NETENT

NetEnt CasinoModule™ 10.10

Seamless Wallet SOAP API Integration Guide

Revision Table

Version	Description	Date
1.0	NetEnt CasinoModule™ 10.10 Seamless Wallet SOAP API Integration Guide	7 April 2017

Table of Contents

Seamless Wallet SOAP API Integration Guide

This guide provides information about how to integrate Seamless wallet functionality with an operator's system. It contains a description of the integration process and the tasks involved.

The main parts in this section are:

- Seamless Wallet Integration Process
  Provides an overview of the integration process and gives detailed information about main activities in the integration process.
- Seamless Wallet Integration Reference
  Provides information about required external wallet SOAP API methods, gives instructions for error handling and describes transaction handling scenarios.
- Seamless Wallet SOAP API Examples
  Descriptions and examples of how to use the Seamless wallet SOAP API.

Note:

Seamless wallet is one of NetEnt's optional offerings. If you would like to understand more how this function can add value to your day-to-day operation, please contact your Account Manager at NetEnt for further discussions.

Seamless Wallet Infrastructure

The wallet client adapter is the web service client in the CasinoModule server that handles the integration with external wallets.

When CasinoModule is configured forSeamless wallet option, web service calls are made to the operator’s wallet server to retrieve player balances, perform monetary transactions, and perform transaction rollbacks.

Please note the following about infrastructure in a standard integration:

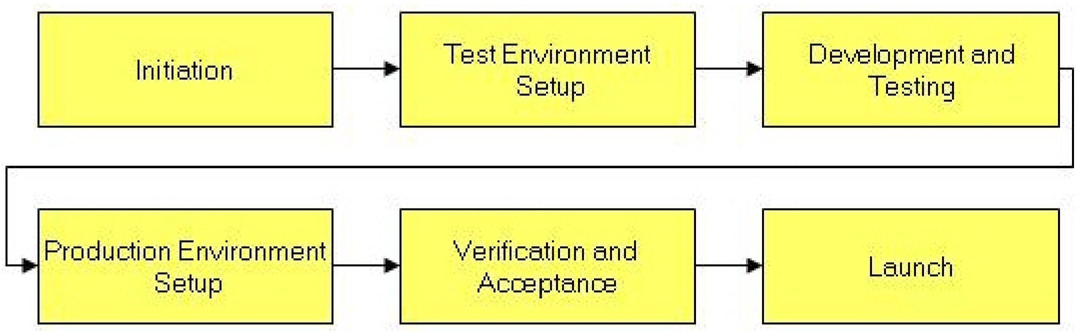
- The operator provides a web service, implementing the methods described in the WSDL file. The WSDL file is provided by NetEnt.
- All requests are HTTP POST requests performed over HTTPS/SSL using the SOAP protocol. The standard port (443) must be used.
- Authentication is handled with user name and password parameters in the SOAP requests, combined with HTTPS/SSL encryption and IP address restrictions on the operator's side.
- The wallet client adapter does not support SSL client certificates or HTTP authentication.
- An SSL certificate from a major CA should be installed on the operator's side.
- The application servers and load balancers on the operator's side must support HTTP Keep Alive, in order to reduce the number of connections made between the networks.
- If there are bandwidth limitations on the operator's side, NetEnt recommends the use of QoS mechanisms. The bandwidth usage is estimated to no more than 3 kbps per active player.

Seamless Wallet Integration Process

The goal of Seamless wallet integration is to launch Seamless wallet functionality in a CasinoModule installation. To do this, the operator must develop a web service API based on specifications from NetEnt and allocate test resources prior to launch, as well as during the first week after launch. Before Seamless wallet is launched, the casino must pass quality assurance tests performed by NetEnt.

Depending on which Seamless wallet Mode the operator chooses, additional changes may be required to the operator's website.

The main activities in the Seamless wallet integration process are shown below.



Initiation

Below are the activities involved in the initiation process.

Workflow

#	Description
1.	The operator reviews the documentation available on the **Client Area** of the NetEnt website.
2.	The operator plans change to the website.

Test Environment Setup

The operator sets up a staging environment for the Seamless wallet integration.

- Any existing NetEnt test environment may be reused.
- The operator's staging environment must be accessible from NetEnt offices, IP 195.81.148.x.

Development and Testing

The development of the wallet implementation is the responsibility of the operator. The time required for development is dependent on the number of development resources the operator provides.

The API and integration documentation can be found on the **Client Area** of the NetEnt website.

NetEnt provides support on integration related questions.

NetEnt also provides the WSDL file on which the wallet server implementation is based on.

Workflow

The operator's development can be divided in the following activities.

#	Description
1.	Start development of the Wallet API.
2.	Work on changes to the web site.
3.	Continue and finalize development of the Wallet API.
4.	Finalise changes to the website.

Things to Consider

There are multiple settings (casino properties) that control how the CasinoModule Seamless wallet adapter behaves. Depending on the values of these settings, performance will be affected.

Setting	Description	Default value	Note
Wallet time-out	Controls how long CasinoModule waits for a response from the operator's wallet API before timing out	20 seconds	It is important that the time-out values on the NetEnt side are mapped logically toward the values on the operator's side.
Rollback job (quartz job setting)	Controls how often the rollback job is run. The rollback job looks for uncommitted transactions and initiates a rollback towards the operator API.	10 minutes	There is a trade off between transaction integrity and performance.
Rollback: Wait time	When the rollback job is run it looks at this value and searches only for transactions older than this value.	60 seconds	
Rollback: Hours to go back	When the rollback job is run it looks at this value and searches only for transactions newer than this value.	240 hours (10 days)	
Bigwin	The amount in casino currency that is considered a big win.	10	Must be set when `withdrawAndDeposit` is used.

Production Environment Setup

The test environment is set up so the operator can develop their integration toward it. The production system setup is ordered once the integration is approved.

If moving to Full Mode, consider the following:

- Any ongoing bonus program must end prior to switching to Full Seamless mode.
- Any ongoing tournament must end prior to switching to Full Seamless mode.
- Any Bonus money that is left on the player's account when migrating to Seamless wallet will expire automatically after a certain time (normally within 30 days). A operator may choose to have the bonus money converted to real money instead.

  Note:

  This decision must be made (along with associated database change) prior to the Seamless wallet migration

- Any awarded free rounds that have not been used up prior to the migration will expire automatically after a certain time.

Verification and Acceptance

NetEnt assists the operator with their test plan by providing specified test areas to ensure the product functionality.

Workflow

#	Description
1.	NetEnt provides the operator with a test area guide to be used when performing tests in the operator's test environment.
2.	The operator tests the Wallet API, with the assistance of NetEnt if required. The tests include functional testing with relevant use cases (withdraw, deposit, rollbacks, balance inquiries).

Launch

Below are the activities involved in the launching process.

Workflow

#	Description
1.	NetEnt switches Seamless wallet mode on the production casino.
2.	The operator deploys new code, including the new Wallet API and any changes to the website.

NetEnt provides extended support for the first week after commercial go-live date. After that, the Customer Support team handles the regular support cases through support@netent.com.

Seamless Wallet Integration Reference

This section is designed to serve as a reference for the technical staff who performs the required integration work on the operator's side. You can view information about required external wallet API methods, get instructions for error handling and read about transaction handling scenarios.

There are two supported external wallet APIs in CasinoModule:

Release	Description	External Wallet API Information
Wallet server API 3.0	Introduced in CasinoModule release 9.0.	Wallet Server API 3.0 Reference
Wallet server API 2.1	Replaced by API 3.0 but still supported. All new integrations should be done towards API 3.0.	Wallet Server API 2.1 Reference

Important:

Wallet server API 1.1 is discontinued from the 9.0 release. For more information, contact your Technical Integration Manager, integration@netent.com.

Additional resource information is found in the following topics:

- Error Handling
- Transaction Handling Scenarios
- Customized Messages

Wallet Server API 3.0 Reference

Wallet server API 3.0 with its corresponding adapter is valid for all new customers and for any existing customers who upgrade to CasinoModule 9.0 or later.

Important:

All new integrations should be done towards the External Wallet API that corresponds to the specific CasinoModule release.

Additions to Wallet Server API 3.0

The new wallet server API 3.0 includes additions to the `deposit`, `withdraw`, and `withdrawAndDeposit` methods. The additions include returning information about:

- Issuing bonus program and deposition ID for `deposit` and `withdrawAndDeposit` calls with the reason type `WAGERED_BONUS`.
- Tournament ID and tournament occurrence ID for `deposit` and `withdrawAndDeposit` calls with the reason type `AWARD_TOURNAMENT_WIN`.
- Bonus bet amount in `withdraw` and `withdrawAndDeposit` calls for the reason type `GAME_PLAY`.
- Bonus win in `deposit` and `withdrawAndDeposit` calls for the reason type `GAME_PLAY`.
- Big win in `deposit` or `withdrawAndDeposit` calls for the reason types `GAME_PLAY` and `GAME_PLAY_FINAL`.
- Jackpot win amounts in deposit calls for the reason types `GAME_PLAY` and `GAME_PLAY_FINAL`.

- Jackpot contribution amounts in `withdraw` and `withdrawAndDeposit` calls for the reason types `GAME_PLAY` and `GAME_PLAY_FINAL`.
- Source and start date in `deposit` and `withdrawAndDeposit` calls for the reason type `CLEAR_HANGED_GAME_STATE`.
- Free rounds winnings and bets in `deposit` and `withdraw` calls with reason types `FREE_ROUND_PLAY` and `FREE_ROUND_FINAL`.

The operator's external wallet server must implement an API specified by the WSDL file received from NetEnt with the methods described below.

Note:

— Requests to any of the methods above include the session ID, if available.
— In case of a refund of a game round spanning over several player sessions, the session ID of the game rounds first game transaction is used.
— Calls with reason types `AWARD_TOURNAMENT_WIN` and `WAGERED_BONUS` do not include session IDs.
— All parameters may not be included in all calls. Parameters that are not always included are marked with "if applicable" in the descriptions below. For more information about your particular setup, contact your Technical Integration Manager.

deposit

This method makes a deposit to the player's account and returns the operator's transaction reference and player's account balance after the transaction. For example, this method can be used to collect a win, or collect a prize in a tournament. The `sessionId` might be empty or expired in the calls.

This method can be configured to be called also upon each free round win. This configuration is made by NetEnt staff.

To allow a flexible setup of bonus programs and tournaments, `bonusProgramId`, `depos-itionId` and `Group identifier` information is included in the calls if any bonus wagered money is deposited, and `tournamentId` and `tournamentOccurrenceId` are included if tournament bonus is wagered or tournament win is reported.

Bonus win amount information is included in the calls, giving operators the possibility to track bets and wins made with bonus money. Operators can select if transactions with zero amount should be reported, to indicate the bonus part, or not. The default is that only deposits with zero amounts are reported. To change this setting, contact your Technical Integration Manager.

Note:

You may have a different integration setup depending on your particular jurisdiction.

Deposited amounts may be fractions of cents (for example 2.75 cents), like jackpot wins or even regular wins in certain games. Therefore the operator's wallet system must accept deposits of fractions of cents.

The usage of this method is influenced by support for `withdrawAndDeposit` operations. If `withdrawAndDeposit` is supported, `withdrawAndDeposit` is used for all game-related transactions except when the 'big win' threshold is exceeded. Large wins are always handled with separate withdraw and deposit methods. Game-related transactions will always use the `GAME_PLAY` and `GAME_PLAY_FINAL` reason types. For these reason types, Big Win and jackpot amounts are always included in the call. If multiple jackpots are included, they are split up on different IDs.

For reason type `CLEAR_HANGED_GAME_STATE`, source and startDate information is returned.

Note:

`deposit` calls, unlike `withdraw` and `withdrawAndDeposit` calls, may be retried if the win amount exceeds the 'big win' threshold. All Seamless wallet transaction calls are idempotent, that is, will not change the initial call result for the method, so the same `transactionRef` value will be used for the first attempt as well as all retries. The wallet server must not process the deposit if a previous transaction with the same `transactionRef` exists on the operator side. If the transaction is successful, a positive response should be returned to the client. The deposit will be retried for a maximum of 10 days or until a response is received. For more information, see Development and Testing.

Important:

Setting the big win threshold to zero will effectively cause all **game play** deposit transactions to be retried rather than rolled back on errors. If the Casino Property 'seamlesswallet.disablerollback' is set to 'true' then **all** deposit transactions will be retried, including **Non-game play** deposit transactions, such as wagered bonus deposits and tournament winnings.

If `withdrawAndDeposit` is not supported, the `deposit` method is used for all transactions. For more information, see withdrawAndDeposit.

Method Signature

```
depositResponse deposit(
    String callerId,
    String callerPassword,
    String  playerName,
    Double amount,
    Object bonusPrograms,
    Tournaments tournaments,
    Boolean bigWin,
    Double jackpotAmount,
    Double bonusWin,
    String currency,
    Long transactionRef,
    Long gameRoundRef,
    String gameId,
    String reason,
    String source,
    DateTime startDate,
    String sessionId,
   Object  freeRoundWin
)
```

Parameters

- `callerId` - Identifier for the CasinoModule on the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with CasinoModule.
- `amount` - The amount to deposit into the player's account.
- `currency` - The three-character ISO code for the currency of the amount. The returned balance (in the `TransferResponse` object) is also in this currency.
- `transactionRef` - CasinoModule's reference for the transaction.
- `gameRoundRef` - CasinoModule's reference for the game round, if applicable (may be null).
- `gameId` - CasinoModule's identifier for the game, if applicable (may be null).

  Note:
  gameId will only be included for game-related transactions.

- `bonusPrograms` - Object containing `bonusProgramId`, `depositionId` and Group identifier for bonus wagered money, if applicable.
- `tournaments` - Object containing `tournamentOccurrenceId` and `tournamentId`, if applicable (may be null). All tournament occurrences relate to a specific tournament.
- `bonusWin` - Bonus win amount, if applicable. This amount is not included in the `amount` parameter. For free rounds, this parameter is used for the last free round call, which then contains the total of the accumulated winnings for the series of free rounds.
- `source` - Reason for clearing a game round, if applicable, for reason type `CLEAR_HANGED_GAME_STATE`.
- `startDate` - Start date for the game round, if applicable.
- `bigWin` - Information about if the deposit is a big win or not, if applicable.
- `jackpotAmount` - Amount that originates from a jackpot win, if applicable. The amount is included in the amount parameter.
- `freeRoundWin` - Object containing information about amount that is won in free round, along with bonus details and accumulated winnings for the current series of free rounds, if applicable. This parameter is used only when the casino is configured to call the `deposit` method upon each free round win.

  Note:
  Messages sent to the game client in the response are not supported for free round calls.

Example:

```
“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": 9
}
```

- `reason` - The reason for the deposit. The different reason types are:
  - `AWARD_TOURNAMENT_WIN`
    Happens if you use the tournaments functionality when paying "the prize" to the winners.
  - `WAGERED_BONUS`
    Used when bonus money is converted to real money and is paid to the wallet server.
  - `CLEAR_HANGED_GAME_STATE`
    Game rounds that are not completed by the players can be closed by the system or though the API with this reason type. The bet made is by default returned to the player.
    We have a "sanity check" feature. The value of the check is configurable. If the sanity check is triggered, it will perform a transaction rollback using a deposit call with reason type `CLEAR_HANGED_GAME_STATE` instead of using the `rollbackTransaction` method. The purpose of the sanity check is to prevent deposits of very large amounts if errors occur in the CasinoModule game servers.
  - `GAME_PLAY` (if `supportswithdrawanddeposit` setting is disabled)
    Occurs during game play if you do not use the combined `withdrawAndDeposit` method, or if the win amount exceeds the big win threshold.
  - `GAME_PLAY_FINAL` (if `supportswithdrawanddeposit` setting is disabled)
    Occurs during final game play (last transaction of the game round) if you do not use the combined withdrawAndDeposit method, or if the win amount exceeds the big win threshold.
  - `FREE_ROUND_PLAY` - Sent for all free rounds except for the last free round (which will be `FREE_ROUND_FINAL`)
  - `FREE_ROUND_FINAL` - Sent for the last free round, but only in any of these cases:
    - if 'ignoreZeroDepositsForFreeRoundBonus' is true and free round win is non zero.
    - if 'ignoreZeroDepositsForFreeRoundBonus' is false.
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real), if applicable. Valid for and reused during the complete player session.

Returns

- `depositResponse` - An object containing the parameters:
  - Double `balance` - The balance of the player's account after the transaction.

- String `transactionId` - The operator's reference for the transaction.
- String `message` - a customized message sent from Seamless Wallet server.

Type Definitions

`bonusPrograms`:

- bonus (may be repeated)
  - Long `bonusProgramId`
  - Long `depositionId`
- String `Group identifier` (only available if `giveFreeRounds` method is used).

tournaments:

- tournaments (may be repeated)
  - Long `tournamentId`
  - Long `tournamentOccurrenceId`

Throws in Case of Error

- `depositFault` - An object containing the parameter:
  - int `errorCode` - Possible error codes: 2=Illegal currency; 3=Negative deposit; 5=Authentication failed.
  - String `message` - A message describing the problem.

getBalance

This method returns the available balance in the player's account with the operator. It is called when games are loaded and while finishing unfinished game rounds. It may also be called during other events, such as back office queries or calls to the casino API.

The `sessionId` might be empty or expired in the calls.

Method Signature

```
getBalanceResponse  getBalance(
  String callerId,
  String callerPassword,
  String playerName,
  String currency,
  String gameId,
  String sessionId
)
```

Parameters

- `callerId` - Identifier for the CasinoModule on the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with CasinoModule.

- `currency` - The three-character ISO code for the expected balance.
- `gameId` - CasinoModule's identifier for the game , if applicable (may be null).
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real), if applicable. Valid for and reused during the complete player session.

Returns

- `getBalanceResponse` - An object containing the parameter:
  - Double `balance` - The available balance in the player's account.

Throws in Case of Error

- `getPlayerCurrencyFault` - An object containing the parameter.
  - int `errorCode` - Possible error codes: 2=Illegal currency; 5=Authentication failed.
  - String `message` - A message describing the problem.

getPlayerCurrency

This method returns the currency that the player should use when playing Seamless wallet games (the currency of the player's operator account). It is called the first time a player logs into the casino but may also be called during other events, such as back office queries. The `sessionId` might be empty in the calls.

Method Signature

```
getPlayerCurrencyResponse getPlayerCurrency(
  String callerId,
  String callerPassword,
  String playerName,
  String sessionId
)
```

Parameters

- `callerId`- CasinoModule's merchant ID for the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with CasinoModule.
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real), if applicable. Valid for and reused during the complete player session.

Returns

- `getPlayerCurrencyResponse` - object containing the parameter.
  - String `currencyISOCode` - The ISO Currency Code that the user should use when playing Seamless wallet games.

Throws in Case of Error

- `getPlayerCurrencyFault` - An object containing the parameter.
  - int `errorCode` - Possible error codes: 5=Authentication failed.
  - String `message` - A message describing the problem.

rollbackTransaction

If, for any reason, CasinoModule needs to reimburse the player for an already placed bet, it will try to roll back the withdrawal using this method until it succeeds.

Note:

Transaction rollbacks may be performed for withdrawals as well as deposits.

If the withdrawal has successfully gone through the external wallet server (that is, the `transactionRef` is known by the server), this method must re-deposit the amount of the bet to the player's account.

If the deposit has successfully gone through the external wallet server (that is, the `transactionRef` is known by the server), this method must remove the amount of the winning from the player's account.

This method must return a positive SOAP response regardless if the transaction was located or not. There are many scenarios where the transaction simply does not exist on the wallet server side, for instance due to network communication issues or wallet server failures. In these scenarios, the positive response from the wallet server will remove the transaction from the rollback queue on the CasinoModule side, and mark it as resolved.

Note:

In rare cases, if a withdrawal takes so much time that the CasinoModule call times out, we will send a `rollbackTransaction` request even though the external wallet server withdrawal method continues to execute. If this situation should occur, the external wallet server must never allow the same transaction reference to be used again.

Method Signature

```
rollbackTransactionResponse rollbackTransaction(
  String callerId,
  String callerPassword,
  Long  transactionRef
  String playerName
  String gameId,
  String sessionId
)
```

Parameters

- `callerId` - Identifier for the CasinoModule on the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `transactionRef` - CasinoModule's reference for the transaction.

- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with CasinoModule.
- `gameId` - CasinoModule's identifier for the game, if applicable (may be null).
- `sessionId`- ID returned from the CasinoModule web service (Playing for Real), if applicable. Valid for and reused during the complete player session.

Returns

- `rollbackTransactionResponse` - An object that does not contain any parameters.

Throws in Case of Error

- `rollbackTransactionFault` - An object containing the parameter:
  - int `errorCode` - Possible error codes: 5=Authentication failed.
  - String `message` - A message describing the problem.

withdraw

This method withdraws money from the player's account with the operator and returns the operator's transaction reference and the player's account balance after the transaction. For example, this method can be used to place a bet. The `sessionId` might be empty in the calls.

Bonus bet amount information is included in the calls, giving operators the possibility to track bets and wins made with bonus money. Operators can select if transactions with zero amount should be reported, to indicate the bonus part, or not. The default is that zero amounts are not reported. To change this setting, contact your Technical Integration Manager. You may have a different integration set-up depending on your particular jurisdiction.

This method can be configured to be called also upon each free round bet. This configuration is made by NetEnt staff.

The usage of this method is influenced by support for `withdrawAndDeposit` operations. If `withdrawAndDeposit` is supported, `withdrawAndDeposit` is used for all game-related transactions except when the "big win" threshold is exceeded. Large wins are always handled with separate `withdraw` and `deposit` methods. Game-related transactions will always use the `GAME_PLAY` and `GAME_PLAY_FINAL` reason types. For these reason types, jackpot contribution is always included in the call. If multiple jackpots are included, they are split up on different ID:s.

If `withdrawAndDeposit` is not supported, the `withdraw` method is used for all transactions. For more information, see withdrawAndDeposit.

Method Signature

```
withdrawResponse withdraw(
  String callerId,
  String callerPassword,
  String playerName,
  Double amount,
  Double bonusBet,
  Object jackpotContributions,
  String currency,
  Long transactionRef,
  Long gameRoundRef,
  String gameId,
  String reason,
  String sessionId,
  Object freeRoundBet
)
```

Parameters

- `callerId` - Identifier for the CasinoModule on the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with CasinoModule.
- `amount` - The amount to withdraw from the player's account.
- `bonusBet` - Bonus bet amount, if applicable. This amount is not included in the amount parameter.
- `JackpotContributions` - Jackpot amount, if applicable. If more than one jackpot is included, a jackpot ID is added.
- `currency` - The three-character ISO code for the currency of the amount. The returned balance (in the `TransferResponse` object) is also in this currency.
- `transactionRef` - CasinoModule's reference for the transaction.
- `gameRoundRef` - CasinoModule reference for the game round, if applicable (could be null).
- `gameId` - CasinoModule's identifier for the game, if applicable (may be null).

  Note:

  `gameId` will only be included for game-related transactions.

- `freeRoundBet` - Object containing information about amount that is bet in free round, along with bonus details, if applicable. This parameter is used only when the casino is configured to call the `withdraw` method upon each free round bet.
  Note that messages sent to the game client in the response are not supported for free round calls.

Example:

```
“freeRoundBet”:{
    “bonusProgramId”:123,
    "amount":1.0,
    “externalReferenceId”:”free round 1”,
    “bonusProgramName”:”FR1”,
    “bonusProgramDescription”:”Free round open to all”,
    “groupId”:”FRGroup 1”,
    “promotionCode”:”promo1”,
    "remainingRounds": 9
}
```

- `reason` - The reason for the withdrawal, if applicable. The different reason types are:
  - `GAME_PLAY` (if `supportswithdrawanddeposit` setting is disabled)
    Occurs during game play if you do not use the combined withdrawAndDeposit method.
  - `GAME_PLAY_FINAL` (if `supportswithdrawanddeposit` setting is disabled)
    Occurs during final game play (last transaction of the game round) if you do not use the combined withdrawAndDeposit method.
  - `FREE_ROUND_PLAY` - Sent for all free rounds, except for the last free round if 'ignoreZeroDepositsForFreeRoundBonus' is true and `freeRoundWin` is zero.
  - `FREE_ROUND_FINAL` - Sent for the last free round only if 'ignoreZeroDepositsForFreeRoundBonus' is true and `freeRoundWin` is zero (because `freeRoundBet` is never zero).
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real), if applicable. Valid for and reused during the complete player session.

Returns

- `withdrawResponse` - An object containing the parameter:
  - Double `balance` - The balance of the player's account after the transaction.
  - String `transactionId` - The operator's reference for the transaction.
  - String `message` - a customized message sent from Seamless Wallet server.

Type Definitions

`jackpotContributions:`

- jackpot (may be repeated)
  - String `jackpotId`
  - Double `contribution`

Throws in Case of Error

- `withdrawFault` - An object containing the parameter:
  - int `errorCode` - Possible error codes: 1=Not Enough Money; 2=Illegal currency; 4=Negative withdrawal; 5=Authentication failed; 6=Player limit exceeded.
  - String `message` - A message describing the problem.
  - Double `balance` - The available balance.

withdrawAndDeposit

For all game transactions, for example, placing a bet, collecting a win, or both, this method simultaneously withdraws and deposits money to the specified player's account in the specified currency. The withdraw amount can be the bet; while the deposit amount can be the win for the same game round.

`withdrawAndDeposit` provides a more efficient way of making deposits and withdrawals as it requires one call rather than two (that is, it replaces the `deposit` and `withdraw` methods for most game transactions). Process the withdrawal first. If the player does not have enough money for the withdrawal, immediately throw a WalletException with `errorCode` 1 for 'Not Enough Money'.

Note:

The `withdraw` and `deposit` methods must be implemented even if the `withdrawAndDeposit` method is used, because many transactions will use them. This includes tournament transactions, bonus transactions, and game transactions in some cases. For more information, see deposit and withdraw.

Method Signature

```
withdrawAndDepositResponse withdrawAndDeposit(
    String callerId,
    String callerPassword,
    String playerName,
    Double withdraw,
    Double deposit,
    Boolean bigWin,
    Double jackpotAmount,
    Double bonusWin,
    Double bonusBet,
    BonusPrograms bonusPrograms,
    Tournaments tournaments,
    Object jackpotContributions,
    String currency,
    Long transactionRef,
    Long gameRoundRef,
    String gameId,
    String reason,
    String source,
    DateTime startDate,
    String sessionId
)
```

Parameters

- `callerId` - Identifier for the CasinoModule on the external wallet server, as provided by the operator.

- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with CasinoModule.
- `withdraw` - The amount to withdraw from the player's account (must be a positive number).
- `deposit` -The amount to deposit to the player's account (must be a positive number).
- `bigWin` - Information about if the deposit is a big win or not, if applicable.
- `jackpotAmount` - Amount that originates from a jackpot win, if applicable.
- `bonusWin` - Bonus win amount, if applicable. This amount is not included in the amount parameter.
- `bonusBet` - The bonus bet amount, if applicable. This amount is not included in the amount parameter.
- `bonusPrograms` - Object containing deposition ID for bonus wagered money, if applicable.
- `tournaments` - Object containing `tournamentOccurrenceId` and `tournamentId`, if applicable (may be null). All tournament occurrences relate to a specific tournament.
- `jackpotContributions` - Jackpot amount, if applicable. If more than one jackpot is included, a `jackpotId` is added.
- `currency` - The three-character ISO code for the currency of the amount. The returned balance (in the `TransferResponse` object) must also be in this currency.
- `transactionRef` - CasinoModule's reference for this transaction.
- `gameRoundRef` - CasinoModule's reference for the game round, if applicable (may be null).
- `gameId` - CasinoModule's identifier for the game, if applicable (may be null).

  Note:
  `gameId` will only be included for game-related transactions.

- `reason` - The reason for the deposit, if applicable. The different reason types are:
  - `GAME_PLAY` (if `supportswithdrawanddeposit` setting is enabled) Happens during game play.
  - `GAME_PLAY_FINAL` (if `supportswithdrawanddeposit` setting is enabled) Happens during final game play.
  - `WAGERED_BONUS` Used when bonus money is converted to real money and is paid to the wallet server.
  - `AWARD_TOURNAMENT_WIN` Happens if you use the tournaments functionality when paying "the prize" to the winners.
- `startDate` - Start date for the game round, if applicable.
- `source` - Reason for clearing a game round, if applicable, for reason type `CLEAR_HANGED_GAME_STATE`.
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real), if applicable. Valid for and reused during the complete player session.

Returns

- `withdrawAndDepositResponse` - An object containing the parameters:
  - Double `newBalance` - The balance of the player's account after the transaction.

  - String `transactionId`- The operator's reference for the transaction.

Type Definitions

`bonusPrograms`:

- bonus (may be repeated)
  - Long `bonusProgramId`
  - Long `depositionId`

`tournaments`:

- tournaments (may be repeated)
  - Long `tournamentId`
  - Long `tournamentOccurrenceId`

`jackpotContributions`:

- jackpot (may be repeated)
  - String `jackpotId`
  - Double `contribution`

Throws in Case of Error

- `withdrawAndDepositFault` - An object containing the parameter:
  - int `errorCode` - Possible error codes: 1=Not Enough Money; 2=Illegal currency; 3=Negative deposit; 4=depositFault; 5=Authentication failed; 6=Player limit exceeded.
  - String `message` - A message describing the problem.
  - Double `balance` - The available balance.

Wallet Server API 2.1 Reference

This version of the Seamless wallet Server API, 2.1 (introduced in CasinoModule 5.6), is valid for CasinoModule release 5.6 or earlier. From release 9.0, the wallet server version 3.0 should be used.

Note:

Operators are recommended to stop using the Seamless wallet Server API version 1.1 and start using version 3.0 instead. The support for version 1.1 will be discontinued in the next major CasinoModule release. For more information, contact NetEnt's Customer Support: support@netent.com

Wallet server API version 2.1 has the following changes compared to version 2.0 (introduced in CasinoModule 5.5):

- Added new deposit transaction reason type `WAGERED_BONUS`.
- Removed obsolete deposit transaction reason type `FREE_ROUND_BONUS`.

Important:

All new integrations should be done towards the External Wallet API that corresponds to the specific CasinoModule release.

Important:

The operator's external wallet server must implement an API specified by the WSDL-file received from NetEnt AB with the methods described below.

Note:

Requests to any of the methods above include the session ID, if available.
- In case of a refund of a game round spanning over several player sessions, the session ID of the game rounds first game transaction is used.
- Calls with reason types `AWARD_TOURNAMENT_WIN` and `WAGERED_BONUS` do not include session IDs.

getPlayerCurrency

This method returns the currency that the player should use when playing Seamless wallet games (the currency of the player's operator account). It is called the first time a player logs into the casino but may also be called during other events, such as back office queries. The SessionId might be empty in the calls.

Method Signature

```
getPlayerCurrencyResponse getPlayerCurrency(
    String callerId,
    String callerPassword,
    String playerName,
    String sessionId
)
```

Parameters

- `callerId` - Identifier for the CasinoModule on the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with CasinoModule.
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real). Valid for and reused during the complete player session.

Returns

- `getPlayerCurrencyResponse` - object containing the parameter:
  - String `currencyISOCode` - The ISO Currency Code that the user should use when playing Seamless wallet games

Throws in Case of Error

- `getPlayerCurrencyFault` - An object containing the parameter:
  - Integer `errorCode` - Could be 5=Authentication failed
  - String `message` - A message describing the problem

getBalance

This method returns the available balance in the player's account with the operator. It is called when games are loaded from CasinoModule ADMIN TOOL, and also while finishing unfinished game rounds. It may also be called during other events, such as back office queries. The `ses-sionId` might be empty or expired in the calls.

Method Signature

```
getBalanceResponse getBalance(
    String callerId,
    String callerPassword,
    String playerName,
    String currency,
    String gameId,
    String sessionId
)
```

Parameters

- `callerId` - Identifier for the CasinoModule on the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with CasinoModule.
- `currency` - The three-character ISO code for the expected balance.
- `gameId` - CasinoModule's identifier for the game. (may be null)

- `sessionId` - ID returned from the CasinoModule web service (Playing for Real). Valid for and reused during the complete player session.

Returns

- `getBalanceResponse` - An object containing the parameter:
  - Double `balance` - The available in the player's account.

Throws in Case of Error

- getBalanceFault - An object containing the parameter:
  - Integer `errorCode` - Could be 2=Illegal currency or 5=Authentication failed
  - String `message` - A message describing the problem

withdraw

This method withdraws money from the player's account with the operator and returns the operator's transaction reference and the player's account balance after the transaction. For example, this method can be used to place a bet. The sessionId might be empty in the calls.

The usage of this method is influenced by support for `withdrawAndDeposit` operations. If `withdrawAndDeposit` is supported, `withdrawAndDeposit` is used for all game-related transactions except when the "big win" threshold is exceeded. Large wins are always handled with separate `withdraw` and `deposit` methods. Game-related transactions will always use the GAME_PLAY and `GAME_PLAY_FINAL` reason types.

If `withdrawAndDeposit` is not supported, the withdraw method is used for all transactions. For more information, see withdrawAndDeposit.

Method Signature

```
withdrawResponse withdraw(
    String callerId,
    String callerPassword,
    String playerName,
    Double amount,
    String currency,
    String transactionRef,
    String gameRoundRef,
    String gameId,
    String reason,
    String sessionId
)
```

Parameters

- `callerId` - Identifier for the CasinoModule on the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.

- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with the CasinoModule.
- `amount` - The amount to withdraw from the player's account.
- `currency` - The three-character ISO code for the currency of the amount. The returned balance (in the `TransferResponse` object) is also in this currency.
- `transactionRef` - CasinoModule's reference for the transaction.
- `gameRoundRef` - CasinoModule reference for the game round, if applicable (could be null).
- `gameId` - CasinoModule's identifier for the game.

Note:
`gameId` will only be included for game-related transactions.

- `reason` - The different reason types are:
  - `GAME_PLAY` (if `supportswithdrawanddeposit` setting is disabled)
    Happens during game play if you do not use the combined withdrawAndDeposit method.
  - `GAME_PLAY_FINAL` (if `supportswithdrawanddeposit` setting is dis-abled)
    Happens during final game play (last transaction of the game round) if you do not use the combined withdrawAndDeposit method.
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real). Valid for and reused during the complete player session.

Returns

- `withdrawResponse` - An object containing the parameter:
  - Double `balance` - The balance of the player's account after the transaction.
  - Long `transactionId` - The operator's reference for the transaction.

Throws in Case of Error

- `withdrawFault` - An object containing the parameter:
  - Integer `errorCode` - Could be 1=Not Enough Money 2=Illegal currency 4=d-depositFault 5=Authentication failed or 6=Player limit exceeded.
  - String `message` - A message describing the problem.
  - Double `balance` - The available balance.

deposit

This method makes a deposit to the player's account and returns the operator's transaction reference and player's account balance after the transaction. For example, this method can be used to collect a win, or collect a prize in a tournament. The `sessionId` might be empty or expired in the calls.

The usage of this method is influenced by support for `withdrawAndDeposit` operations. If `withdrawAndDeposit` is supported, `withdrawAndDeposit` is used for all game-related transactions except when the "big win" threshold is exceeded. Large wins are always handled wit h separate `withdraw` and `deposit` methods. Game-related transactions will always use the GAME_PLAY and `GAME_PLAY_FINAL` reason types.

Note:

Deposit calls, unlike `withdraw` and `withdrawAndDeposit` calls, may be retried if the win amount exceeds the "big win" threshold. If this is the case the same `transactionRef` value will be used for the first attempt as well as all retries. The wallet API must not process the deposit if a previous transaction with the same `transactionRef` exists on the operator side. If the transaction is successful, a positive response should be returned to the client. The deposit will be retried for a maximum of 10 days or until a response is received.

If `withdrawAndDeposit` is not supported, the `deposit` method is used for all transactions. For more information, see withdrawAndDeposit.

Method Signature

```
depositResponse deposit(
      String callerId,
      String callerPassword,
      String playerName,
      Double amount,
      String currency,
      StringtransactionRef,
      String gameRoundRef,
      String gameId,
      String reason,
      String sessionId
)
```

Parameters

- `callerId` - Identifier for the CasinoModule on the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `playerName` - The user name of the player. This is be the user name the operator supplied when registering the player with CasinoModule.
- `amount` - The amount to deposit into the player's account.
- `currency` - The three-character ISO code for the currency of the amount. The returned balance (in the `TransferResponse` object) is also in this currency.
- `transactionRef` - CasinoModule's reference for the transaction.
- `gameRoundRef` - CasinoModule's reference for the game round, if applicable (may be null).
- `gameId` - CasinoModule's identifier for the game.

Note:

`gameId` will only be included for game-related transactions.

- `reason` - The reason for the deposit. The different reason types are:
  - `AWARD_TOURNAMENT_WIN`
    Happens if you use the tournaments functionality when paying "the prize" to the winners.

  - `WAGERED_BONUS`
    Used when bonus money is converted to real money and is paid to the wallet server.
  - `CLEAR_HANGED_GAME_STATE`
    On a six months rolling basis we "close" game rounds that are unfinished. This means that we "finish the game round" and pay back the bet that was made to the player. We have a "sanity check" feature. The value of the check is configurable. If the sanity check is triggered, it will perform a transaction rollback using a `deposit` call with reason type `CLEAR_HANGED_GAME_STATE` instead of using the `rollbackTransaction` method. The purpose of the sanity check is to prevent deposits of very large amounts if errors occur in the CasinoModule game servers.
  - `GAME_PLAY` (if `supportswithdrawanddeposit` setting is disabled)
    Happens during game play if you do not use the combined `withdrawAndDeposit` method, or if the win amount exceeds the big win threshold.
  - `GAME_PLAY_FINAL` (if `supportswithdrawanddeposit` setting is disabled)
    Happens during final game play (last transaction of the game round) if you do not use the combined withdrawAndDeposit method, or if the win amount exceeds the big win threshold.
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real). Valid for and reused during the complete player session.

Returns

- `depositResponse` - An object containing the parameters:
  - Double `balance` - The balance of the player's account after the transaction.
  - Long `transactionId` - The operator's reference for the transaction.

Throws in Case of Error

- `depositFault` - An object containing the parameter:
  - Integer `errorCode` - Could be 2=Illegal currency 3=Negative deposit or 5=Authentiication failed
  - String `message` - A message describing the problem

withdrawAndDeposit

For all game transactions, for example, placing a bet, collecting a win, or both, this method simultaneously withdraws and deposits money to the specified player's account in the specified currency. The withdraw amount can be the bet; while the deposit amount can be the win for the same game round.

`withdrawAndDeposit` provides a more efficient way of making deposits and withdrawals as it requires one call rather than two (that is, it replaces the deposit and withdraw methods for most game transactions). Process the withdrawal first. If the player does not have enough money for the withdrawal, immediately throw a WalletException with `errorCode` 1 for 'Not Enough Money'.

Note:

The withdraw and deposit methods must be implemented even if the `withdrawAndDeposit` method is used, because many transactions will use them. This includes tournament transactions, bonus transactions, and game transactions in some cases. For more information, see deposit and withdraw.

Method Signature

```
withdrawAndDepositResponse withdrawAndDeposit(
    String callerId,
    String callerPassword,
    Double withdraw,
    Double deposit,
    String currency,
    String transactionRef,
    String gameRoundRef,
    String gameId,
    String reason,
    String sessionId
)
```

Parameters

- `callerId` - CasinoModule's merchant ID for the external wallet server, as provided by the operator.
- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with the CasinoModule.
- `withdraw` - The amount to withdraw from the player's account (must be a positive number).
- `deposit` -The amount to deposit to the player's account (must be a positive number).
- `currency` - The three-character ISO code for the currency of the amount. The returned balance (in the `TransferResponse` object) must also be in this currency.
- `transactionRef` - CasinoModule's reference for this transaction.
- `gameRoundRef` - CasinoModule's reference for the game round, if applicable (may be null).
- `gameId` - CasinoModule's identifier for the game.

Note:
gameId will only be included for game-related transactions.

- `reason` - The reason for the deposit. The different reason types are:
  - `GAME_PLAY` (if `supportswithdrawanddeposit` setting is enabled) Happens during game play.
  - `GAME_PLAY_FINAL` (if `supportswithdrawanddeposit` setting is enabled) Happens during final game play.
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real). Valid for and reused during the complete player session.

Returns

- `withdrawAndDepositResponse` - An object containing the parameters:
  - Double `newBalance` - The balance of the player's account after the transaction.

- Long `transactionId` - The operator's reference for the transaction.

Throws in Case of Error

- `withdrawAndDepositFault` - An object containing the parameter:
  - Integer `errorCode` - Could be 1=Not Enough Money 2=Illegal currency 3=Negative deposit 4=depositFault 5=Authentication failed or 6=Player limit exceeded.
  - String `message` - A message describing the problem.
  - Double `balance` - The available balance.

rollbackTransaction

If, for any reason, CasinoModule needs to reimburse the player for an already placed bet, it will try to roll back the withdrawal using this method until it succeeds.

Note:
Transaction rollbacks may be performed for withdrawals as well as deposits.

If the withdrawal has successfully gone through the external wallet server (that is, the `transactionRef` is known by the server), this method must re-deposit the amount of the bet to the player's account.

If the deposit has successfully gone through the external wallet server (that is, the `trans-actionRef` is known by the server), this method must remove the amount of the winning from the player's account.

This method must return a positive SOAP response regardless if the transaction was located or not. There are many scenarios where the transaction simply does not exist on the wallet server side, for instance due to network communication issues or wallet server failures. In these scenarios, the positive response from the wallet server will remove the transaction from the rollback queue on the CasinoModule side, and mark it as resolved.

Note:
In rare cases, if a withdrawal takes so much time that the CasinoModule call times out, we will send a `rollbackTransaction` request even though the external wallet server withdrawal method continues to execute. If this situation should occur, the external wallet server must never allow the same transaction reference to be used again.

Method Signature

```
rollbackTransactionResponse rollbackTransaction(
        String callerId,
        String callerPassword,
        Long  transactionId
        String playerName
        String gameId,
        String sessionId
)
```

Parameters

- `callerId` - CasinoModule's merchant ID for the external wallet server, as provided by the operator.

- `callerPassword` - CasinoModule's password for the external wallet server, as provided by the operator.
- `transactionId` - CasinoModule's reference for the transaction.
- `playerName` - The user name of the player. This is the user name the operator supplied when registering the player with CasinoModule.
- `gameId` - CasinoModule's identifier for the game. (may be null).
- `sessionId` - ID returned from the CasinoModule web service (Playing for Real). Valid for and reused during the complete player session.

Returns

- `rollbackTransactionResponse` - An object that does not contain any parameters.

Throws in Case of Error

- `rollbackTransactionFault` - An object containing the parameter:
  - Integer `errorCode` - Could be 5=Authentication failed.
  - String `message` - A message describing the problem.

Error Handling

The external wallet server's API should throw errors in the circumstances listed below. An error is thrown by returning a fault/exception object (*), containing an **`errorCode`** integer and message.

For some games, it is possible for the operator to send a customized error message that will be displayed for the player in the game client. See section Customized Messages for more details.

Error code	Error message (can be customized)	When to throw	Error message displayed in game client (the exact wording of the messages may differ between desktop games and Touch games)
1	Not Enough Money	When there is not enough money in the player's account to complete the transaction.	'Not enough money in your account'
2	Illegal currency	When the currency is incorrect.	'Technical error, please try again later'
3	Negative deposit	If the amount is negative.	'Technical error, please try again later'
4	Negative withdrawal	If the amount is negative.	'Technical error, please try again later'

Error code	Error message (can be customized)	When to throw	Error message displayed in game client (the exact wording of the messages may differ between desktop games and Touch games)
5	Authentication failed	When `callerId` or `callerPassword` is incorrect.	'Your account could not be reached right now, please try again later'
6	Player limit exceeded	If the player's limit has been exceeded.	'You have reached your play limit. Welcome back later.'
Any other integer value	General error	On any validation error or runtime error.	'Technical error, please try again later'

Any API call that returns a valid fault object will not trigger a transaction rollback or retry. If you want to trigger a transaction rollback, respond with broken output such as an http 500 server error or blank output.

That is, any unknown error, including time-outs, will result in an a rollback of the Seamless wallet transaction.

If the transaction is marked as a big win, no rollback will be performed. CasinoModule will try again until the transaction is successful.

Note:
The transaction will be retried for a maximum of 10 days or until a response is received.

Transaction Handling Scenarios

API Withdraw Call Fails with Known Exception

- The CasinoModule game transaction will be rolled back and the transaction will be set to failed.

API Withdraw Call Fails with Unknown Exception

For example, IOException

- The CasinoModule game transaction will be rolled back.
- CasinoModule will try to rollback the transaction by calling the `rollbackTransaction` method until it succeeds.

CasinoModule Throws an Exception after a Successful Call to the API Withdraw Method

For example, RuntimeException

- The CasinoModule game transaction will be rolled back.
- CasinoModule will try to rollback the transaction by calling the API `rollbackTransaction` method until it succeeds.

API Deposit Call Fails with Any Exception

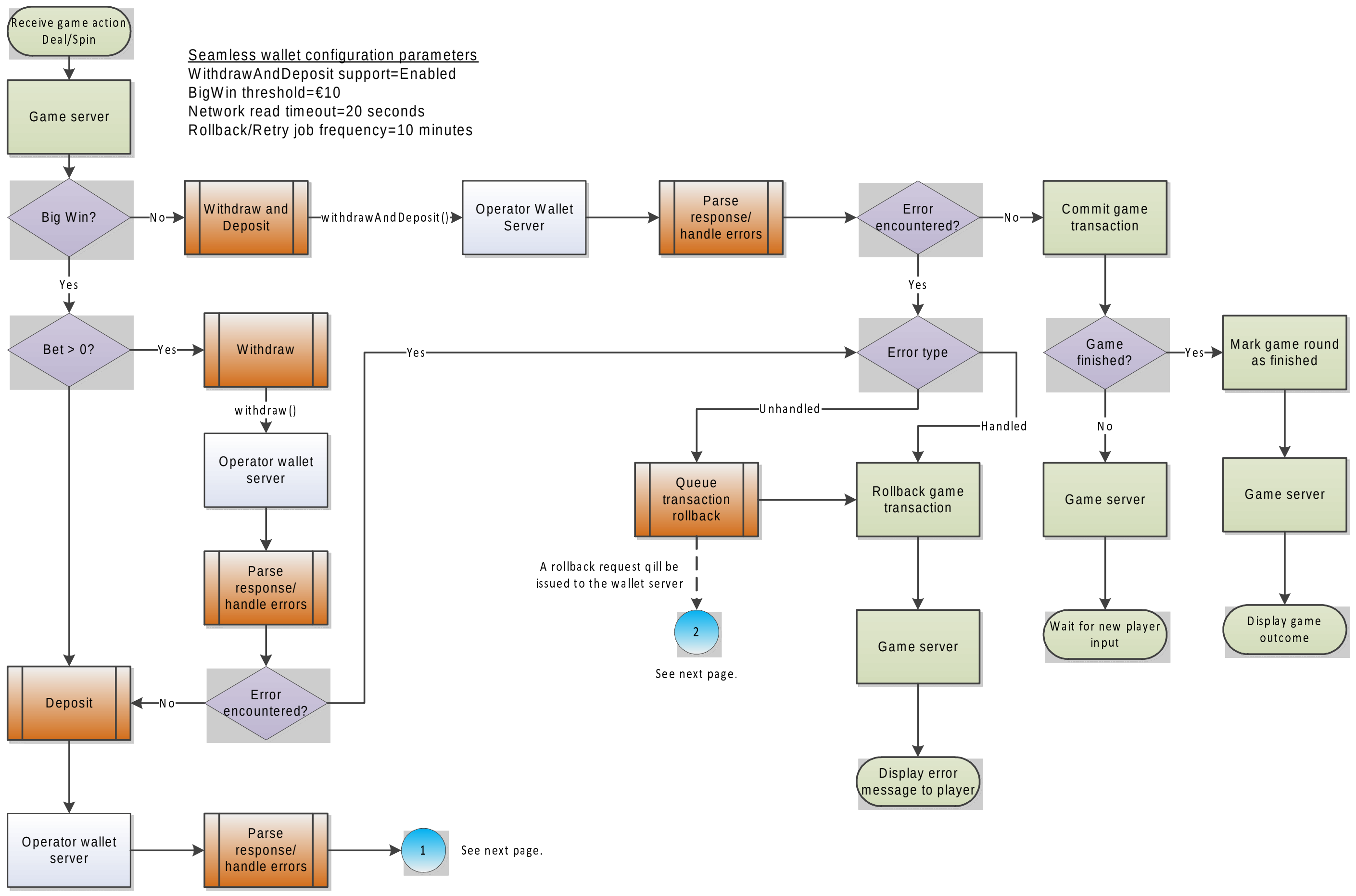
- CasinoModule will either rollback the transaction using the `rollbackTransaction` method, or retry it using the deposit method. The method used depends on the configured big win threshold and the transaction amount.

API Calls for a Transaction that has Already Successfully Gone Through

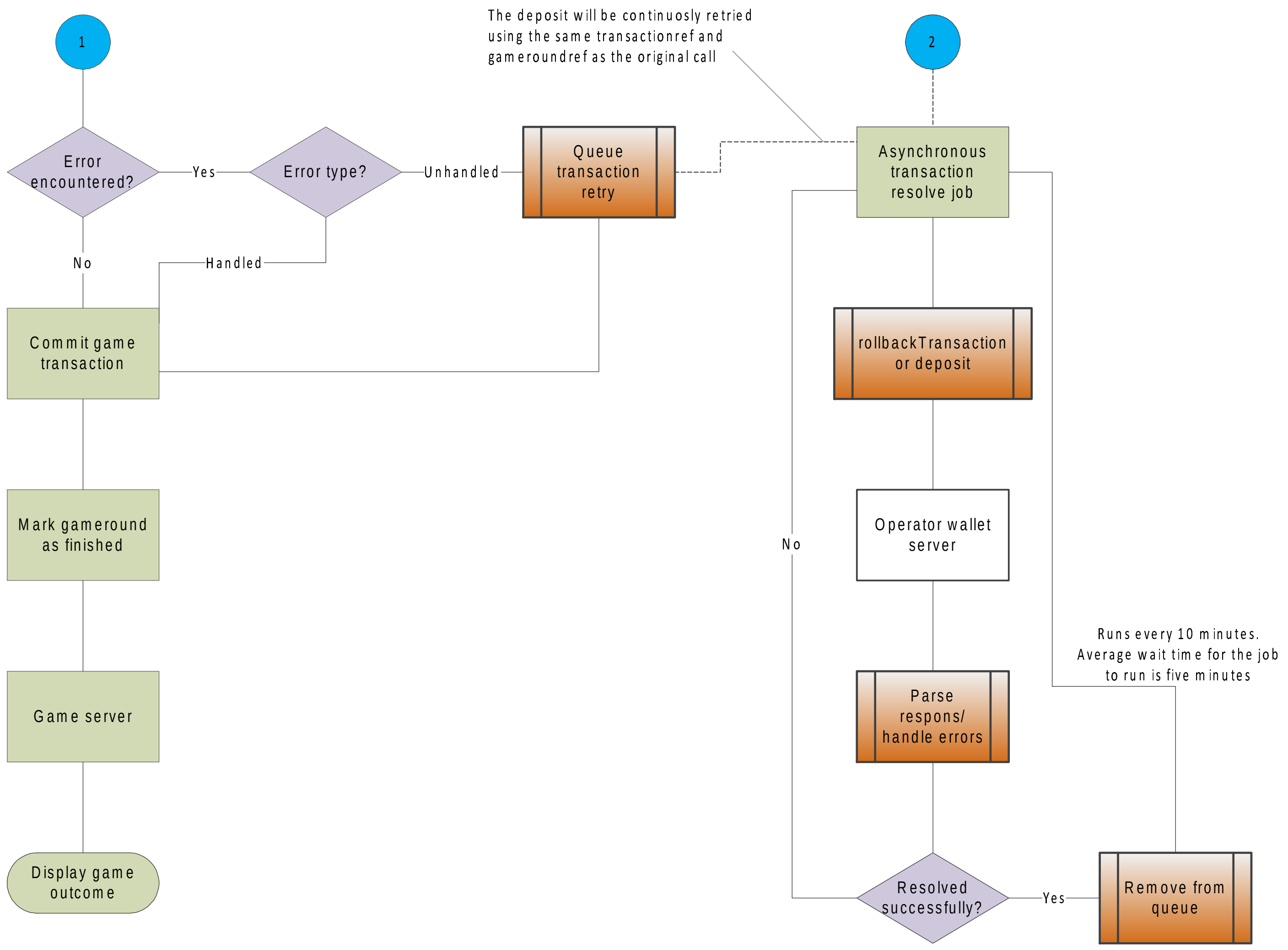
- If the wallet server receives a transaction using a `transactionRef` that has already been used, it is excepted to respond with a Response object containing its original `trans-actionId` for the transaction, and the current account balance.

Seamless Wallet Error Handling

The following flowcharts describe error handling when using the Seamless wallet.



Seamless wallet configuration parameters
WithdrawAndDeposit support=Enabled
BigWin threshold=€10
Network read timeout=20 seconds
Rollback/Retry job frequency=10 minutes


1
Error encountered?
Yes
Error type?
Unhandled
Queue transaction retry
Handled
No
Commit game transaction
Mark gameround as finished
Game server
Display game outcome
The deposit will be continuosly retried using the same transactionref and gameroundref as the original call
2
Asynchronous transaction resolve job
rollbackTransaction or deposit
Operator wallet server
Parse respons/ handle errors
Resolved successfully?
No
Yes
Remove from queue
Runs every 10 minutes. Average wait time for the job to run is five minutes

Customized Messages

The CasinoModule system has a feature that makes it possible for the operator to send a customized message (error or information) with their own content. The customized message will be displayed directly in the player's game client. Sending a customized message can greatly increase the player's user experience when playing games.

Note:

Not all games are supported to utilise this feature, and some games support one method but not the others. For specific information about which games that support customized messages, please contact support@netent.com.

Depending on the desired game client behaviour, four different types of customized message boxes can be displayed in the game client during game play. See tables below for details about the different message types.

As long as the message box is displayed on the screen, it will not be possible for the player to perform any interaction with the game. Depending on the shown message box type, the player will be forced to click on the **Home, Close** or **Continue** button to resume interaction with the game client.

Customized messages have a total maximum length of 1000 characters. The characters will be distributed in 3 rows with 50 characters in each row. Only characters from the ISO Latin-1 character set are supported in customized messages.

If desired it is also possible to include one HTML hyperlink in the customized message.

The customized message is sent utilising the `withdraw`, `deposit` or `withdrawAndDeposit` API method to call the operator's wallet server. The message can then be incorporated in the response call from the operator's wallet server.

Please refer to section Send a Customized Message to see API examples on how to use this feature. You can also find more information in the latest provided WSDL file.

Customized Messages for Touch Games

Error code	When is the Message Displayed	When to use this Message Type (Examples)	API Method to Utilise	User Interaction Button and Game Client behaviour
990	Immediately	The player cannot continue game play for real money.	`withdraw`	**Home** button ▪ Game play is terminated immediately. Player can return to lobby.
991	Immediately	Sorry, you can't perform this action. You have 0.5€ remaining of your daily wagering limit. You can change this limit in "My Account".	`withdraw`	**Continue** button ▪ Game play can be resumed when the cause to the interruption has been corrected.
992	After finished game round. Message will be triggered only when reason type is 'GAME_PLAY_FINAL'.	Player time limit has been exceeded.	`withdraw, deposit or withdrawAndDeposit`	**Home** button ▪ Game play is terminated immediately. Player can return to lobby.
993	After finished game round. Message will be triggered only when reason type is 'GAME_PLAY_FINAL'.	Tax information message.	`withdraw, deposit or withdrawAndDeposit`	**Continue** button ▪ Game play will be resumed when the message box is closed.

Customized Messages for Desktop Games

Error code	When is the Message Displayed	When to use this Message Type (Examples)	API Method to Utilise	User Interaction Button and Game Client behaviour
990	Immediately	The player cannot continue game play for real money.	`withdraw`	No button, only text ■ Game play is terminated immediately. ■ Operator to provide navigation so that player can return to game lobby.
991	Immediately	Sorry, you can’t perform this action. You have 0.5€ remaining of your daily wagering limit. You can change this limit in "My Account".	`withdraw`	**Continue** button ■ Game play can be resumed when the cause to the interruption has been corrected.
992	After finished game round. Message will be triggered only when reason type is 'GAME_PLAY_FINAL'.	Player time limit has been exceeded.	`withdraw, deposit or withdrawAndDeposit`	No button, only text ■ Game play is terminated immediately. ■ Operator to provide navigation so that player can return to game lobby.
993	After finished game round. Message will be triggered only when reason type is 'GAME_PLAY_FINAL'.	Tax information message.	`withdraw, deposit or withdrawAndDeposit`	**Continue** button ■ Game play will be resumed when the message box is closed.

Note:

For personal assistance when creating customized messages, please contact support@netent.com.

Seamless Wallet SOAP API Examples

This section lists examples of Seamless wallet SOAP API calls with in-line sequence diagrams.

Note:

CasinoModule has configuration settings that affect the API call sequence as well as the number of calls made.

- The `withdrawAndDeposit` call can be enabled to reduce the number of transactions. This call combines withdraw and deposit methods into a single API call, and into a single transaction. Many games are stateless (meaning that the game outcome is known immediately when the bet is placed) and can be finalized with a single `withdrawAndDeposit` call.

  Enabling this call has the result that all game-play calls (reason types `GAME_PLAY` and `GAME_PLAY_FINAL`) use the `withdrawAndDeposit` method, except ones where the win amount exceeds the pre-defined big win threshold value. Game rounds that result in a big win will always use separate deposit and withdraw methods. Non-game play calls, such as wagered bonus deposits and tournament winnings, will always use deposit calls.
- Balance caching can be enabled to reduce the number of `getBalance` calls. CasinoModule will cache player balances for 30 seconds by default. This functionality can be disabled by setting the cache to zero seconds.

  Note:

  CasinoModule will query the wallet server for an updated balance before each bet, with the result that the balance cache will reduce the number of API calls significantly.

  The `getBalance` call that is performed before each bet are not included in the examples.

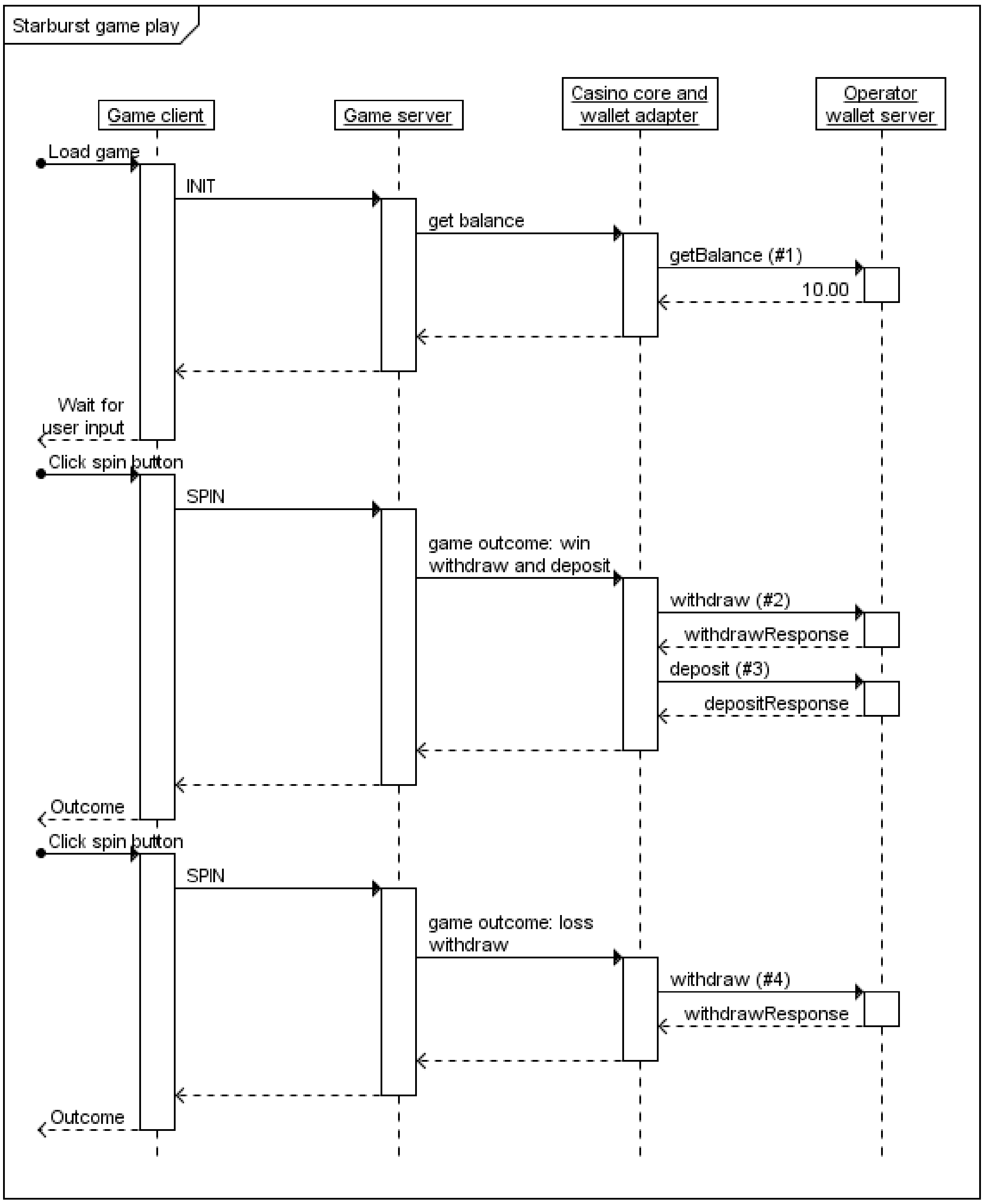
The following Seamless wallet API usage examples are described:

- Starburst Game Play
- Blackjack Game Play
- Clear Hanged Game State
- Rollback and Retry of Transactions
- Tournament Win
- Withdraw and Deposit
- Send a Customized Message

Starburst Game Play

This example shows basic game play in a stateless game. Most video slots are stateless, as well as the roulette games. In a stateless game, the game server will know the game outcome as soon as the bet has been placed, and the winnings will be deposited to the wallet server immediately after the bet has been withdrawn.

The following diagram shows the example flow:



Starburst game play

Methods

getBalance (#1)

This method retrieves the player’s balance in the specified currency. The gameId and sessionId parameters are supplied on all game play-related calls.

Request:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:getBalance xmlns:ns3="http://types.walletserver.casinomodule.com/2_0/"
      xmlns:ns2="walletserver:ext_2_1">
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>netent_test</playerName>
         <currency>EUR</currency>
         <gameId>starburst_sw</gameId>
         <ns2:sessionId>1357498797399-339-P0FSA2J3QY1GJ</ns2:sessionId>
      </ns3:getBalance>
   </S:Body>
</S:Envelope>
```

Response:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns2="http://types.walletserver.casinomodule.com/2_0/">
   <SOAP-ENV:Body>
      <ns2:getBalanceResponse>
         <balance>10</balance>
      </ns2:getBalanceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

withdraw (#2)

This method withdraws the bet amount from the player’s account. The gameId and sessionId parameters are supplied for reference and can be stored on the operator's side for reporting purposes.

Note:

The reason parameter is `GAME_PLAY`, this means that there will be additional transactions in this game round. A game round can consist of more than one withdraw call.

Request:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:withdraw xmlns:ns3="http://types.walletserver.casinomodule.com/2_0/"
      xmlns:ns2="walletserver:ext_2_1">
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>netent_test</playerName>
         <amount>1.5</amount>
         <bonusBet>0</bonusBet>
         <currency>EUR</currency>
         <transactionRef>10295</transactionRef>
         <gameRoundRef>8309</gameRoundRef>
         <gameId>starburst_sw</gameId>
         <reason>GAME_PLAY</reason>
         <ns2:sessionId>1357498797399-339-P0FSA2J3QY1GJ</ns2:sessionId>
      </ns3:withdraw>
   </S:Body>
</S:Envelope>
```

Response:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:withdrawResponse xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
         <balance>8.50</balance>
         <transactionId>1872067161</transactionId>
      </ns2:withdrawResponse>
   </S:Body>
</S:Envelope>
```

deposit (#3)

This method deposits the winnings to the player's account.

Note:

The `gameRoundRef` is the same as in the withdraw call in withdraw (2). The `transactionRef`, which is a unique transaction identifier, is different.

The reason parameter is now `GAME_PLAY_FINAL` indicating that the game round has ended and no more `GAME_PLAY` or `GAME_PLAY_FINAL` calls will be made for this gameround.

This is also a bigwin type transaction, to illustrate the use of this parameter.

Request:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns3="http://types.walletserver.casinomodule.com/3_0/">
   <S:Body>
      <ns3:deposit>
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>netent_test</playerName>
         <amount>3</amount>
         <bigWin>true</bigWin>
         <jackpotAmount></jackpotAmount>
         <bonusWin>0</bonusWin>
         <currency>EUR</currency>
         <transactionRef>10316</transactionRef>
         <gameRoundRef>8309</gameRoundRef>
         <gameId>starburst_sw</gameId>
         <reason>GAME_PLAY_FINAL</reason>
         <sessionId>1357498797399-339-P0FSA2J3QY1GJ</sessionId>
         <tournamentOccurrenceId></tournamentOccurrenceId>
         <bonusprograms><!--Zero or more repetitions:-->
            <bonus>
               <bonusprogramid></bonusprogramid>
               <depositionid></depositionid>
            </bonus>
         </bonusprograms>
         <source></source>
         <startDate>2012-12-11 09:23:02.000</startDate>
      </ns3:deposit>
   </S:Body>
</S:Envelope>
```

Response:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:depositResponse xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
         <balance>11.5</balance>
         <transactionId>1872067162</transactionId>
      </ns2:depositResponse>
   </S:Body>
</S:Envelope>
```

withdraw (#4)

In this example, the player spins again, but does not win this time. A single withdraw call is made, and the reason parameter is set to `GAME_PLAY_FINAL`.

Request:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns3="http://types.walletserver.casinomodule.com/3_0/">
   <S:Body>
      <ns3:withdraw>
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>netent_test</playerName>
         <amount>1.5</amount>
         <bonusBet>0</bonusBet>
         <currency>EUR</currency>
         <transactionRef>10317</transactionRef>
         <gameRoundRef>8318</gameRoundRef>
         <gameId>starburst_sw</gameId>
         <reason>GAME_PLAY_FINAL</reason>
         <sessionId>1357498797399-339-P0FSA2J3QY1GJ</sessionId>
      </ns3:withdraw>
   </S:Body>
</S:Envelope>
```

Response:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:withdrawResponse xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
         <balance>10.00</balance>
         <transactionId>1872067163</transactionId>
      </ns2:withdrawResponse>
   </S:Body>
</S:Envelope>
```

Blackjack Game Play

This example shows basic game play in Blackjack.

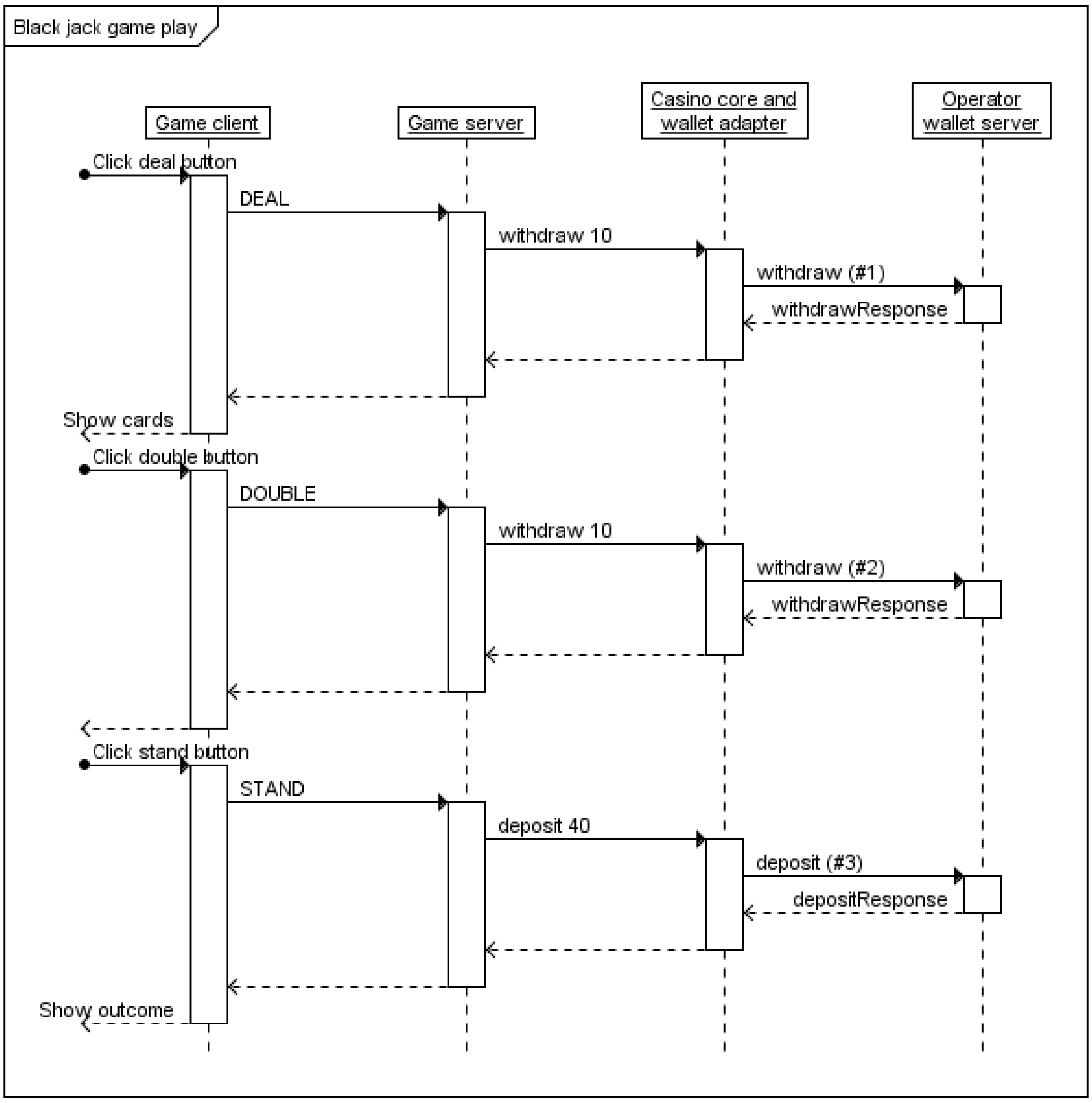
Blackjack is a stateful game, so there will be a delay between withdraws and deposit calls, while waiting for player decisions and input.

The initial `getBalance` call is not included in this and the following examples as it is identical between game play scenarios.

Note:

The final deposit call will be performed even if the player doesn't win. If this is the case, the amount will be zero, `transactionRef` will .still have a unique value, and the reason parameter will be `GAME_PLAY_FINAL.`

The following diagram shows the example flow:



Blackjack game play

Methods

withdraw (#1)

Request:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
   <S:Body>
      <ns2:withdraw>
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>netent_test</playerName>
         <amount>10</amount>
         <bonusBet>0</bonusBet>
         <currency>EUR</currency>
         <transactionRef>10321</transactionRef>
         <gameRoundRef>8321</gameRoundRef>
         <gameId>blackjack2-3h_sw</gameId>
         <reason>GAME_PLAY</reason>
         <sessionId>1357498797399-339-P0FSA2J3QY1GJ</sessionId>
      </ns2:withdraw>
   </S:Body>
</S:Envelope>
```

Response:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:withdrawResponse
      xmlns:ns2="http://types.walletserver.casinomodule.com/3ns2/">
         <balance>100</balance>
         <transactionId>1900140875</transactionId>
      </ns2:withdrawResponse>
   </S:Body>
</S:Envelope>
```

withdraw (#2)

Request:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
   <S:Body>
      <ns2:withdraw>
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>netent_test</playerName>
         <amount>10</amount>
         <bonusBet>0</bonusBet>
         <currency>EUR</currency>
         <transactionRef>10322</transactionRef>
         <gameRoundRef>8321</gameRoundRef>
         <gameId>blackjack2-3h_sw</gameId>
         <reason>GAME_PLAY</reason>
         <sessionId>1357498797399-339-P0FSA2J3QY1GJ</sessionId>
      </ns2:withdraw>
   </S:Body>
</S:Envelope>
```

Response:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:withdrawResponse xmlns:n-
      s2="http://types.walletserver.casinomodule.com/3ns2/">
         <balance>90</balance>
         <transactionId>1900140876</transactionId>
      </ns2:withdrawResponse>
   </S:Body>
</S:Envelope>
```

deposit (#3)

Request:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
   <S:Body>
      <ns2:deposit>
         <callerId>testcaller</callerId>
         <playerName>netent_test</playerName>
         <amount>40</amount>
         <jackpotAmount></jackpotAmount>
         <bonusWin></bonusWin>
         <currency>EUR</currency>
         <transactionRef>10323</transactionRef>
         <gameRoundRef>8321</gameRoundRef>
         <gameId>blackjack2-3h_sw</gameId>
         <reason>GAME_PLAY_FINAL</reason>
         <sessionId>1357498797399-339-P0FSA2J3QY1GJ</sessionId>
         <tournamentOccurrenceId></tournamentOccurrenceId>
         <bonusprograms><!--Zero or more repetitions:-->
            <bonus>
               <bonusprogramid></bonusprogramid>
               <depositionid></depositionid>
            </bonus>
         </bonusprograms>
         <source></source>
         <startDate>2013-01-08 12:20:01.000</startDate>
      </ns2:deposit>
   </S:Body>
</S:Envelope>
```

Response:

Example:

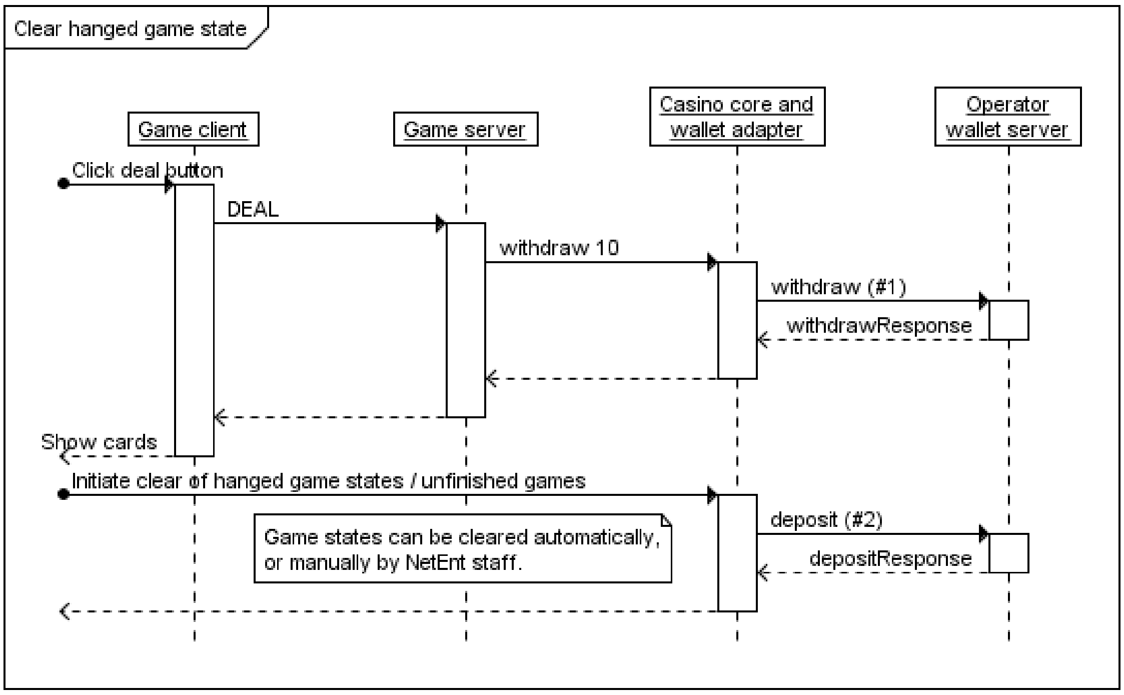
```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:depositResponse xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
         <balance>130</balance>
         <transactionId>1900140877</transactionId>
      </ns2:depositResponse>
   </S:Body>
</S:Envelope>
```

Clear Hanged Game State

CasinoModule has the functionality to forcefully end unfinished game rounds. This may be done when the software is updated, normally for some game types in major and minor updates, and when game rounds reach a certain age for data life-cycle reasons, but not for patch updates. Some jurisdictions require that unfinished games are forcefully ended after a certain period of time.

When this occurs the bet amount will be refunded to the player, using a deposit call with reason=`CLEAR_HANGED_GAME_STATE`.

The following diagram shows the example flow:



Clear hanged game state

Methods

withdraw (#1)

This is a single withdraw call, made when the player places his bet.

Request:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
   <S:Body>
      <ns2:withdraw>
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>netent_test</playerName>
         <amount>10</amount>
         <bonusBet>0</bonusBet>
         <currency>EUR</currency>
         <transactionRef>10325</transactionRef>
         <gameRoundRef>8322</gameRoundRef>
         <gameId>blackjack2-3h_sw</gameId>
         <reason>GAME_PLAY</reason>
         <sessionId>1357498797399-339-P0FSA2J3QY1GJ</sessionId>
      </ns2:withdraw>
   </S:Body>
</S:Envelope>
```

Response:

Example:

```
S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:withdrawResponse xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
         <balance>50</balance>
         <transactionId>2065955413</transactionId>
      </ns2:withdrawResponse>
   </S:Body>
</S:Envelope>
```

deposit (#2)

This deposit call makes the bet refund payment to the player's account. Note that the gameroundRef refers to the game round that is being refunded. The source parameter holds a comment about the reason/context for this event.

Request:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
   <S:Body>
      <ns2:deposit>
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>netent_test</playerName>
         <amount>10</amount>
         <jackpotAmount></jackpotAmount>
         <bonusWin></bonusWin>
         <currency>EUR</currency>
         <transactionRef>10328</transactionRef>
         <gameRoundRef>8322</gameRoundRef>
         <gameId>blackjack2-3h_sw</gameId>
         <reason>CLEAR_HANGED_GAME_STATE</reason>
         <sessionId>1357498797399-339-P0FSA2J3QY1GJ</sessionId>
         <tournamentOccurrenceId></tournamentOccurrenceId>
         <bonusprograms><!--Zero or more repetitions:-->
            <bonus>
               <bonusprogramid></bonusprogramid>
               <depositionid></depositionid>
            </bonus>
         </bonusprograms>
         <source>Manually cleared game</source>
         <startDate>2013-01-08 12:27:33.000</startDate>
      </ns2:deposit>
   </S:Body>
</S:Envelope>
```

Response:

Example:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:depositResponse xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
         <balance>60</balance>
         <transactionId>2065958744</transactionId>
      </ns2:depositResponse>
   </S:Body>
</S:Envelope>
```

Rollback and Retry of Transactions

If there are connection problems between CasinoModule and the wallet server, or if there is an internal processing error that results in an invalid response from the wallet server, the affected transaction must be resolved.

Important:

Wallet server responses that contain a valid Fault object (that is, withdrawFault) are considered to indicate a *handled* error. The player experience will follow the description below, but the transaction will not be resolved. If in doubt, and if the transaction needs to be resolved, always return a server error, and not a valid XML response.

This is done asynchronously from the player experience after a certain delay, normally within 10 minutes.

The player experience may vary depending on how the service has been configured:

- Issues with `withdraw` transactions will always result in interrupted game play. An error message is displayed in the game, and the game must be restarted to continue playing. The affected transaction is placed in the rollback queue.
- Issues with `deposit` transactions can either result in interrupted game play, as per above, or without any visible impact to the player. This depends on if the service has been configured to allow rollbacks of deposit transactions. If rollbacks are allowed, issues result in interrupted game play and the affected transaction is placed in the rollback queue. If rollbacks are not allowed, game play will continue and the affected transaction is placed in the retry queue.
- Issues with `withdrawAndDeposit` transactions will always result in interrupted game play, and the affected transaction is placed in the rollback queue. WithdrawAndDeposit transactions are only used if rollbacks of deposits are allowed.

Any transaction that is placed in the rollback queue is also removed from the CasinoModule database, thereby not affecting reporting. This is done using database-level transaction handling, and only a limited set of information is kept for transactions (and subsequently game rounds) that are removed.

When processing a rollback, the wallet server must debit/credit the player’s account accordingly:

- Rollbacks of `deposit` transactions should result in a deposit of funds to the player’s account.
- Rollbacks of `withdraw` transactions should result in withdrawal of funds from the player’s account.
- Rollbacks of `withdrawAndDeposit` transactions should revert both underlying transactions.
- Retries of `deposit` transactions should result in a deposit of funds to the player’s account.

Note:

Rollback requests and retry requests are idempotent, that is, will not change the initial call result for the method, and may be issued to the wallet server several times until a proper response is received. Each attempt will use the same transactionRef as the original transaction. If the original transaction does not exist on the wallet server side, the wallet server should send a valid response – not an error – as this is the only way for CasinoModule to mark the transaction as resolved.

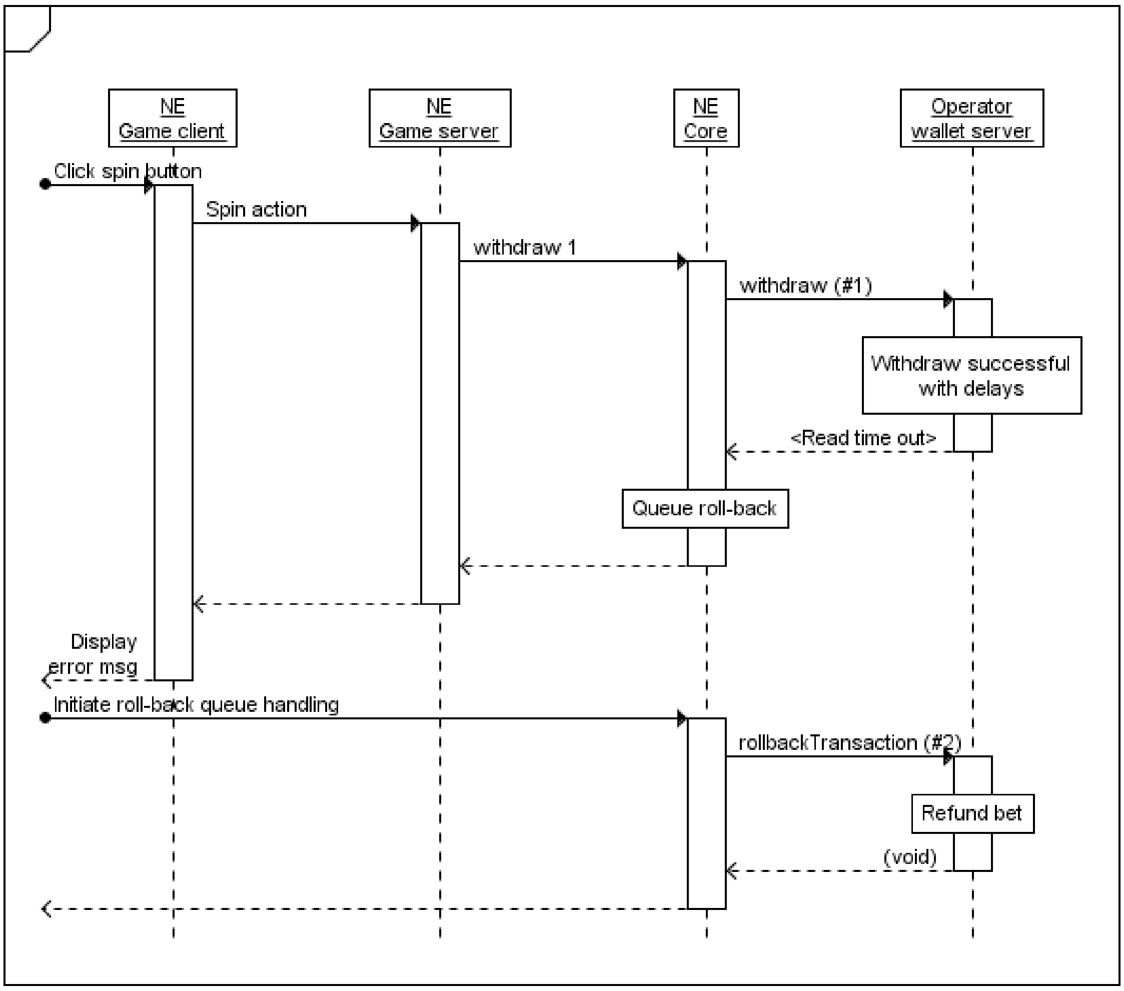
Read more about rollback transactions in the following examples:

- Rollback Example 1: Unhandled Error on Withdraw
- Rollback Example 2: Insufficient Funds on Withdraw
- Rollback Example 3: Internal Error on Withdraw
- Rollback Example 4: Deposit Errors
- Rollback Example 5: Withdraw and Deposit

Rollback Example 1: Unhandled Error on Withdraw

This example shows the rollback handling if the initial withdraw request times out, or if an unexpected response is returned by the wallet server.

The following diagram shows the example flow:

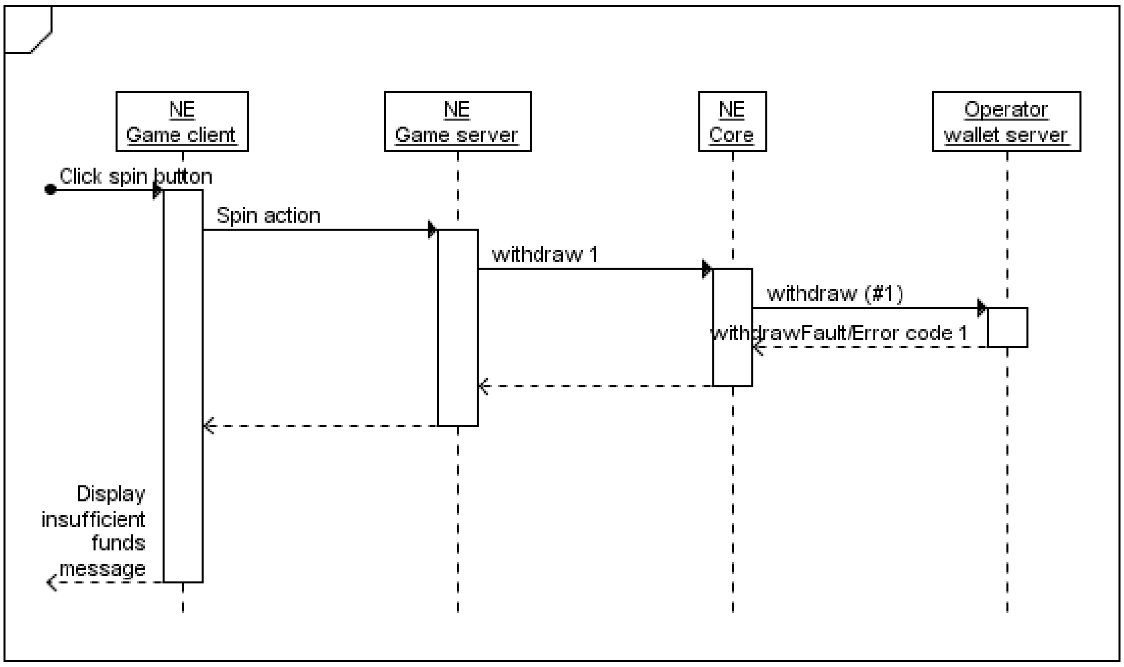


Rollback example 1: Unhandled error on withdraw

Rollback Example 2: Insufficient Funds on Withdraw

This example shows handling of errors returned by the wallet server on an initial withdraw, because of insufficient funds. In this scenario, the withdraw transaction is *not* resolved.

The following diagram shows the example flow:



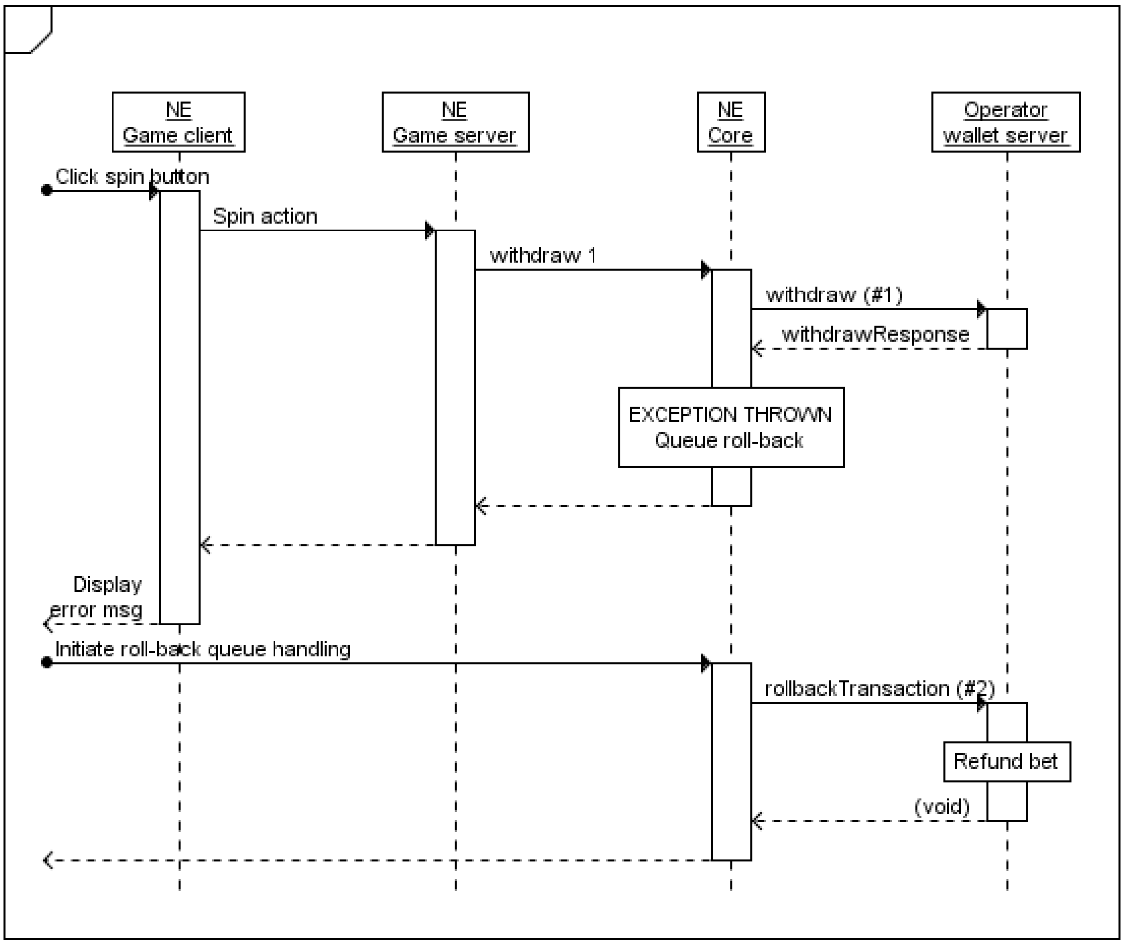
Rollback example 2: Insufficient funds on withdraw

Rollback Example 3: Internal Error on Withdraw

In this example, the withdraw call to the operator's wallet server is successful, but an error occurs in CasinoModule afterwards, before the transaction can be committed and before the outcome is displayed to the player.

Since the withdrawal has been successfully committed on the wallet server side (that is, the `transactionRef` is recognized by the operator), the bet must be refunded to the player's account.

The following diagram shows the example flow:

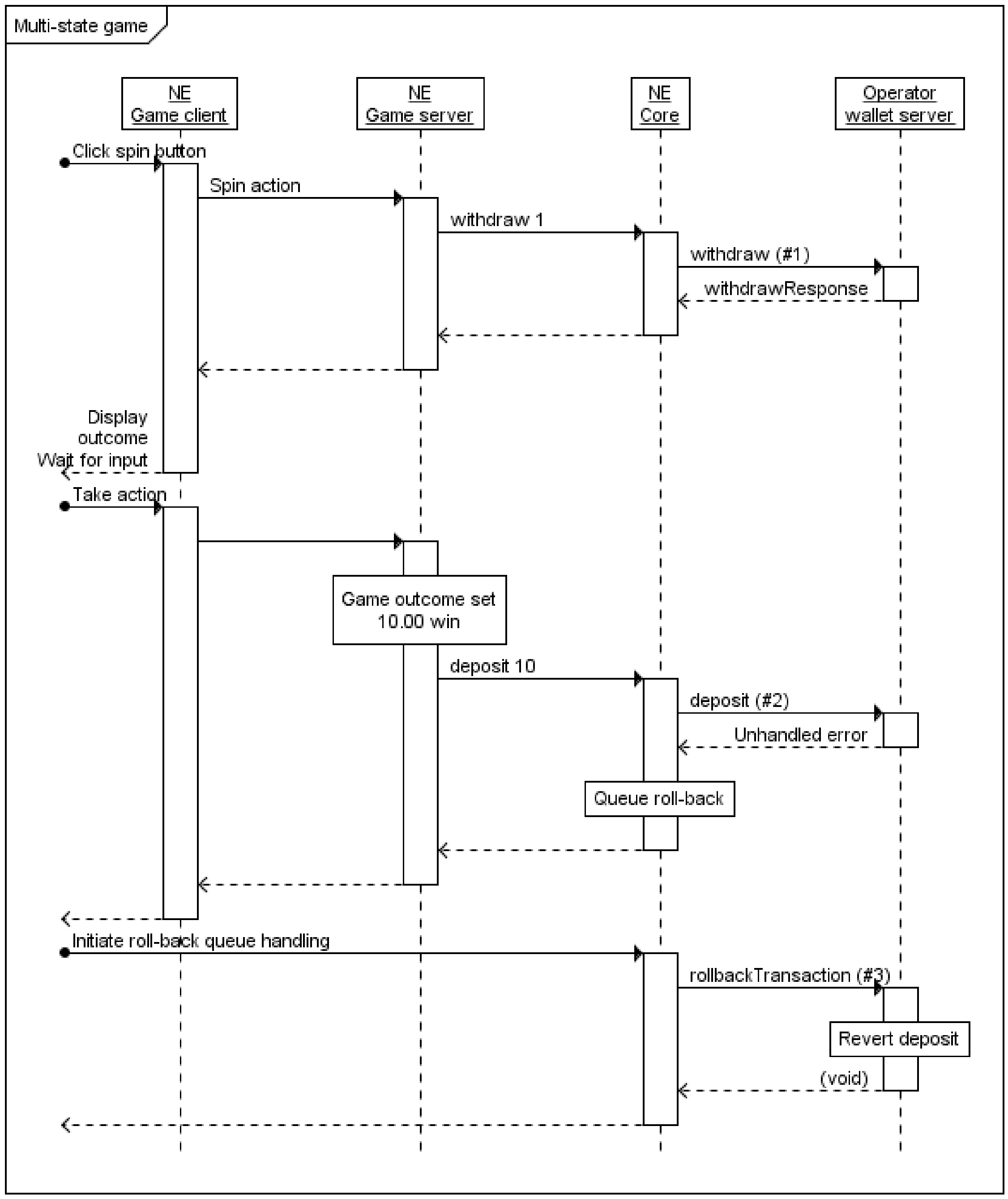


Rollback example 3: Internal error on withdraw

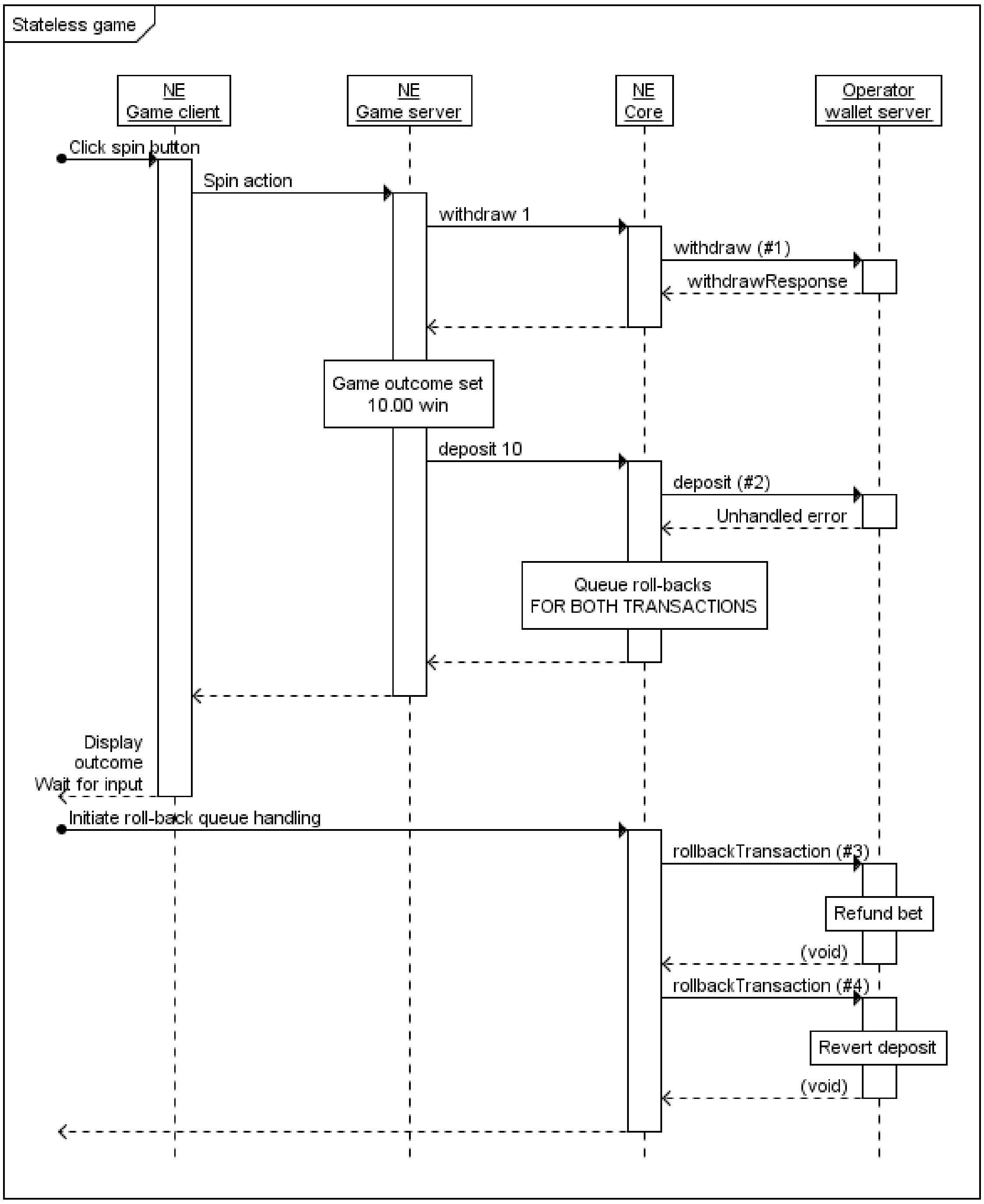
Rollback Example 4: Deposit Errors

The following examples show deposit calls that mark the end of a game round, but fail due to unexpected errors. In these scenarios, the wallet server supports rollbacks of deposits.

The following two examples show the difference between multi-state games (such as Black-jack or video slot games with bonus modes) and stateless games (typically video slot games).



Rollback Example 4a: Deposit errors in a multi-state game

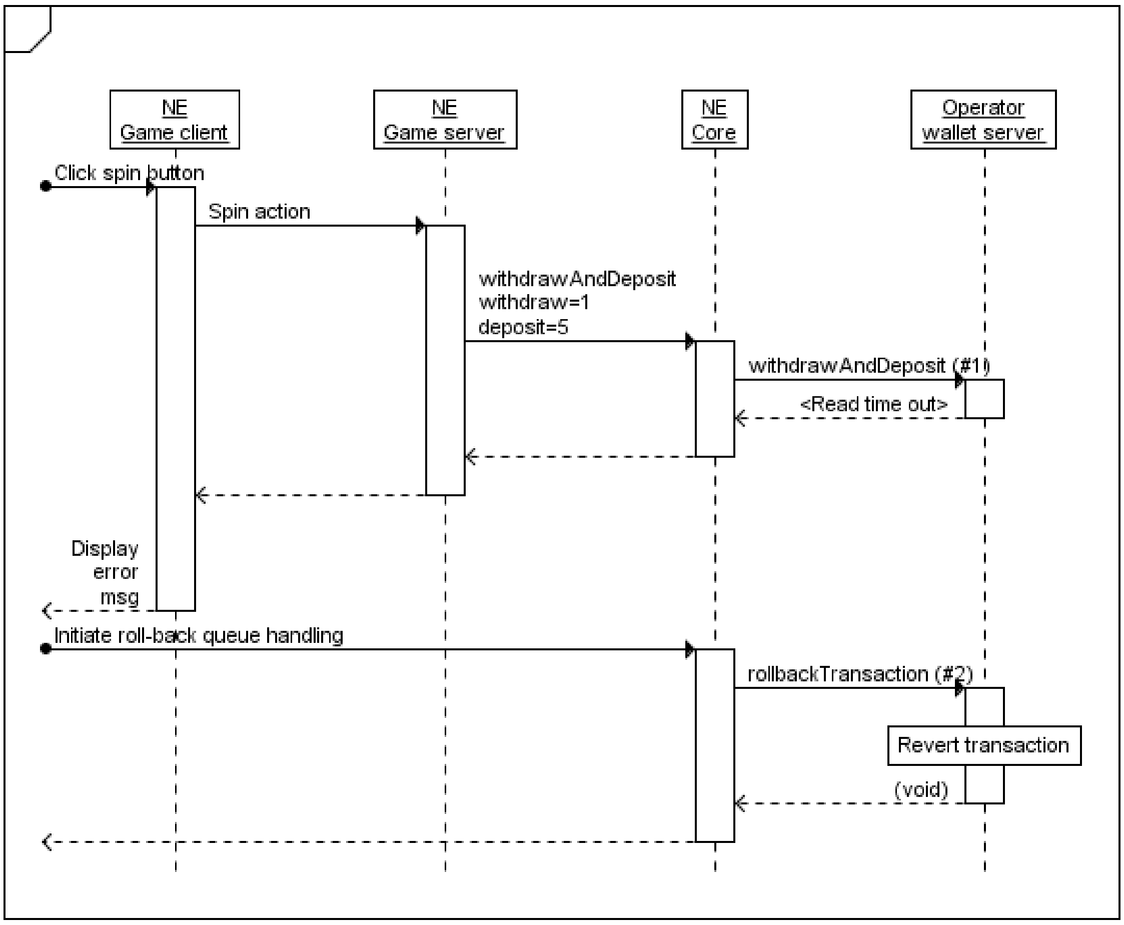


Rollback example 4b: Deposit errors in a stateless game

Rollback Example 5: Withdraw and Deposit

In this example, there is no response from the wallet server on the `withdrawAndDeposit` request.

The following diagram shows the example flow:



Rollback example 5: Withdraw and deposit

Tournament Win

Tournament winnings are paid either as bonus money or as real money, depending on how the tournament has been configured.

Bonus money winnings will end up in the player's bonus wallet, and will not trigger any calls to the wallet server.

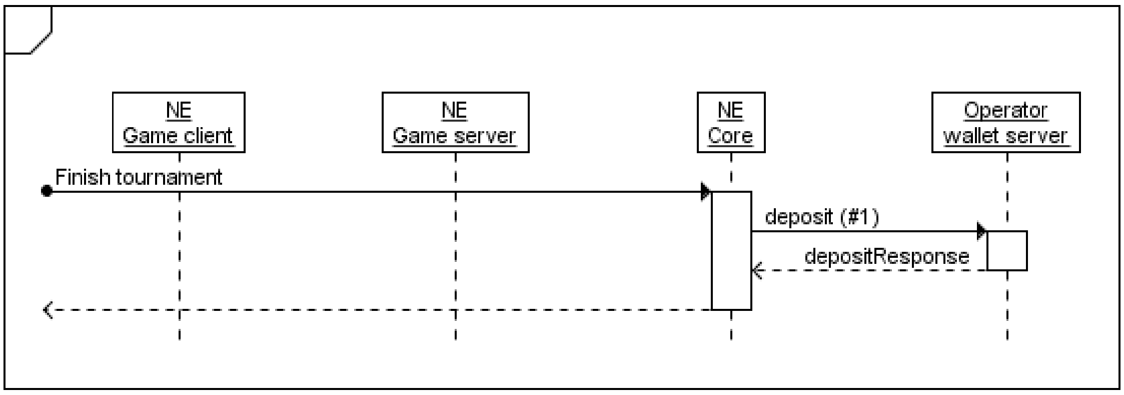
Real money winnings are paid with deposit calls using reason `AWARD_TOURNAMENT_WIN`.

Tournaments are finished (and payments of winnings are made) asynchronously from game play by a scheduled job as soon as the tournament end time is reached. This can occur after the player has logged out and left the operator's website, so the wallet server must allow this deposit to be processed even though the player is not logged in.

Note:

The `gameId` and `gameRoundRef` parameters are null/blank for payment of tournament winnings.

The following diagram shows the example flow:



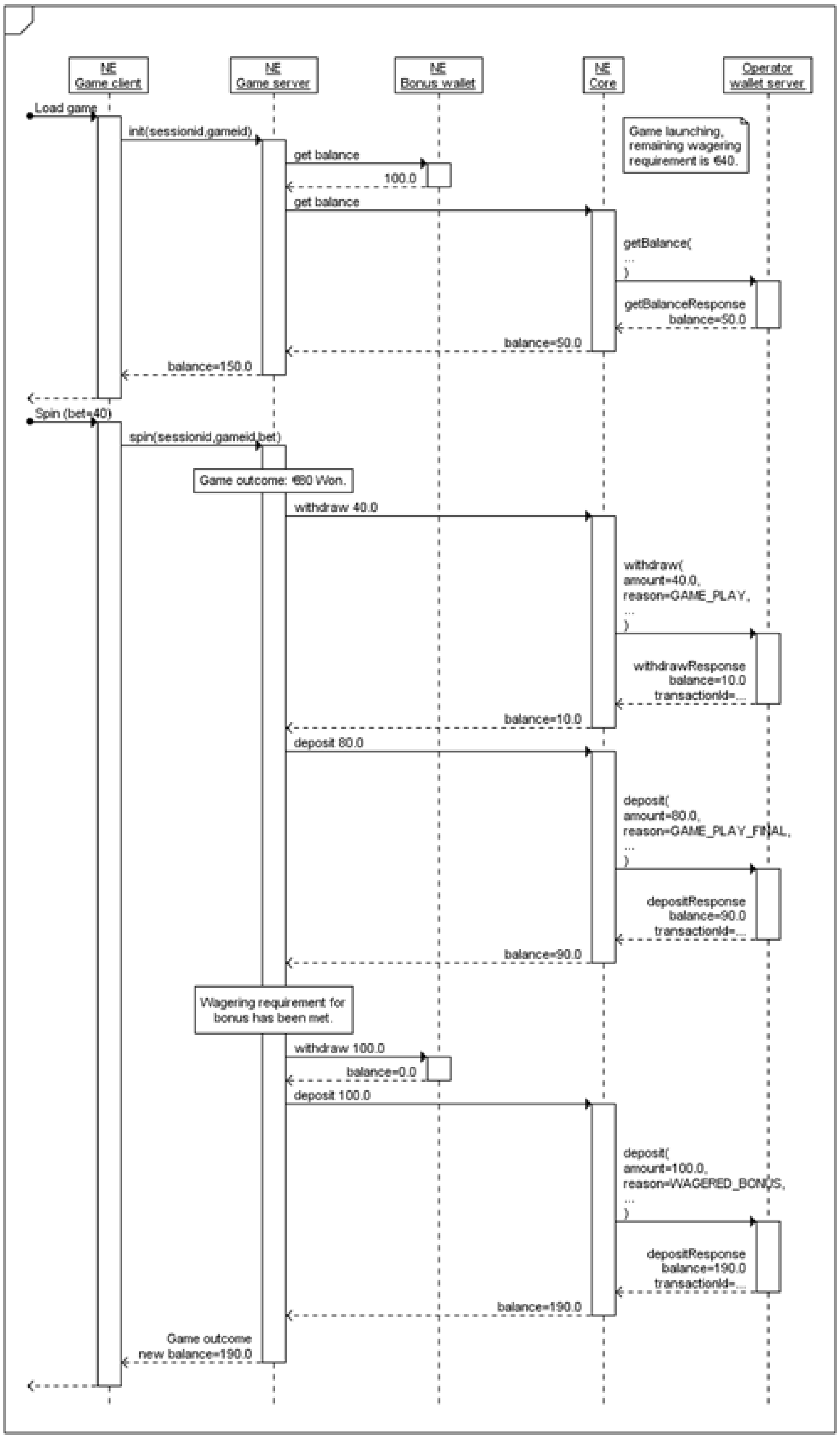
Tournament win

Deposit of Wagered Bonus

Bonus money that has been awarded to a player is kept in a separate bonus wallet in CasinoModule. If the player manages to meet the wagering requirement associated with the bonus, the remaining bonus money will be paid to the wallet server using a deposit request.

The deposit has its reason parameter set to `WAGERED_BONUS`, and will normally be made as part of a game play event – that last bet which fills up the wagering requirement.

The following diagram shows the example flow:


NE Game client
NE Game server
NE Bonus wallet
NE Core
Operator wallet server
Load game
init(sessionid,gameid)
Game launching, remaining wagering requirement is €40.
get balance
100.0
get balance
getBalance(...)
getBalanceResponse balance=50.0
balance=50.0
balance=150.0
Spin (bet=40)
spin(sessionid,gameid,bet)
Game outcome: €80 Won.
withdraw 40.0
withdraw(amount=40.0, reason=GAME_PLAY, ...)
withdrawResponse balance=10.0 transactionId=...
balance=10.0
deposit 80.0
deposit(amount=80.0, reason=GAME_PLAY_FINAL, ...)
depositResponse balance=90.0 transactionId=...
balance=90.0
Wagering requirement for bonus has been met.
withdraw 100.0
balance=0.0
deposit 100.0
deposit(amount=100.0, reason=WAGERED_BONUS, ...)
depositResponse balance=190.0 transactionId=...
balance=190.0
Game outcome new balance=190.0

Deposit of wagered bonus

Withdraw and Deposit

The `withdrawAndDeposit` method is used to reduce the number of calls made to the wallet server by combining withdraw and deposit transactions into a single wallet server request. If enabled, most game play transactions will use this method, while tournament and bonus related transactions will be made using the deposit method.

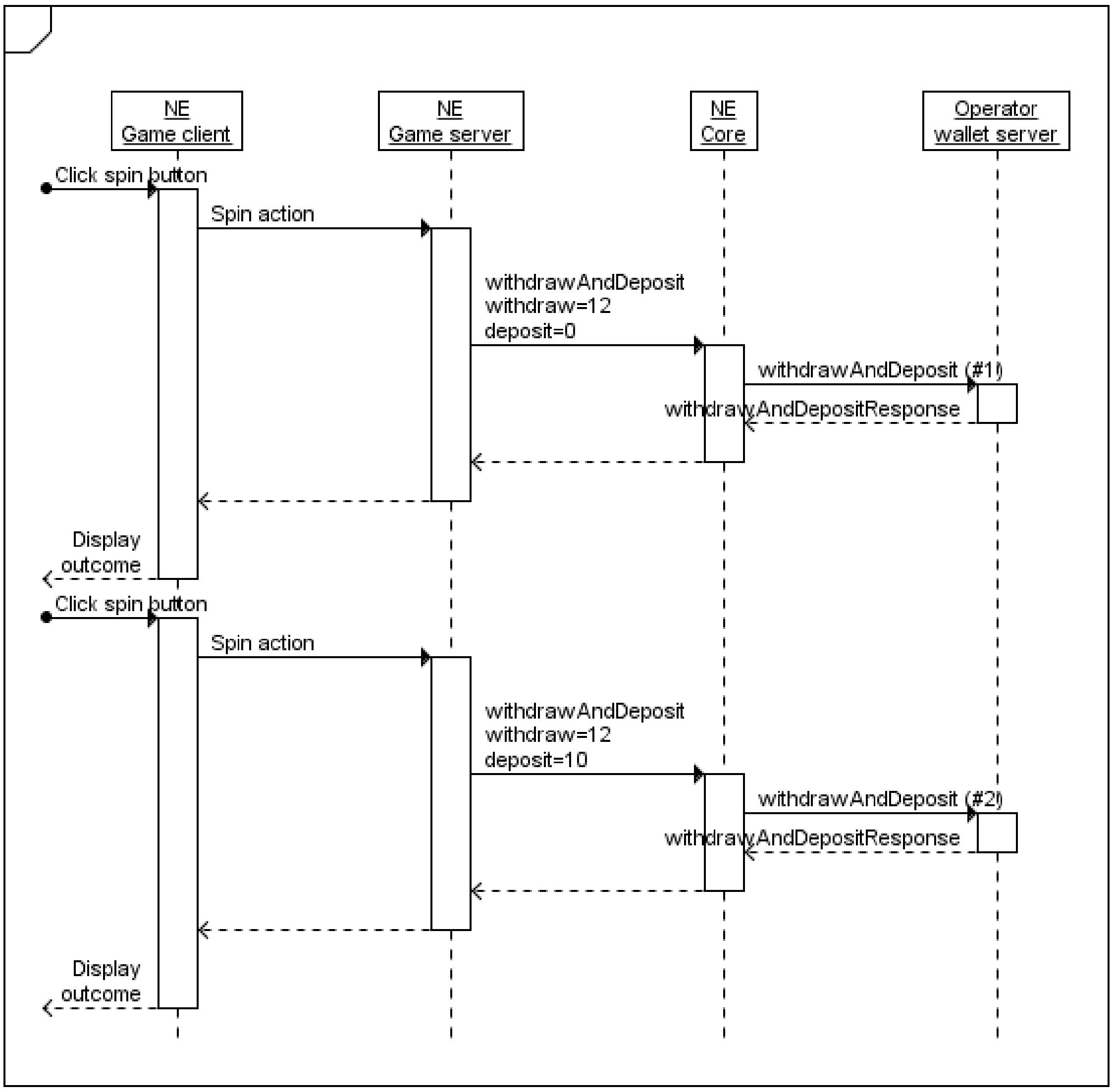
Note:

withdrawAndDeposit cannot be used if the wallet server does not allow rollback of deposits.

For stateful games such as Blackjack, there will be `withdrawAndDeposit` calls with withdraw=0, deposit=0, to mark the end of a game round with no win.

Some games, such as Roulette, allow zero bet spins stored and propagated to the wallet server. These spins will also trigger a `withdrawAndDeposit` call with withdraw=0, deposit=0.

The following diagram shows the example flow:



Withdraw and deposit

Send a Customized Message

The examples in this section describe how an operator can send error or information messages with customized content. The customized messages will be displayed directly in the player's game client.

Note:

Not all games are supported to utilise this feature, and some games support one method but not the others. For specific information about which games that support customized messages, please contact support@netent.com.

Depending on the desired game client behaviour, four different types of customized message boxes can be displayed in the game client during game play. Each type of message has its own code parameter (990, 991, 992 and 993). For detailed information about the different message types, please refer to section Customized Messages in this guide.

The customized message is sent utilising the `withdraw`, `deposit` or `withdrawAndDeposit` API method to call the operator's wallet server. The message can then be incorporated in the response call from the operator's wallet server.

The Methods section below describes how to send a message using the `withdraw` method.

Methods

withdraw (Response Variant #1)

In this example an ordinary withdraw call is made to the operator's wallet server. In cases when the response from the wallet server is executed without any errors and the reason type for the withdraw call is `GAME_PLAY` or `GAME_PLAY_FINAL` then it is possible for the operator to incorporate a customized message that will be sent back to the game client and displayed to the player. Only messages that use the code 992 or 993 can be sent during these conditions.

Request:

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
"xmlns:_0="http://types.walletserver.casinomodule.com/3_0/">
   <soapenv:Header/>
      <soapenv:Body>
         <_0:withdraw>
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>test</playerName>
         <amount>10</amount>
         <currency>EUR</currency>
         <transactionRef>123</transactionRef>
         <reason>GAME_PLAY_FINAL</reason>
      </_0:withdraw>
   </soapenv:Body>
</soapenv:Envelope>
```

Response:

Example:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:withdrawResponse xmlns:ns2="http://types.walletserver.casinomodule.com/3_0/">
         <balance>4987.0</balance>
         <transactionId>6</transactionId>
            <message>
               <code>992</code>
               <message>testmessage</message>
         </message>
      </ns2:withdrawResponse>
   </soap:Body>
</soap:Envelope>
```

withdraw (Response Variant #2)

In this example an ordinary withdraw call is made to the operator's wallet server, but in this case the transaction fails and the wallet server responds with an error code. The error code could be any of the standard error codes, which generates a standard error message. For example "Not Enough Money", "Player Limit Exceeded", etc.

In this situation it would be possible for the operator to incorporate a customized error message that will be sent back to the game client and displayed to the player. Only messages that use the code 990 or 991 can be sent during these conditions.

Request:

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
"xmlns:_0="http://types.walletserver.casinomodule.com/3_0/">
   <soapenv:Body>
      <_0:withdraw>
         <callerId>testcaller</callerId>
         <callerPassword>testpassword</callerPassword>
         <playerName>netent_test</playerName>
         <amount>10</amount>
         <currency>EUR</currency>
         <transactionRef>123</transactionRef>
         <reason>GAME_PLAY_FINAL</reason>
      </_0:withdraw>
   </soapenv:Body>
</soapenv:Envelope>
```

Response:

Example:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>testError</faultstring>
            <detail>
               <ns2:withdrawFault xmlns:n-
               s2="http://types.walletserver.casinomodule.com/3_0/">
               <errorCode>990</errorCode>
               <message>testmessage</message>
               </ns2:withdrawFault>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```