonusProgramId** (long integer): Identifier of the bonus program by which the free rounds were issued. ▪ **externalReferenceId** (string): Identifier for free rounds that were awarded via API. ▪ **bonusProgramName** (string): Name of the free rounds bonus in case it was issued by a bonus program. ▪ **bonusProgramDescription** (string): Description that was added when bonus program was created. ▪ **groupId** (string): Identifier that was manually added when awarding free rounds via API or Back Office, for the purpose of identifying the bonus in a summary report. ▪ **promotionCode** (string): Promotion code, if such was added when bonus program was set up. ▪ **remainingRounds** (integer): Indicates the number of free rounds remaining for the player for this bonus. ▪ **bonusPrograms** (JSON object, optional): List of bonuses (object **bonus**) of which wagering requirement has been fulfilled as part of this game round. The **bonus** attributes are: ▪ **bonusProgramId** (long integer): Identifier of the bonus program by which the bonus was issued. ▪ **depositionId** (long integer): When bonus is granted to a player a bonus deposition is created in the Client for the purpose of tracking the status of the bonus whether it is granted, wagered, expired, etc. ▪ **bonusProgramDescription** (string): Description that was added when bonus program was created. ▪ **externalReferenceId** (string): Identifier for free rounds that were awarded via API. ▪ **groupId** (string): Identifier that was manually added when awarding free rounds via API or Back Office, for the purpose of identifying the bonus in a summary report. ▪ **promotionCode** (string): Promotion code, if such was added when bonus program was set up. ▪ **wageredAmount** (double): The amount left of the bonus winnings when wagering requirements have been fulfilled. If no wagering is required, this amount is the full bonus winnings which are converted to real money immediately. ▪ **tournaments** (JSON object, optional): List of tournaments (object **tournament**) whose prizes have been wagered. For tournaments with wagering requirement>0, this request is made in context of a game round (which wagered the prize). If wagering requirement=0, the request is made outside the context of a game round, and also the reason type is different (AWARD_TOURNAMENT_WIN as against WAGERED_BONUS – see Reason Codes). The **tournament** attributes are: ▪ **tournamentId** (long integer): Identifier automatically generated when tournament was created.

- **tournamentOccurrenceId** (long integer): Identifier of a single instance of a recurring tournament.
- **wageredAmount** (double): The amount left of the tournament winnings when wagering requirements have been fulfilled. If no wagering is required, this amount is the full tournament winnings which are converted to real money immediately.

- **jackpotWinnings** (JSON object, optional): List of jackpots (object **jackpots**) and player's winnings made as part of this game round (if any).
  The **jackpots** attributes are:
  - **jackpotId** (string): Identifier of the jackpot that was won. Examples from the 'Hall of Gods' game: 'hog_small', 'hog_medium', 'hog_large'.
  - **win** (double): Amount won by the player.
- **reason** (string, mandatory): String representing the reason for this particular deposit. See Reason Codes for list of allowed reason strings, and their relevance. If reason is CLEAR_HANGED_GAME_STATE, then the **source** parameter will be sent as well.
- **startDate** (date in UTC Offset format, according to ISO 8601 standard, optional): The start time of game round, if deposit is in context of a game round.
- **transactionDate** (date in UTC format, according to ISO 8601 standard, optional): The time at which current transaction was initiated. This parameter will be made available in an upcoming release of the CasinoModule.

Response for Player Account Deposit

Response Body for Player Account Deposit

- **responseCode** (int, mandatory): Status of the request. Zero (0) indicates success. For other response codes, see API Response Codes.
- **serverTransactionRef** (string, mandatory if responseCode=0, optional otherwise): A value identifying the transaction on the Server, used for tracking/debugging purposes.
- **serverToken** (string, optional): A token sent by the Server, which is included in all the future requests. See Server Tokens for more information.
- **balance** (double, mandatory): The resulting player account balance. Java double value range, with maximum of 6 decimal places. The Server returns a balance even if an error response code is being returned, to minimize the get balance requests.
- **serverBonusDeposit** (double, optional): Indicates how much of the deposited amount that is bonus money. This parameter is reserved for future use.
- **responseMessage** (string, optional): Any response message string (related to the **responseCode**), according to ISO 8859-1 standard (Latin-1), which is stored for audit purposes. However, messages for codes 990 or 991 will as well be displayed to the player. See API Response Codes for more information.
- **messagesToPlayer** (JSON object, optional): A list of messages (object **messages**) to be displayed to the player after the game round outcome has been displayed, according to ISO 8859-1 standard (Latin-1), see Message Codes.
  The **messages** attributes are:
  - **code** (string)
  - **message** (string)

Example Request for Player Account Deposit (normal game play)

Example:

```
POST /walletserver/players/token74/account/withdraw HTTP/1.1
Host: wallet.operator.com
Charset: utf-8
Content-type: application/json
Authorization: Basic Auth

{
"session":"1478079643792-96-5F7FYI06M0KOR"
"serverToken":"token909014780796437 92-96-5F7FYI06M0KOR-1478079681358",
"currency":"EUR",
"game":"beach_sw",
"gameRoundRef":1773,
"transactionRef":4686,
"amountToDeposit":37.0,
"bonusWin":55.5,
"bonusBalance":55.5,
"reason":"GAME_PLAY_FINAL",
"startDate":"2016-11-02T09:41:46.000+0000",
"transactionDate":"2016-11-02T09:41:46.000+0000"
}
```

Example Request for Player Account Deposit (with free rounds, bonus programs and tournaments)

Example:

```
POST /walletserver/players/player1/account/deposit HTTP/1.1
Host: wallet.operator.com
Charset: utf-8
Content-type: application/json
Authorization: Basic Auth

{
"session":"1476867934846-247-EBI040JUU3E24"
"serverToken":"token1101476867934846-247-EBI040JUU3E24-1476867945571",
"currency":"EUR",
"game":"blacklagoon_sw",
"gameRoundRef":12712,
"transactionRef":18699,
"amountToDeposit":20.0,
"bonusWin":1.0,
"bonusBalance":23.5,
"freeRoundWin":{
      "amount":1.0,
      "accumulatedWinnings”:3.0,
      "bonusProgramId":1,
      "externalReferenceId":"free round 1",
      "bonusProgramName":"FR1",
      "bonusProgramDescription":"Free round open to all",
      "groupId":"FRGroup 1",
      "promotionCode":"promo1",
      "remainingRounds":2
      },
```

Continued on next page

Example:

```
"bonusPrograms":{
      "bonus":[ {"bonusProgramId":1,
               "depositionId":1,
               "bonusProgramName":"bp1",
               "bonusProgramDescription":"Good Bonus",
               "groupId":"Bonus Group 1",
               "promotionCode":"promo1",
               "externalReferenceId":"extRef1",
               "wageredAmount":1.0
               },
               {"bonusProgramId":2,
               "depositionId":2,
               "bonusProgramName":"bp2",
               "bonusProgramDescription":"Good Bonus2",
               "groupId":"Bonus Group 2",
               "promotionCode":"promo2",
               "externalReferenceId":"extRef2",
               "wageredAmount":2.0}
               ]
      },
"tournaments":{
      "tournament":[ {"tour-
      namentId":29,"tournamentOccurrenceId":29,"wageredAmount":20.0},
      {"tournamentId":2,"tournamentOccurrenceId":2,"wageredAmount":1.0}
                ]
      },
"jackpotWinnings":{
      "jackpots":[ {"jackpotId":"jp1","win":1.0},
               {"jackpotId":"jp2","win":2.0},
               …
               ]
      },
"reason":"GAME_PLAY",
"startDate":"2016-11-02T09:41:41.000+0000",
"transactionDate":"2016-11-02T09:41:41.000+0000"
}
```

Example Response for Player Account Deposit

Example:

```
{
"responseCode":0,
"responseMessage":"Success",
"serverTransactionRef":4686,
"serverToken":"token909014780796437 92-96-5F7FYI06M0KOR-1478079681358",
"balance":37.0,
"serverBonusDeposit": 55.5,
"messagesToPlayer":{
      "messages": [ {"code":993,"message":"Happy Birthday"},
                  {"code":993,"message":"You have 100 bonus"},
                  …
                  ]
      }
}
```

Player Account Withdraw Rollback

This resource represents the rollback of a withdraw transaction made earlier from a player's account. If the Server does not find a transaction that is being requested to be rolled back, it should just indicate it as a success (a non existing transaction on Server is equivalent to a rolled back one from system perspective).

Note:

- The Client accepts separate external transaction reference ID in response to rollbacks (if the Server treats rollbacks as new transactions).
- Server tokens are supported in request/responses, but not in rollback requests as they occur outside of game round context.
- If game round identifier exists, it is sent as additional info to rollback transactions.

Request Data for Player Account Withdraw Rollback

Type	Request Data for Player Account Withdraw Rollback
Resource	Player Account Withdraw
HTTP method	DELETE
URI path	`https://<hostname>/walletserver/players/{player}/account/withdraw?game=<gameId>&gameRoundRef=<gameroundId>&transactionRef=<transaction reference>&session=<sessionId>`
URI path parameter	■ **player** (string, mandatory): The player name, whose account information is being modified.
Query parameters	■ **game** (string, optional): Game identifier of the game for which the original transaction was made. ■ **gameRoundRef** (long integer, optional): Game round identifier, for which the original transaction (which is being rolled back) was made. This is sent if available (which is not always the case, since game round might have been completely rolled back leaving no trace of game round). ■ **transactionRef** (long integer, mandatory): The transaction reference indicating the original withdraw request which is being rolled back. This is the same value which was sent with the similar named request parameter in withdraw request, to enable Server to identify which transaction to roll back (delete). ■ **session** (string, mandatory): The session identifier in which the original withdraw request (which is being rolled back) was made.

Response Data for Player Account Withdraw Rollback

Response Body for Player Account Withdraw Rollback

- **responseCode** (int, mandatory): Status of the request. Zero (0) indicates success. For other response codes, see API Response Codes.
- **serverTransactionRef** (string, optional): A value identifying the transaction (rollback) on the Server, used for tracking and debugging purposes. Implementations must return the same value as in the original transaction (which is being rolled back) or a new value. If the transaction was non-existent on the Server, a null value may be returned.
- **balance** (double, optional): The player account balance. Double value range, with maximum of 6 decimal places. The Server must return a balance if the transaction was actually rolled back on the Server (resulting in a change in player balance).
- **responseMessage** (string, optional): Any response message string (related to the **responseCode**), according to ISO 8859-1 standard (Latin-1). See API Response Codes for more information.

Example Request for Player Account Withdraw Rollback

Example:

```
DELETE https://wallet.operator.com/
walletserver/players/player1/account/withdraw?
game=starburst_sw&gameRoundRef=123456789&transactionRef=1122331&
session=1476270388070-45-9QAWXB5EBP6BA
```

Example Response for Player Account Withdraw Rollback

Example:

```
{
"responseCode":0,
"serverTransactionRef":4589,
"balance":12.123456,
"responseMessage":"Success"
}
```

Codes

Reason Codes

These are the reasons that the Client sends in the withdraw and deposit requests.

Reason code	Description
GAME_PLAY	Normal game play.
GAME_PLAY_ FINAL	Game round is finished. Every game round will send only one transaction with this type.
AWARD_ TOURNAMENT_ WIN	Tournament win with zero wagering requirements, so the winnings are deposited immediately into winner's account once the tournament is finished.
CLEAR_ HANGED_ GAME_STATE	Non-completed game rounds that have been cleared through Back Office, API or scheduled job.
FREE_ROUND_ PLAY	A free round is played. Note that by default, the Client does not send calls with this reason type. Contact NetEnt integration manager if you need to enable this.
FREE_ROUND_ FINAL	Last free round is played. Every free round bonus will send only one transaction with this type. Note that by default, the Client does not send calls with this reason type. Contact NetEnt integration manager if you need to enable this.
WAGERED_ BONUS	Wagering requirement of the bonus has been fulfilled and bonus money is converted to real money and is paid to the player's account. If the wagering requirement is zero, then this call will pay out the bonus money immediately.

API Response Codes

Following are the expected codes in response to the requests (sent in the **responseCode** parameter).

Code	Description
0	Success.
1	Not enough money in player account.
2	Invalid/illegal currency.
3	Negative deposit.
4	Negative withdraw.
6	Player limit exceeded.
7	Invalid server token exception.
99	Retry exception.

100	Unknown error. The message in the **responseMessage** parameter indicates the reason for the error.
990	No game outcome, no further game play is allowed, the game client is inactivated. Example use case: Fraud, session terminated, gaming limit reached, Server down. If this code is sent, then the message in the **responseMessage** parameter will be displayed to the player. Message format is according to ISO 8859-1 standard (Latin-1). Example message displayed to player: "Player limit is exceeded".
991	No game outcome, further game play is allowed, game client is still active. Example use case: The player is out of funds. If this code is sent, then the message in the **responseMessage** parameter will be displayed to the player.Message format is according to ISO 8859-1 standard (Latin-1). Example message displayed to player: “Please deposit to be able to play”.

HTTP Status Codes

The Server is expected to send at least the following HTTP status codes as per description in the table.

The Client expects to receive HTTP 200 for success, and for business errors such as not enough money, illegal currency, etc., the Server must use code 403 for the Client to treat them as forbidden. If other HTTP status codes are received, the Client treats them as normal errors and will retry the request (unless it is a withdraw request that fails, in which case the Client makes a rollback).

HTTP response code	Description
200	OK - The Server was able to process the request, and return a response successfully.
400	Bad Request - The Server cannot or will not process the request due to an apparent Client error (e.g., malformed request syntax, invalid request message framing, missing mandatory parameter, etc.) The response should indicate the exact error.
401	Unauthorized - Authentication failed for the Client request.
403	Forbidden - Business logic or business errors, such as not enough money, will be sent in response of JSON data.
500	Internal Server error - A generic error message, given when an unexpected condition was encountered and no more specific message is suitable.

Message Codes

Message codes can be used in the **messagesToPlayer** parameter by the Server when you wish to display messages directly to the player, according to ISO8859-1 standard (Latin-1), after the game round outcome has been displayed.

Message code	Description
992	Game outcome, no further game play/game client is inactivated. Example use case: Graceful kick-out (geolocation issues), gaming limit reached. Example message displayed to player: "Your player time limit has been exceeded"
993	Game outcome, further game play allowed/game client is still active. Example use case: Just-for-information, tax message in New Jersey, ads, near session end warning, etc. Example messages displayed to player: "Happy Birthday", "You have received 200 free rounds"

Appendix: Migrate from SOAP API to REST API

This appendix is a comparison of the methods, parameters and object attributes that are valid for the Seamless Wallet SOAP API and REST API respectively, to be used when migrating the integration API from SOAP to REST.

The first table lists the methods and their parameters, and the second table lists attributes to the relevant objects.

Methods and Parameters

SOAP API method	REST API method and resource	SOAP parameters	=	REST parameters
getPlayerCurrency	GET on Player Account Currency resource	Request callerId callerPassword playerName sessionId Response currencyISOCode **No SOAP equivalent** **No SOAP equivalent**		Request **[in header]** **[in header]** player session Response currencyISOCode responseCode responseMessage
getBalance	GET on Player Account Balance resource	Request callerId callerPassword playerName sessionId currency gameId Response balance **No SOAP equivalent** **No SOAP equivalent** **No SOAP equivalent**		Request **[in header]** **[in header]** player session currency game Response balance serverToken responseCode responseMessage
withdraw	POST on Player Account Withdraw resource	Request callerId callerPassword playerName sessionId **No SOAP equivalent** currency gameId gameRoundRef		Request **[in header]** **[in header]** player session serverToken currency game gameRoundRef

'withdraw' continued on next page

SOAP API method	REST API method and resource	SOAP parameters	=	REST parameters
withdraw	POST on Player Account Withdraw resource	Request		Request
		transactionRef		transactionRef
		amount		amountToWithdraw
		bonusBet		bonusBet
		No SOAP equivalent		bonusBalance
		jackpotContributions		jackpotContributions
		freeRoundBet		freeRoundBet
		reason		reason
		No SOAP equivalent		transactionDate
		Response		Response
		No SOAP equivalent		responseCode
		transactionId		serverTransactionRef
		No SOAP equivalent		serverToken
		balance		balance
		No SOAP equivalent		serverBonusWithdraw
		No SOAP equivalent		responseMessage
		message		messagesToPlayer
deposit	POST on Player Account Deposit resource	Request		Request
		callerId		**[in header]**
		callerPassword		**[in header]**
		playerName		player
		sessionId		session
		No SOAP equivalent		serverToken
		currency		currency
		gameId		game
		gameRoundRef		gameRoundRef
		transactionRef		transactionRef
		amount		amountToDeposit
		bigWin		**No REST equivalent**
		bonusWin		bonusWin
		No SOAP equivalent		bonusBalance
		jackpotAmount		jackpotWinnings
		bonusPrograms		bonusPrograms
		tournaments		tournaments
		freeRoundWin		freeRoundWin
		reason		reason
		source		source
		startDate		startDate
		No SOAP equivalent		transactionDate
		Response		Response
		No SOAP equivalent		responseCode
		transactionId		serverTransactionRef
		No SOAP equivalent		serverToken
		balance		balance
		No SOAP equivalent		serverBonusDeposit

		No SOAP equivalent message		responseMessage messagesToPlayer

SOAP API method	REST API method and resource	SOAP parameters	=	REST parameters
rollbackTransaction	DELETE on Player Account Withdraw resource	Request callerId callerPassword playerName sessionId **No SOAP equivalent** transactionRef gameId Response **No parameters** (except on errors)		Request **[in header]** **[in header]** player session gameRoundRef transactionRef game Response responseCode serverTransactionRef balance responseMessage
withdrawAndDeposit	**Not used for REST A PI**			

Object Attributes

SOAP API object	REST API object	SOAP attributes	=	REST attributes
freeRoundBet	freeRoundBet	bonusProgramId amount externalReferenceId bonusProgramName bonusPro-gramDescription groupId promotionCode remainingRounds		bonusProgramId amount externalReferenceId bonusProgramName bonusPro-gramDescription groupId promotionCode remainingRounds
freeRoundWin	freeRoundWin	bonusProgramId amount accumulatedWinnings externalReferenceId bonusProgramName bonusPro-gramDescription groupId promotionCode remainingRounds		bonusProgramId amount accumulatedWinnings externalReferenceId bonusProgramName bonusPro-gramDescription groupId promotionCode remainingRounds
bonusPrograms (list of 'bonus')	bonusPrograms (list of 'bonus')	bonusProgramId depositionId externalReferenceId		bonusProgramId depositionId externalReferenceId

SOAP API object	REST API object	SOAP attributes	=	REST attributes
		No SOAP equivalent **No SOAP equivalent** **No SOAP equivalent** **No SOAP equivalent**		bonusPro-gramDescription groupId promotionCode wageredAmount
tournaments (list of 'tour-nament')	tournaments (list of 'tournament')	tournamentId tournamentOccurrenceId **No SOAP equivalent**		tournamentId tournamentOccurrenceId wageredAmount
jack-potContributions (list of 'jackpot')	jackpotContributions (list of 'jack-potContributions')	jackpotId contribution		jackpotId contribution
jackpotAmount	jackpotWinnings (list of 'jackpots')	**No SOAP equivalent** jackpotAmount		jackpotId win
message (list of 'messages')	messagesToPlayer (list of 'messages')	code message		code message