Mascot Gaming API Documentation

v. 2.4.1

April 30, 2021

Contents

Changelog

Date	Version	Change description
30.01.2020	2.0.0	1st Draft
13.02.2020	2.1.1	Added free rounds parameters to Seamless V2 API. Modified Bonus.Set method. Added `sessionAlternativeId` parameter to Seamless V2 methods (deposit, withdraw, getBalance, withdrawAndDeposit)
14.02.2020	2.1.1	Added request and response examples for Seamless V2 API methods
28.02.2020	2.1.2	Added error code `CURRENCY_NOT_FOUND` to the list of error codes. Fixed a mistake in `Start free rounds (bonus) game`
17.03.2020	2.1.3	Added new parameter *seriesId* for *Session.Create*, *getBalance*, *withdrawAndDeposit*
02.04.2020	2.1.4	Added principles of work of the RestorePolicy parameter
06.04.2020	2.1.5	Updated information about HTTP-headers
21.04.2020	2.1.6	Minor improvements
24.04.2020	2.1.7	Removed Wallet API description. Added description of the *reason* field in the *Seamless V2 API*. Moved the section *Seamless reference* to the *How it works under the hood* section Added note about the *RestorePolicy* parameter
27.04.2020	2.2.0	Updated section about errors and possible errors. Updated description of *AlternativeId* parameter in *Session.Create* method
19.05.2020	2.3.0	Documentation re-design
25.06.2020	2.3.1	Minor improvements
07.07.2020	2.3.2	Updated description of *rollbackTransaction* method
11.08.2020	2.3.3	Updated UML-diagrams
10.11.2020	2.3.4	Added *Params* object description in *Session.CreateDemo* method description
01.12.2020	2.3.5	Updated list of currencies
04.02.2021	2.4.0	Added possible causes and solutions for error with 495 code Removed *withdraw* and *deposit* methods Other minor improvements
02.04.2021	2.4.1	Updated available languages to set for *Session.create* method

General information

All APIs are based on the JSON-RPC 2.0 protocol.

And it works over HTTPS, so you will find this documentation useful as well: JSON-RPC 2.0 over HTTPS.

> **Attention:** The HTTP-headers have to be strictly the same as these:
>
> `Content-Type: application/json`
>
> `Content-Length`: must contain the correct length according to the HTTP-specification.
>
> `Accept: application/json`

2.1 API types

Operator API - basic API methods for creating players/bank groups/bonuses/sessions.

There is one method type to store player's balance:

Seamless V2 API - methods for store player's balance on game provider side.

2.2 API endpoint

The *game provider* endpoint for *Operator API* is: https://api.mascot.games/v1/.

It expects to receive calls using JSON-RPC 2.0 over HTTPS.

2.3 Basic terminology

Before further reading, we strongly recommend you to get familiar with the basic terminology in the *vocabulary* for better understanding.

Getting Started

This section provides information about how to integrate your casino with Mascot Gaming API in several steps.

3.1 Preparations

3.1.1 Get client certificate

This certificate needs for verifying and identifying you when you send requests to the *game provider*.

To get the client certificate, you need to use instruction below.

> **Attention:** Game provider uses `PEM` format.
>
> For example, if you use `PKCS` standard, you might use this command to convert `PKCS` to `PEM` format:
>
> ```
> openssl pkcs12 -in client.p12 -out client.pem
> ```

Before you start using the API you need to generate a certificate to be able to use it as an authorized user.

You can do it with any utility you want but here is an example of using `openssl` in Linux:

Step 1

Create a folder ('ssl' or any other name you want, it doesn't matter at all), where you are going to generate a private key and a certificate signing request file. And just go there.

```
mkdir ssl
cd ssl
```

Step 2

Generate a key and csr files:

```
openssl req -out client.csr -new -newkey rsa:2048 -nodes -keyout client.key
```

Attention: You should pre-install the openssl component before you do this.

```
Generating a 2048 bit RSA private key
.....................................................+++
.....................................................+++
writing new private key to 'client.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

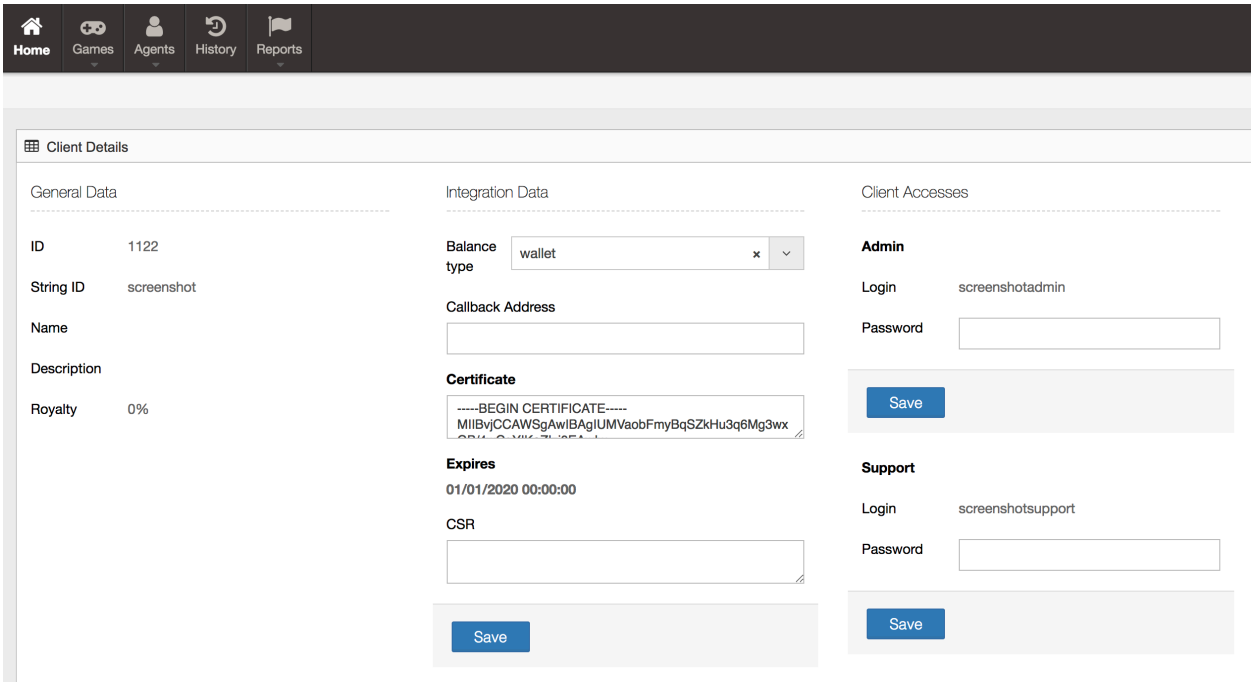
Note: You can fill all the fields out with whatever you want or just leave it blank.

After executing this command, you will see new files in that folder.

```
ls -la

total 16
drwxrwxr-x 2 hidden hidden 4096 Jul 25 15:10 .
drwxrwxr-x 7 hidden hidden 4096 Jul 25 15:10 ..
-rw-rw-r-- 1 hidden hidden  956 Jul 25 15:10 client.csr
-rw-rw-r-- 1 hidden hidden 1704 Jul 25 15:10 client.key
```

Put the content of `client.csr` into the 'CSR' text area (you can find it on the home page of your back office account) and click 'Save' button. You will get a signed API certificate in the 'Certificate' text area.



Step 3

Now just copy the content of the 'Certificate' text area and put it to `client.crt` file. That's your certificate for our API.

> **Attention:** The certificate is issued only for a year — after it needs to be re-issued.

Note: **Step 4** is required for customers who use PHP API Client.

Step 4 (Optional)

And finally, you need to join (concatenate) `client.key` and `client.crt` and save it to a new file named `apikey.pem`. Because our API Client requires to do that.

On Linux you can do it like that:

```
cat client.key client.crt > apikey.pem
```

That's all, you made your certificate for our API!

As a result, you are supposed to have these files:

```
ls -la

total 16
drwxrwxr-x 2 hidden hidden 4096 Jul 25 15:10 .
drwxrwxr-x 7 hidden hidden 4096 Jul 25 15:10 ..
```

(continues on next page)

(continued from previous page)

```
-rw-rw-r-- 1 hidden hidden 3383 Jun 25 15:11 apikey.pem
-rw-rw-r-- 1 hidden hidden 1680 Jun 25 15:11 client.crt
-rw-rw-r-- 1 hidden hidden  956 Jul 25 15:10 client.csr
-rw-rw-r-- 1 hidden hidden 1704 Jul 25 15:10 client.key
```

3.1.2 Install API (JSON-RPC) Client (Optional)

First of all, you need to install Composer (Dependency Manager for PHP) if you haven't done it before.

Here you can find how to do this.

Then, go to the root of your project path, create a file, name it **composer.json** and put this content there:

```
{
    "minimum-stability": "dev",
    "repositories": [
        {
            "type": "vcs",
            "url": "https://github.com/MascotGaming/mascot-api-client.git"
        }
    ]
}
```

Then run the installation of our api package:

```
composer require mascotgaming/mascot-api-client
```

Now you got it!

And don't forget to add the autoloading of all installed composer packages into your project.

Read more about that here .

For example, you can create **test.php** file to check whether it works or not.

```
<?php

use mascotgaming\mascot\api\client\Client;

require __DIR__.'/vendor/autoload.php';

$client = new Client(array(
        'url' => 'https://api.mascot.games/v1/',
        'sslKeyPath' => __DIR__.'/ssl/apikey.pem',
));

var_export($client->listGames());
```

3.2 Start demo game

Just do the following **requests** by your server.

Warning: Do not forget to do *preparation steps*

Note: For a more quick start, you can use our PHP API Client.

3.2.1 Get games

Request:

```
{
    "jsonrpc": "2.0",
    "method": "Game.List",
    "id": 1920911592
}
```

Expected response:

```
{
    "jsonrpc": "2.0",
    "id": 1920911592,
    "result": {
        "Games": [
            {
                "Id": "bennys_the_biggest_game",
                "Name": "Benny's the Biggest game",
                "Description": "20 Lines, Free, Bonus",
                "SectionId": "mascot",
                "Format": "html",
                "Type": "slots"
            },
            { "..." }
        ]
    }
}
```

Note: Step **2** is intended for operators who are using the API for the first time. Therefore, there is no need to re-create the bank group if you have already done so.

3.2.2 Set bank group

Request:

```
{
    "jsonrpc": "2.0",
    "method": "BankGroup.Set",
    "id": 1225625456,
    "params": {
        "Id": "new_bank_group",
        "Currency": "EUR"
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Expected response:

```
{
    "jsonrpc": "2.0",
    "id": 1225625456,
    "result": []
}
```

3.2.3 Start demo session

Request:

```
{
    "jsonrpc": "2.0",
    "method": "Session.CreateDemo",
    "id": 321864203,
    "params": {
        "BankGroupId": "new_bank_group",
        "GameId": "bennys_the_biggest_game",
        "StartBalance": 10000
    }
}
```

Expected response:

```
{
    "jsonrpc": "2.0",
    "id": 321864203,
    "result": {
        "SessionId": "xxxxxxxxxx",
        "SessionUrl": "https://xxxxxxxxxx.domain.com/"
    }
}
```

3.2.4 Open session URL from response in browser tab

In the last step you need to open `SessionUrl` in a new browser tab.

3.3 Start standard game

Just do the following **requests** by your server.

> **Warning:** Do not forget to do *preparation steps*

Note: For a more quick start, you can use our PHP API Client.

3.3.1 Get games

Request:

```
{
    "jsonrpc": "2.0",
    "method": "Game.List",
    "id": 1920911592
}
```

Expected response:

```
{
    "jsonrpc": "2.0",
    "id": 1920911592,
    "result": {
        "Games": [
            {
                "Id": "bennys_the_biggest_game",
                "Name": "Benny's the Biggest game",
                "Description": "20 Lines, Free, Bonus",
                "SectionId": "mascot",
                "Format": "html",
                "Type": "slots"
            },
            { "..." }
        ]
    }
}
```

Note: Steps **2, 3** are intended for operators who are using the API for the first time. Therefore, there is no need to re-create the bank group/player or set the initial balance if you have already done so.

3.3.2 Set bank group

Request:

```
{
    "jsonrpc": "2.0",
    "method": "BankGroup.Set",
    "id": 1225625456,
    "params": {
        "Id": "new_bank_group",
        "Currency": "EUR"
    }
}
```

Expected response:

```
{
    "jsonrpc": "2.0",
    "id": 1225625456,
    "result": []
}
```

3.3.3 Set player

Request:

```
{
    "jsonrpc": "2.0",
    "method": "Player.Set",
    "id": 1928822491,
    "params": {
        "Id": "noname",
        "Nick": "Noname",
        "BankGroupId": "new_bank_group"
    }
}
```

Expected response:

```
{
    "jsonrpc": "2.0",
    "id": 1928822491,
    "result": []
}
```

3.3.4 Start session

Request:

```
{
    "jsonrpc": "2.0",
    "method": "Session.Create",
    "id": 321864203,
    "params": {
        "PlayerId": "noname",
        "GameId": "bennys_the_biggest_game"
    }
}
```

Expected response:

```
{
   "jsonrpc": "2.0",
   "id": 321864203,
   "result": {
      "SessionId": "xxxxxxxxxx",
      "SessionUrl": "https://xxxxxxxxxx.domain.com/"
   }
}
```

3.3.5 Open session URL from response in browser tab

In the last step you need to open `SessionUrl` in a new browser tab.

3.4 Start free rounds (bonus) game

Just do the following **requests** by your server.

> **Warning:** Do not forget to do *preparation steps*

Note: For a more quick start, you can use our PHP API Client.

3.4.1 Get games

Request:

```
{
    "jsonrpc": "2.0",
    "method": "Game.List",
    "id": 1920911592
}
```

Expected response:

```
{
    "jsonrpc": "2.0",
    "id": 1920911592,
    "result": {
        "Games": [
            {
                "Id": "bennys_the_biggest_game",
                "Name": "Benny's the Biggest game",
                "Description": "20 Lines, Free, Bonus",
                "SectionId": "mascot",
                "Format": "html",
                "Type": "slots"
            },
            { "..." }
        ]
    }
}
```

Note: Steps **2, 3, 4** are intended for operators who are using the API for the first time. Therefore, there is no need to re-create the bank group/player or set bonus if you have already done so.

3.4.2 Set bank group

Request:

```
{
    "jsonrpc": "2.0",
    "method": "BankGroup.Set",
    "id": 1225625456,
    "params": {
        "Id": "new_bank_group",
        "Currency": "EUR"
    }
}
```

Expected response:

```
{
    "jsonrpc": "2.0",
    "id": 1225625456,
    "result": []
}
```

3.4.3 Set player

Request:

```
{
    "jsonrpc": "2.0",
    "method": "Player.Set",
    "id": 1928822491,
    "params": {
        "Id": "noname",
        "Nick": "Noname",
        "BankGroupId": "new_bank_group"
    }
}
```

Expected response:

```
{
    "jsonrpc": "2.0",
    "id": 1928822491,
    "result": []
}
```

3.4.4 Set bonus

Request:

```
{
    "jsonrpc": "2.0",
    "method": "Bonus.Set",
    "id": 1928822492,
```

(continues on next page)

(continued from previous page)

```
    "params": {
        "Id": "new_bonus"
    }
}
```

Expected response:

```
{
   "jsonrpc": "2.0",
   "id": 1047919053,
   "result": {
      "Bonus": {
         "Id": "new_bonus",
         "FsType": "original",
         "CounterType": "separate"
      }
   }
}
```

3.4.5 Start bonus session

Request:

```
{
    "jsonrpc": "2.0",
    "method": "Session.Create",
    "id": 321864203,
    "params": {
        "PlayerId": "noname",
        "GameId": "bennys_the_biggest_game",
        "BonusId": "new_bonus"
    }
}
```

Expected response:

```
{
   "jsonrpc": "2.0",
   "id": 321864203,
   "result": {
      "SessionId": "xxxxxxxxxx",
      "SessionUrl": "https://xxxxxxxxxx.domain.com/"
   }
}
```

3.4.6 Open bonus session URL from response in browser tab

In the last step you need to open `SessionUrl` in a new browser tab.

API Methods

This section contains a detailed description of all API methods.

4.1 Operator API

4.1.1 Game.List

Type: Request

Returns a list of available games.

This method has no request parameters.

Response parameters:

Key	Type	Description
`Games`	Array of objects	Each object is a **Game object**

Game object struct:

Key	Type	Description
`Id`	String	Unique game identifier
`Name`	String	Game name
`Description`	String	Game short description
`SectionId`	String	Game section identifier

Request example:

```
{
    "jsonrpc": "2.0",
    "method": "Game.List",
    "id": 1920911592
}
```

Response example:

```
{
    "jsonrpc": "2.0",
    "id": 1920911592,
```

(continues on next page)

(continued from previous page)

```
    "result": {
        "Games": [
            {
                "Id": "bennys_the_biggest_game",
                "Name": "Benny's the Biggest game",
                "Description": "20 Lines, Free, Bonus",
                "SectionId": "mascot",
                "Format": "html",
                "Type": "slots"
            },
            { "..." }
        ]
    }
}
```

4.1.2 BankGroup.Set

Type: Request

Creates or updates a bank group (aka "upsert").

> **Warning:** You can set a currency for the bank group only when you create the bank group.

Request parameters:

Key	Type	Required?	Description
Id	String	Yes	Unique bank group identifier
Currency	String	Yes	Bank group *currency code*. This field is ignored on update since currency cannot be changed
SettingsPatch	String	No	Apply a specified patch to the bank group

This method has no response parameters.

Request example:

```
{
    "jsonrpc": "2.0",
    "method": "BankGroup.Set",
    "id": 1225625456,
    "params": {
        "Id": "new_bank_group",
        "Currency": "EUR"
    }
}
```

Response example:

```
{
    "jsonrpc": "2.0",
    "id": 1225625456,
    "result": []
}
```

4.1.3 Player.Set

Type: Request

Creates or updates a player (aka "upsert").

Request parameters:

Key	Type	Required?	Description
`Id`	String	Yes	Unique player identifier
`Nick`	String	No	Player's nickname. Some of the games may display it. Not unique
`BankGroupId`	String	Yes	Bank group identifier. The bank group that the player is going to work with. This field is ignored on update

This method has no response parameters.

Request example:

```
{
    "jsonrpc": "2.0",
    "method": "Player.Set",
    "id": 1928822491,
    "params": {
        "Id": "noname",
        "Nick": "Noname",
        "BankGroupId": "new_bank_group"
    }
}
```

Response example:

```
{
    "jsonrpc": "2.0",
    "id": 1928822491,
    "result": []
}
```

4.1.4 Session.Create

Type: Request

Creates a game session.

Request parameters:

Key	Type	Required?	Description
`PlayerId`	String	Yes	Player identifier
`GameId`	String	Yes	Game identifier
`RestorePolicy`	String	No	Session creation behavior. See the table below
`StaticHost`	String	No	A domain name for getting static files from the server which you specify
`Params`	Object	No	Additional session params
`BonusId`	String	No	Bonus identifier
`AlternativeId`	String	No	An alternative session identifier. **Must be unique**

RestorePolicy options:

Value	Description
`Create`	Creates a brand-new session. **It is used by default if RestorePolicy is not specified**
`Restore`	Creates a new session and tries to restore it from the previous session by copying old data to the new session. If the previous session has not existed yet (the same player and game), then it uses `Create` case (creates a new session)
`Last`	Returns the last session (the same player and game) if it's still open. Otherwise, it uses the `Restore` case

Note: When a new session created all previous open sessions get closed (implicitly).

Note: If the *RestorePolicy* parameter is set to *Create*, the game provider generates a new *session series*, while if the parameter is set to *Restore* or *Last*, the game provider adds a new session to the latest *series* for combination of player + game.

Note: *Restore policies* **do not** work for **free round** sessions. **Only for standard sessions.**

Params options:

Key	Type	Description
`language`	String	Sets language to a game if the game supports it, otherwise `en` is applied. Available list: `ru`, `en`, `de`, `es`, `fr`, `it`, `pt`, `tr`, `cn`
`freeround_bet`	Integer	Fixed bet level during free rounds in coins. Example: `2`
`freeround_denomination`	Float	Fixed coin price during free rounds in currency. Example: `0.01`

Response parameters:

Key	Type	Description
`SessionId`	String	Unique game session identifier
`SessionUrl`	String	Absolute URL for the game session. That's the URL where the player can play the game. You must redirect the player to this link or open it into an iframe on your site. Also, you can open a window using this URL address. However, you have plenty of choices, and you are free to do it as you wish

Request example:

```
{
    "jsonrpc": "2.0",
```

(continues on next page)

(continued from previous page)

```
    "method": "Session.Create",
    "id": 1047919053,
    "params": {
        "PlayerId": "noname",
        "GameId": "bennys_the_biggest_game"
    }
}
```

Response example:

```
{
   "jsonrpc": "2.0",
   "id": 1047919053,
   "result": {
      "SessionId": "xxxxxxxxxx",
      "SessionUrl": "https://xxxxxxxxxx.domain.com/"
   }
}
```

4.1.5 Session.CreateDemo

Type: Request

Creates a demo game session.

Request parameters:

Key	Type	Required?	Description
`GameId`	String	Yes	Game identifier
`BankGroupId`	String	Yes	Bank group identifier
`StartBalance`	Integer	No	Start balance value (in cents). If not specified, then it is 1000 cents
`StaticHost`	String	No	Static host for local cache storing games.
`AlternativeId`	String	No	An alternative session identifier
`Params`	Object	No	Additional session params

Params options:

Key	Type	Description
`language`	String	Sets language to a game if the game supports it, otherwise `en` is applied. Available list: `en`, `fr`, `de`, `ru`, `cn`, `es`, `pl`, `it`

Response parameters:

Key	Type	Description
SessionId	String	Unique game session identifier
SessionUrl	String	Absolute URL for the game session. That's the place where the player can play. You must redirect him or her to this link or open it into an iframe on your site. Also, you can open a window using this URL address. However, you have plenty of choices, and you are free to do it as you wish

Request example:

```
{
    "jsonrpc": "2.0",
    "method": "Session.CreateDemo",
    "id": 321864203,
    "params": {
        "BankGroupId": "new_bank_group",
        "GameId": "bennys_the_biggest_game",
        "StartBalance": 10000
    }
}
```

Response example:

```
{
    "jsonrpc": "2.0",
    "id": 321864203,
    "result": {
        "SessionId": "xxxxxxxxxx",
        "SessionUrl": "https:\/\/xxxxxxxxxx.domain.com\/"
    }
}
```

4.1.6 Session.Close

Type: Request

Explicitly closes the opened session.

Note: To calculate the statistics in the *back-office*, you may need to close the session by this method invocation. This method should only be called if you want to check the statistics immediately. Otherwise, prefer the default way: each new session closes the previous session.

Request parameters:

Key	Type	Required?	Description
SessionId	String	Yes	Session identifier

This method has no response parameters.

Request example:

```
{
    "jsonrpc": "2.0",
    "method": "Session.Close",
    "id": 1362660429,
    "params": {
        "SessionId": "xxxxxxxxxx"
    }
}
```

Response example:

```
{
    "jsonrpc": "2.0",
    "id": 1362660429,
    "result": []
}
```

4.1.7 Bonus.Set

Type: Request

Sets up a bonus.

Request parameters:

Key	Type	Required?	Description
`Id`	String	Yes	Unique bonus identifier

Request example:

```
{
    "jsonrpc": "2.0",
    "method": "Bonus.Set",
    "id": 1928822492,
    "params": {
        "Id": "new_bonus"
    }
}
```

Response example:

```
{
    "jsonrpc": "2.0",
    "id": 1928822492,
    "result": {
        "Bonus": {
            "Id": "new_bonus"
        }
    }
}
```

4.2 Seamless V2 API

The server on the operator side should be ready to receive API calls from the *game provider*. JSON-RPC 2.0 over HTTPS is used to implement the server.

The game provider sends requests to the operator server and expects to receive a corresponding response. If the response is broken or failed for some reason, the system will keep trying to request it again with some delay.

To make it secure the Seamless API handler should expect using a client certificate to authorize requests. Here's the certificate authority to check it: seamless-ca.pem.

Here is the example of Nginx config for TLS termination of traffic and authorization requests:

```
server {
    listen 443 ssl;
    server_name "seamless.example.com";

    ssl_certificate YOUR_OWN_CERTIFICATE;
    ssl_certificate_key YOUR_OWN_CERTIFICATE_KEY;

    ssl_verify_client on;
    ssl_client_certificate /path/to/seamless-ca.pem;

    location / {
        proxy_pass http://YOUR_JSONRPC_SERVER;
        proxy_set_header    Host            $host;
        proxy_set_header    X-Real-IP       $remote_addr;
        proxy_set_header    X-Forwarded-for $remote_addr;
    }
}
```

Here are the methods you need to implement so the game provider could request the operator for balance operations.

4.2.1 getBalance

Type: Request

This method returns the actual balance of the player's account.

Request parameters:

Key	Value type	Required?	Description
`callerId`	Integer	Yes	A numeric operator ID. It's used to distinguish the operator accounts
`playerName`	String	Yes	The ID of the player the operator supplied us with
`currency`	String	Yes	A three-character ISO 4217 code for the currency of the amount
`gameId`	String	No	An ID of the game on the game provider side
`sessionId`	String	No	The ID of game session
`sessionAlternativeId`	String	No	An alternative session identifier
`bonusId`	String	No	The ID of the bonus free round program

Response parameters:

Key	Value type	Required?	Description
balance	Integer	Yes	The actual player's balance
freeroundsLeft	Integer	No	The number of remaining free rounds

Request example:


```
{
   "jsonrpc": "2.0",
   "method": "getBalance",
   "id": 1928822492,
   "params": {
      "callerId": 365,
      "playerName": "noname",
      "currency": "USD",
      "gameId": "bennys_the_biggest_game",
      "sessionAlternativeId": "78342b60-809e-4735-bde4-90abe4aa9e8c"
   }
}
```


Response example:


```
{
   "jsonrpc": "2.0",
   "id": 1928822492,
   "result": {
      "balance": 44
   }
}
```


4.2.2 withdrawAndDeposit

Type: Request

withdrawAndDeposit provides a more efficient way of making deposits and withdraws as it requires one call rather than two.

In case of a bonus free round game, this method should be processed as follows:

If the value of the withdraw field > 0 and the number of bonus free rounds remaining >= the value of the chargeFreerounds field, then reduce the number of bonus free rounds remaining by the value of the chargeFreerounds field and ignore the debiting of the amount from the player's balance specified in the withdraw field.

If the condition described above is not met, the player's balance should be decreased on the amount from the withdraw field.

Request parameters:

Key	Value type	Required?	Description
callerId	Integer	Yes	A numeric operator ID. It's used to distinguish the operator accounts
playerName	String	Yes	The ID of the player the operator supplied us with
withdraw	Integer	Yes	An amount to withdraw from the player's account
deposit	Integer	Yes	An amount to deposit into the player's account (in cents)
currency	String	Yes	A three-character ISO 4217 code for the currency of the amount
transactionRef	String	Yes	An unique ID of transaction on the game provider side
gameRoundRef	String	No	An ID of the game round on the game provider side. Unique only within a *session series*
gameId	String	No	An ID of the game on the game provider side
source	String	No	A reason type for clearing a game round.
reason	String	No	The reason of calling the API method
sessionId	String	No	The ID of game session
sessionAlternativeId	String	No	An alternative session identifier
spinDetails	Object	No	Details of the spin. See the table below
bonusId	String	No	The ID of the bonus free round program
chargeFreerounds	Integer	No	The number of free rounds to reduce the free rounds counter by

spinDetails object:

Key	Value type	Required?	Description
betType	String	No	*Type of bet* in the game
winType	String	No	*Type of win* in the game

reason field:

Key	Value type	Description
GAME_PLAY	String	Mark of the NOT latest action in the game round
GAME_PLAY_FINAL	String	Mark of the latest action in the game round

Response parameters:

Attention: The operator must guarantee the uniqueness of transactionId generated on the operator side.

Key	Value type	Required?	Description
newBalance	Integer	Yes	The actual player's balance after the transaction execution (in cents)
transactionId	String	Yes	The transaction ID on the operator side
freeroundsLeft	Integer	No	The number of remaining free rounds

Request example:

```
{
   "jsonrpc": "2.0",
   "method": "withdrawAndDeposit",
   "id": 1928822492,
   "params": {
      "callerId": 365,
      "playerName": "noname",
      "withdraw": 20,
      "deposit": 0,
      "currency": "USD",
      "transactionRef": "1:P0GKaKsJEqMzKnEk",
      "gameRoundRef": "kq",
      "gameId": "bennys_the_biggest_game",
      "reason": "GAME_PLAY_FINAL",
      "sessionId": "n5vpp2xp406c8f5",
      "sessionAlternativeId": "78342b60-809e-4735-bde4-90abe4aa9e8c",
      "spinDetails": {
         "betType": "spin",
         "winType": "standart"
      }
   }
}
```


Response example:


```
{
   "jsonrpc": "2.0",
   "id": 1928822492,
   "result": {
      "newBalance": 30,
      "transactionId": "2:E0GKbKsJEqMzKnEk"
   }
}
```


4.2.3 rollbackTransaction

Type: Request

This method rolls back the transaction if something went wrong.

The transaction rollback may be performed for withdrawals as well as deposits.

In case of withdrawAndDeposit needs to be reverted, the method must remove the amount (in cents) of the transaction from the player's account.

In case of a bonus free round game deposit or withdraw needs to be reverted, the method must remove (deposit) or increase (withdraw) the amount (in cents) of the transaction from the player's bonus account and increase the number of free rounds.

In case of transaction not exist on your side, you should save it and mark as rollbacked.

Request parameters:

Key	Value type	Required?	Description
`callerId`	Integer	Yes	A numeric operator ID. It's used to distinguish the operator accounts
`playerName`	String	Yes	The ID of the player the operator supplied us with
`transactionRef`	String	Yes	An unique ID of transaction on the game provider side
`gameId`	String	No	An ID of the game on the game provider side
`sessionId`	String	No	The ID of game session
`sessionAlternativeId`	String	No	An alternative session identifier

Response parameters:

Empty response.

The game provider system considers it executed by checking the right HTTP status code and existence of the result field:

- The HTTP status code is *200 OK*.
- The result field must present, the error field must not present.

Otherwise, the rollback is marked as **unsuccessful** and **will be repeated for 2 days with an interval of 60 seconds**.

Request example:

```
{
   "jsonrpc": "2.0",
   "method": "rollbackTransaction",
   "id": 1928822492,
   "params": {
      "callerId": 365,
      "playerName": "noname",
      "transactionRef": "1:P0GKaKsJEqMzKnEk",
      "gameId": "bennys_the_biggest_game",
      "sessionId": "n5vpp2xp406c8f5",
      "sessionAlternativeId": "78342b60-809e-4735-bde4-90abe4aa9e8c"
   }
}
```

Response example:

```
{
    "jsonrpc": "2.0",
    "id": 1928822492,
    "result": {}
}
```

4.2.4 Seamless errors

The server on the operator's side should be ready to send the errors listed below if the cause of the error matches its description.

Error 1 ErrNotEnoughMoneyCode

Game Provider tried to debit money from the player's balance, but the player does not have enough money on it.

Error 2 ErrIllegalCurrencyCode

Game Provider sends a request with a currency code that does not support by the server on the operator's side.

Error 3 ErrNegativeDepositCode

Value of `withdraw` field in the *withdrawAndDeposit* request is less than zero or incorrect.

This value should be greater than zero.

Error 4 ErrNegativeWithdrawalCode

Value of `deposit` field in the *withdrawAndDeposit* request is less than zero or incorrect.

How it works under the hood

This section provides information about how operator/seamless API works.

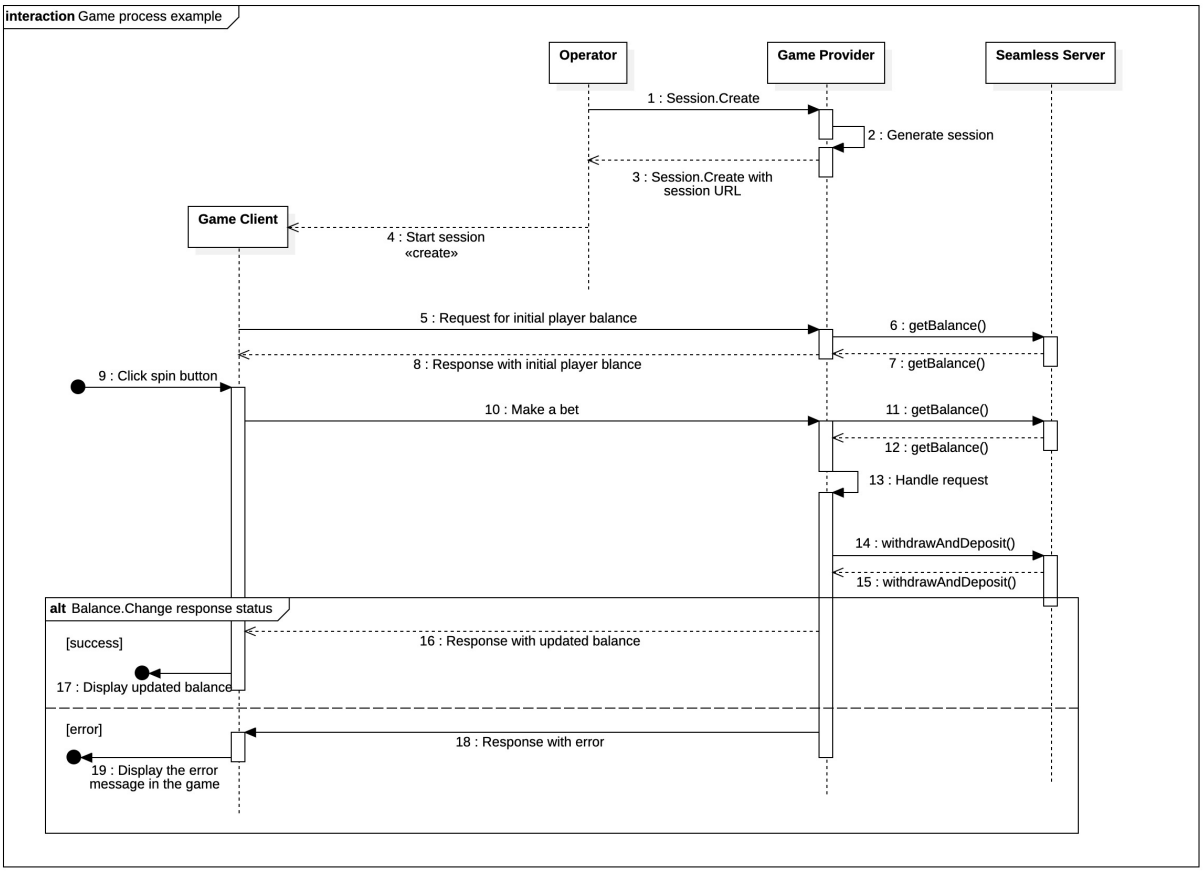
5.1 Operator API

5.1.1 Game process example

Methods: *Session.Create*, *getBalance*, *withdrawAndDeposit*

Definitions: *Operator*, *Game Provider*, *Seamless server*, *Game Client*

This example shows the main game process.



5.1.2 Principles of work of the RestorePolicy parameter

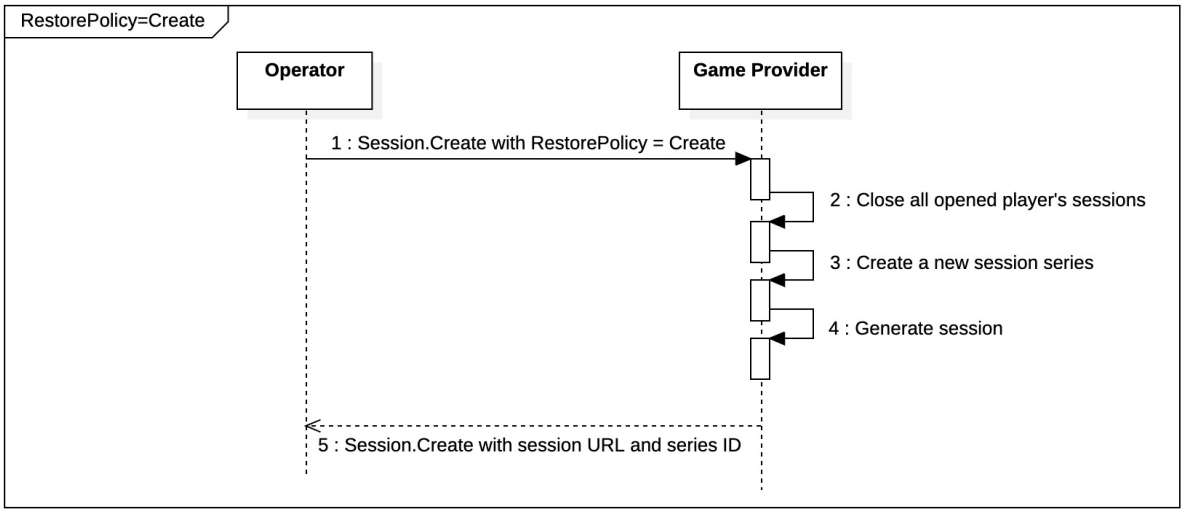
Methods: *Session.Create*

Definitions: *Operator*, *Game Provider*, *Series of sessions*

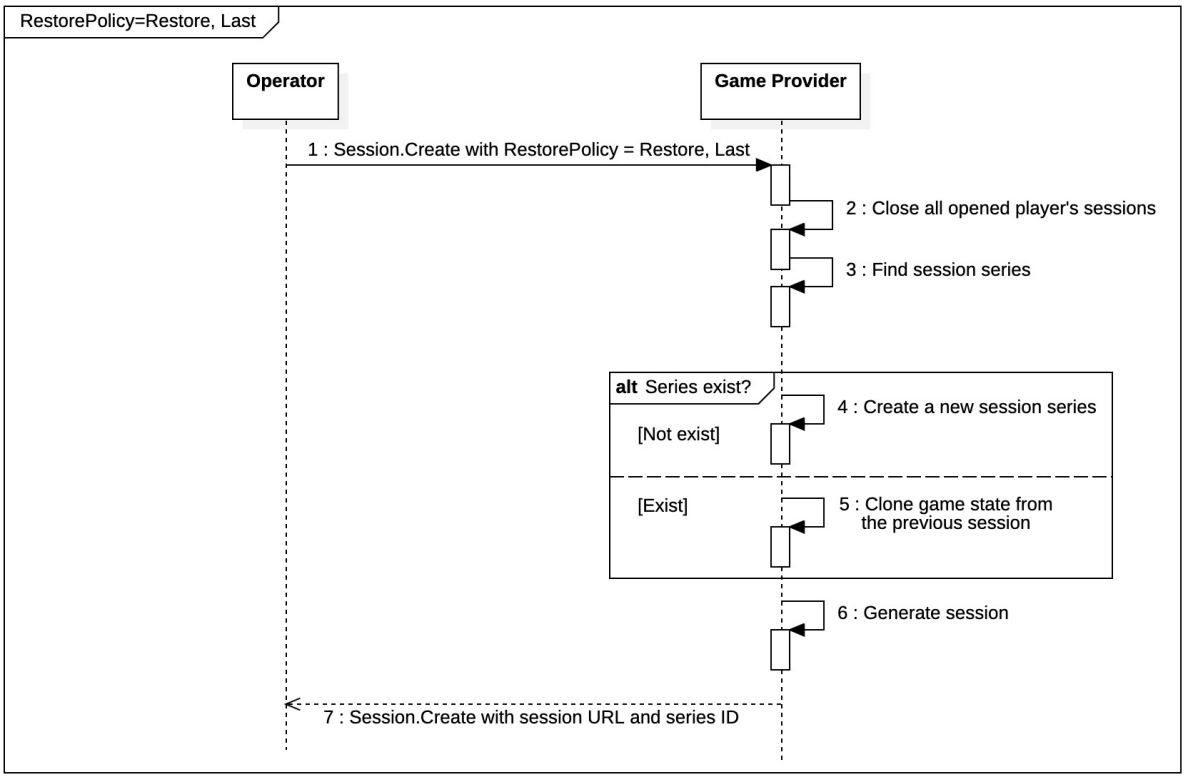
`RestorePolicy` - a required parameter in *Session.Create* method.

Note: When a new session created all previous open sessions get closed (implicitly).

This example shows the process of creating a session with the parameter `RestorePolicy=Create`.



This example shows the process of creating a session with the parameter `RestorePolicy=Restore` or `RestorePolicy=Last`.



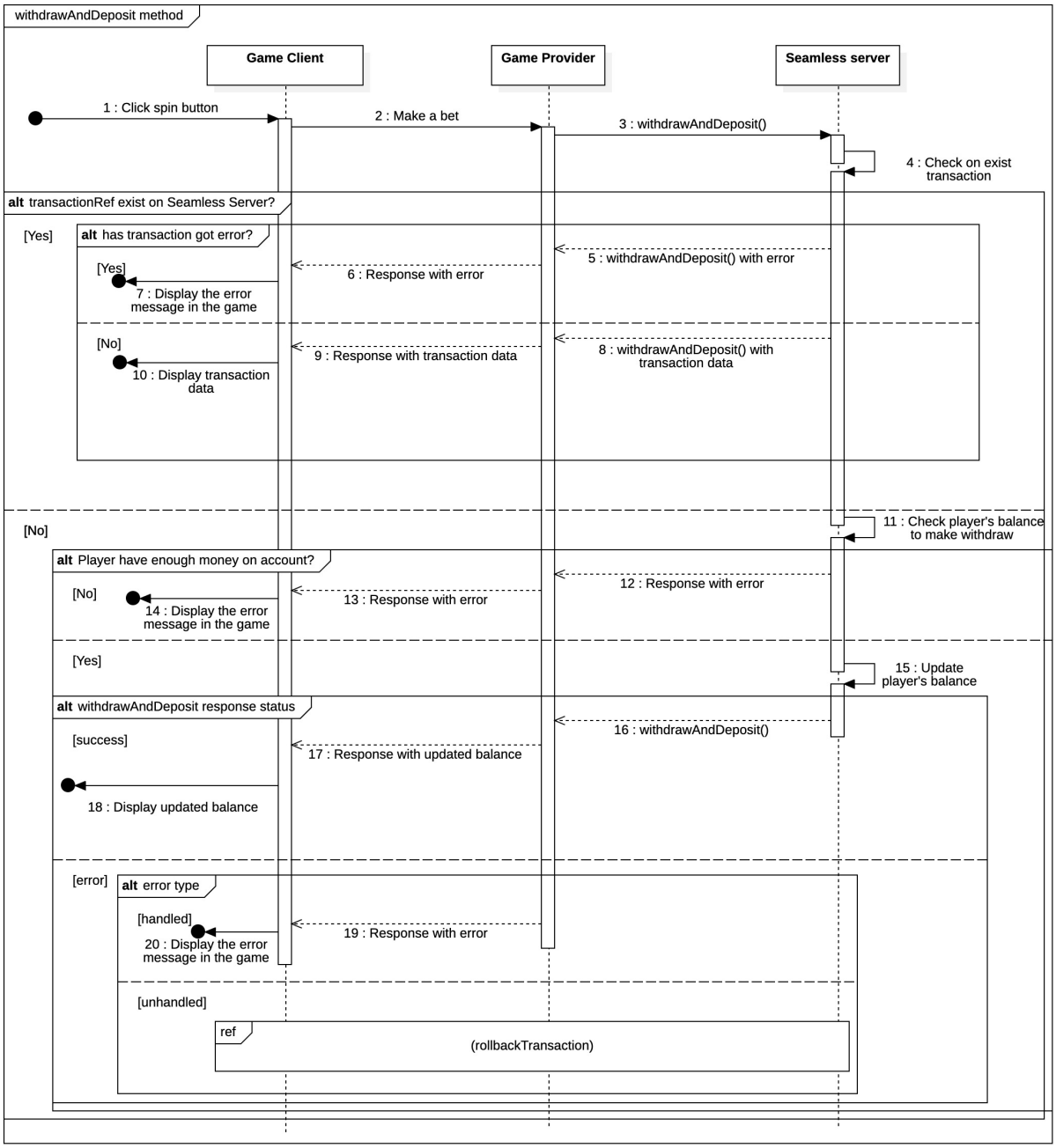
5.2 Seamless V2 API

5.2.1 withdrawAndDeposit method

Methods: *withdrawAndDeposit*, *rollbackTransaction*

Definitions: *Game Client*, *Game Provider*, *Seamless server*

This example shows how the withdrawAndDeposit method works.



5.2.2 rollbackTransaction method

Methods: *rollbackTransaction*

Definitions: *Game Client*, *Game Provider*, *Seamless server*

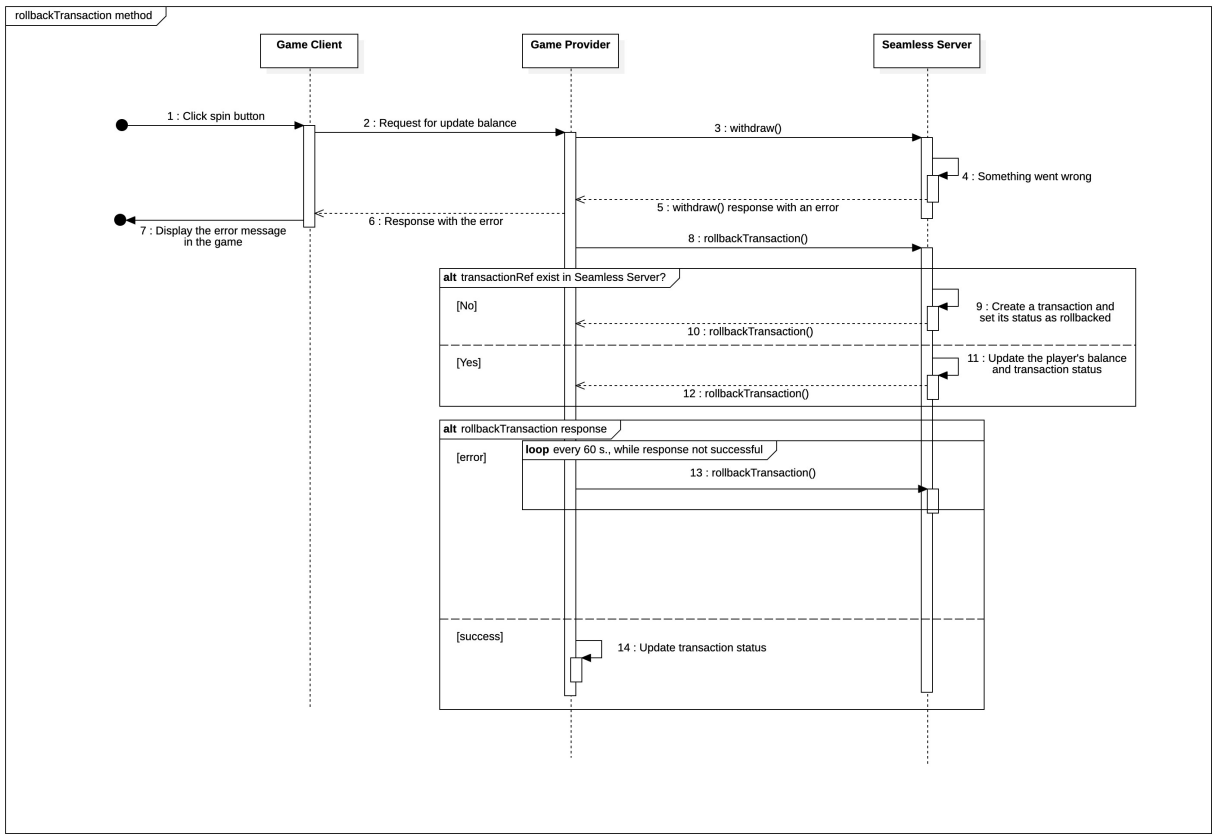
This method rolls back the transaction if something went wrong.

The transaction rollback may be performed for withdrawals as well as deposits.

In case of the withdrawAndDeposit needs to be reverted, the method must re-deposit the amount of the transaction to the player's account.

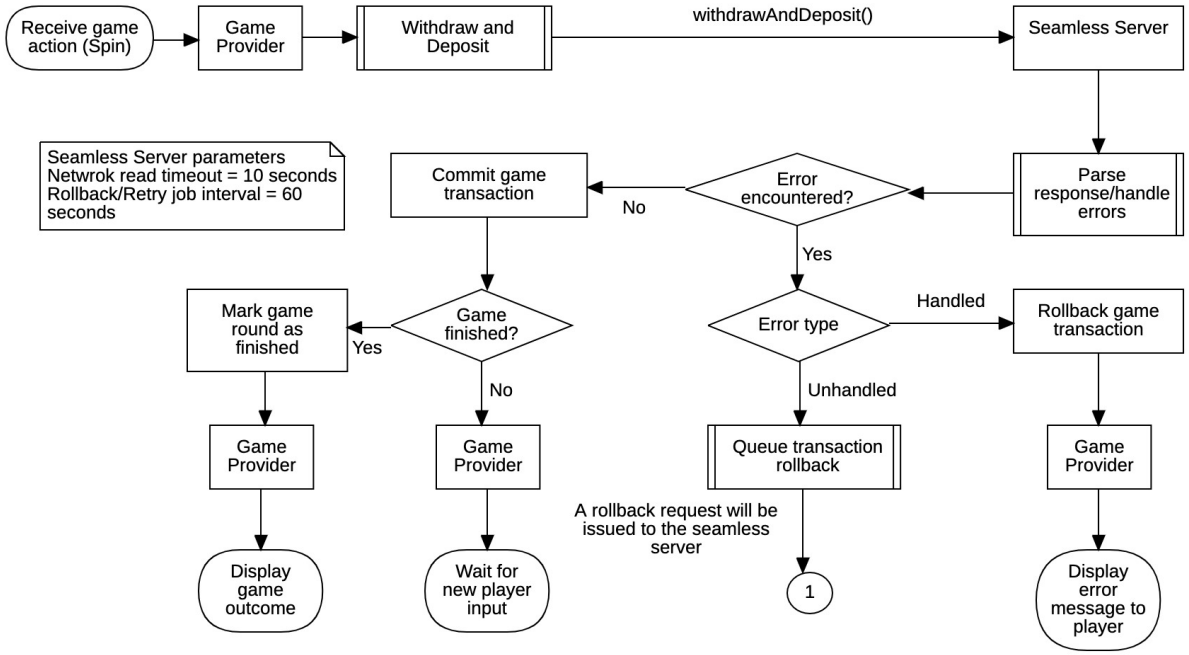
In case of a bonus free round game deposit or withdraw needs to be reverted, the method must remove (deposit) or increase (withdraw) the amount (in cents) of the transaction from the player's bonus account and increase the number of free rounds.

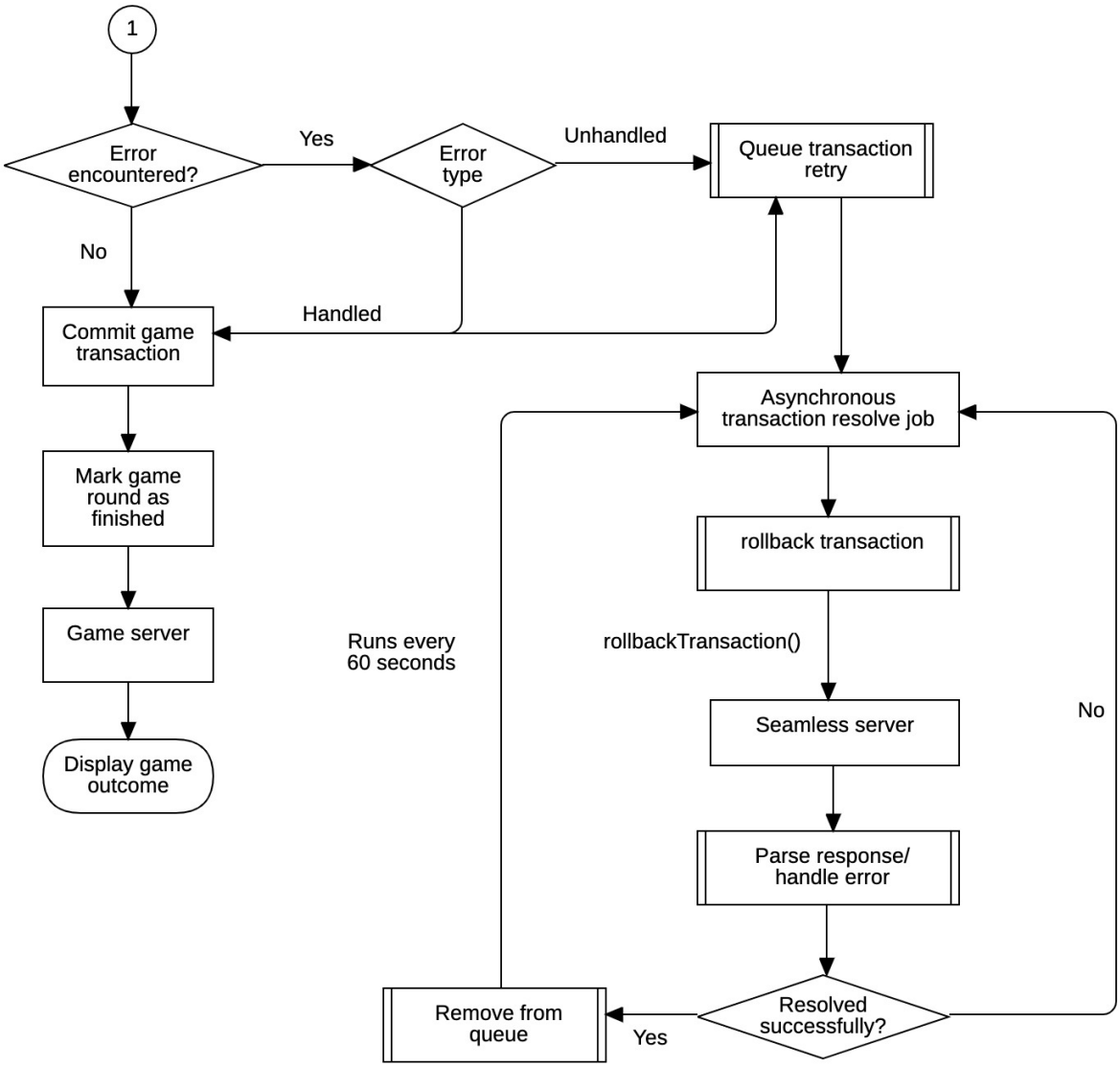
In case of transaction not exist on your side, you should save it and mark as rollbacked.



5.2.3 Error handling and rollback mechanism

The following flowcharts describe error handling and transaction rollback mechanism when using the *seamless V2 API*.


Receive game action (Spin)
Game Provider
Withdraw and Deposit
withdrawAndDeposit()
Seamless Server
Seamless Server parameters
Netwrok read timeout = 10 seconds
Rollback/Retry job interval = 60 seconds
Commit game transaction
Error encountered?
No
Parse response/handle errors
Yes
Mark game round as finished
Game finished?
Yes
No
Error type
Handled
Rollback game transaction
Unhandled
Game Provider
Game Provider
Queue transaction rollback
Game Provider
A rollback request will be issued to the seamless server
Display game outcome
Wait for new player input
1
Display error message to player


1
Error encountered?
Yes
Error type
Unhandled
Queue transaction retry
No
Handled
Commit game transaction
Asynchronous transaction resolve job
Mark game round as finished
rollback transaction
Game server
Runs every 60 seconds
rollbackTransaction()
Seamless server
No
Display game outcome
Parse response/ handle error
Remove from queue
Yes
Resolved successfully?

Errors, possible problems and solutions

Error response has a structure according to the JSON-RPC 2.0 specification.

6.1 Error 10502 SESSION_INVALID_PARAMS

Type of API: *Operator API*

Game Provider tried to create a game session, but something went wrong.

Possible causes:

- You enter incorrect parameters in the request
- Player or/and bank group not exist
- You do not receive any requests from the game provider's server
- Your *Seamless Server* could not handle the *getBalance* request or return an incorrect answer on this request
- Incorrect HTTP-headers
- Incorrect request/response encoding

Possible solutions:

- Verify your request. All fields should be like here: the *Session.Create*
- Create a player or/and bank group
- Check your IP's white list (if it exists). An IP-address of Game Provider's server must be in your white list and must be correct.
- Verify the response from your *Seamless Server* on *getBalance* request
- The HTTP-headers have to be strictly the same as these:

```
Content-Type: application/json
Content-Length: must contain the correct length according to the HTTP-
↪specification
Accept: application/json
```

- Set request/response encoding to UTF-8

6.2 Error 16383 INTERNAL_ERROR

Type of API: *Operator API*

An internal error occurred on the Game's Provider server.

Possible solutions:

- Report about it immediately to your account manager

6.3 Error 495 Cert Error after executing Operator API method

This error occurs due to problems with the certificate.

Possible reasons for the error:

- The SSL certificate has expired
- The PEM-file is not sent with the request

Possible solutions:

- Send the PEM-file along with the request
- Re-issue the SSL certificate by following steps in the "*Get client certificate*" topic

6.4 Error message 'CLOSED' instead of the game

This means that the session has been already ended (closed).

Possible solutions:

- The player should start a new session.

6.5 Error message 'SERVER ERROR' instead of the game

An internal error occurred on the Game's Provider server.

Possible solutions:

- Report about it immediately to your account manager

Application

7.1 Currencies

All currencies are specified by ISO 4217 currencies codes.

Currency Description	Code
AED / United Arab Emirates dirham	AED
ALL / Albanian Lek	ALL
AMD / Armenian dram	AMD
AOA / Angolan Kwanza	AOA
ARS / Argentine peso	ARS
AUD / Australian dollar	AUD
AZN / Azerbaijani manat	AZN
BAM / Bosnia and Herzegovina convertible mark	BAM
BDT / Bangladeshi Taka	BDT
BGN / Bulgarian lev	BGN
BHD / Bahraini Dinar	BHD
BLR / Belarusian ruble	BLR
BMD / Bermudian Dollar	BMD
BND / Bruneian Dollar	BND
BOB / Bolivian Boliviano	BOB
BRL / Brazilian real	BRL
BTC / Bitcoin	BTC
BYN / Belarusian ruble	BYN
CAD / Canadian dollar	CAD
CDF / Congolese franc	CDF
CHF / Swiss franc	CHF
CLP / Chilean peso	CLP
CNY / Chinese yuan	CNY
COP / Colombian peso	COP
CRC / Costa Rican colon	CRC
CZK / Czech koruna	CZK
DAS / Dash Crypto	DAS
DKK / Danish krone	DKK
DOG / Dogecoin	DOG
DOP / Dominican peso	DOP
DTC / Datacoin	DTC
DZD / Algerian Dinari	DZD

continues on next page

Table 1 – continued from previous page

Currency Description	Code
EGP / Egyptian Pound	EGP
EOS / Cryptocurrency	EOS
ETB / Ethiopian birr	ETB
ETH / Ethereum	ETH
EUR / Euro	EUR
GBP / Pound sterling	GBP
GEL / Georgian lari	GEL
GHS / Ghanaian Cedi	GHS
GMC / Gridmaster	GMC
GMD / Gambian Dalasi	GMD
GNF / Guinean franc	GNF
HKD / Hong Kong dollar	HKD
HRK / Croatian Kuna	HRK
HTG / Haitian gourde	HTG
HUF / Hungarian Forint	HUF
IDR / Indonesian rupiah	IDR
ILS / Israeli new shekel	ILS
INR / Indian rupee	INR
IQD / Iraqi Dinar	IQD
IRR / Iranian Rial	IRR
IRT / Iranian touman	IRT
ISK / Icelandic Krona	ISK
JOD / Jordanian Dinar	JOD
JPY / Japanese yen	JPY
KES / Kenyan shilling	KES
KGS / Kyrgyzstani Som	KGS
KHR / Cambodian Riel	KHR
KRW / South Korean Won	KRW
KWD / Kuwaiti Dinar	KWD
KZT / Kazakhstani tenge	KZT
LBP / Lebanese pound	LBP
LRD / Liberian dollar	LRD
LTC / Litecoin	LTC
MAD / Moroccan dirham	MAD
MBC / MicroBitcoin	MBC
MBH / Unknown	MBH
MDL / Moldovan leu	MDL
MET / Metronome	MET
MGA / Malagasy Ariary	MGA
MHC / MetaHash	MHC
MKD / Macedonian denar	MKD
MLT / Maltese Lira	MLT
MMK / Burmese Kyat	MMK
MNT / Mongolian Tughrik	MNT
MXM / Maximine Coin	MXM
MXN / Mexican peso	MXN
MYR / Malaysian ringgit	MYR
MZN / Mozambican Metical	MZN
NAD / Namibian dollar	NAD

continues on next page

Table 1 – continued from previous page

Currency Description	Code
NAN / Not Available Now	NAN
NGN / Nigerian Naira	NGN
NIO / Nicaraguan córdoba	NIO
NOK / Norwegian krone	NOK
NZD / New Zealand dollar	NZD
OMR / Omani Rial	OMR
PEN / Peruvian Sol	PEN
PHP / Philippine peso	PHP
PKR / Pakistani rupee	PKR
PLN / Polish złoty	PLN
PPT / Populous	PPT
PYG / Paraguayan guaraní	PYG
QAR / Qatari Riyal	QAR
RON / Romanian Leu	RON
RSD / Serbian dinar	RSD
RUB / Russian ruble	RUB
SAR / Saudi Arabian Riyal	SAR
SCR / Seychelles Rupee	SCR
SDG / Sudanese Pound	SDG
SEK / Swedish krona/kronor	SEK
SGD / Singapore Dollar	SGD
SLL / Sierra Leonean leone	SLL
SRD / Surinamese dollar	SRD
SZL / Swazi lilangeni	SZL
THB / Thai baht	THB
TJS / Tajikistani Somoni	TJS
TMT / Turkmenistan manat	TMT
TND / Tunisian dinar	TND
TRY / Turkish lira	TRY
TWD / Taiwan New Dollar	TWD
TZS / Tanzanian Shilling	TZS
UAH / Ukrainian hryvnia	UAH
UBC / µBTC	UBC
UGX / Ugandan shilling	UGX
USD / United States dollar	USD
UYU / Uruguayan peso	UYU
UZS / Uzbekistani Som	UZS
VEF / Venezuelan bolívar	VEF
VES / Venezuelan bolívar	VES
VND / Vietnamese Dong	VND
XAF / Cameroon Franc	XAF
XAU / Gold Ounce	XAU
XEM / Cryptocurrency	XEM
XMC / mBCH mili Bitcoin Cash	XMC
XME / mETH mili Ethereum	XME
XML / mLTC mili Litecoin	XML
XMR / Monero	XMR
XOF / CFA franc	XOF
XUT / USDT USD Tether	XUT

continues on next page

Table 1 – continued from previous page

Currency Description	Code
ZAR / South African rand	ZAR
ZEC / ZCash	ZEC
ZMW / Zambian kwacha	ZMW
ZWL / Zimbabwean dollar	ZWL

Note: Some currencies have unknown description, because of they are not defined in ISO 4217.

Note: NAN - it is a special currency that is defined for purposes when currency is unknown.

Note: The currency list may differ from what it is and might be changed by your request.

7.2 Bet types

This section provides essential bet types.

Value	Description
split	additional bet. BlackJack game only. Player split his initial hand into 2 parallel hands
insure	additional bet. BlackJack game only. Player 'buy' insurance for his hand from dealer's blackjack
spin	player`s action, bet in the base game in slots
collect	player`s action. collect current spin`s winnings
draw	Video-poker games only. Reissue of cards draw
picksymbol	player`s action, special symbol selection for bonus or free games feature choose
bonus	player`s action in bonus game
drop	an automatic avalanche of the winning symbols on the reels and the issuance of new ones instead of the collapsed ones (without an additional bet)
shiftreels	an automatic shift of reels to the right by 1 position, and on the first reel new values are given (without additional bet)
pickwintype	choice of winning option or bonus feature or free spin feature
deal	bet in video-poker games
bonusaction	player`s action in bonus game
respin	an automatic re-rotation of reels issuing new symbols, possible sticky symbols, etc. (no additional bet)
double	winning`s double feature
freespin	free game`s spin (without additional bet)
stand	BlackJack game only. Player's choice. Stop of cards getting
platinumaction	player`s action in bonus game
init	game`s initialisation
hit	BlackJack game only. Player's choice. Get one more card
standart	same as spin - standard (base) game winnings
free	free game`s spin win (without additional bet)
tumble	same as drop / avalanche
halfcollect	partial winnings collect in 'double' feature
SG	game feature, where the part of regular symbol wins will be paid as additional game

7.3 Win types

This section provides essential win types.

Value	Description
standart	Standard win
free	Win during free spins
bonus	Bonus win
double	Double (gamble) win

7.4 Vocabulary

This section provides essential definitions.

7.4.1 Game provider

Remote gaming server (casino platform) that provides games by API.

7.4.2 Game client

The game interface the player interacts with.

7.4.3 Operator

The Customer's project that integrates games by API.

7.4.4 Seamless server

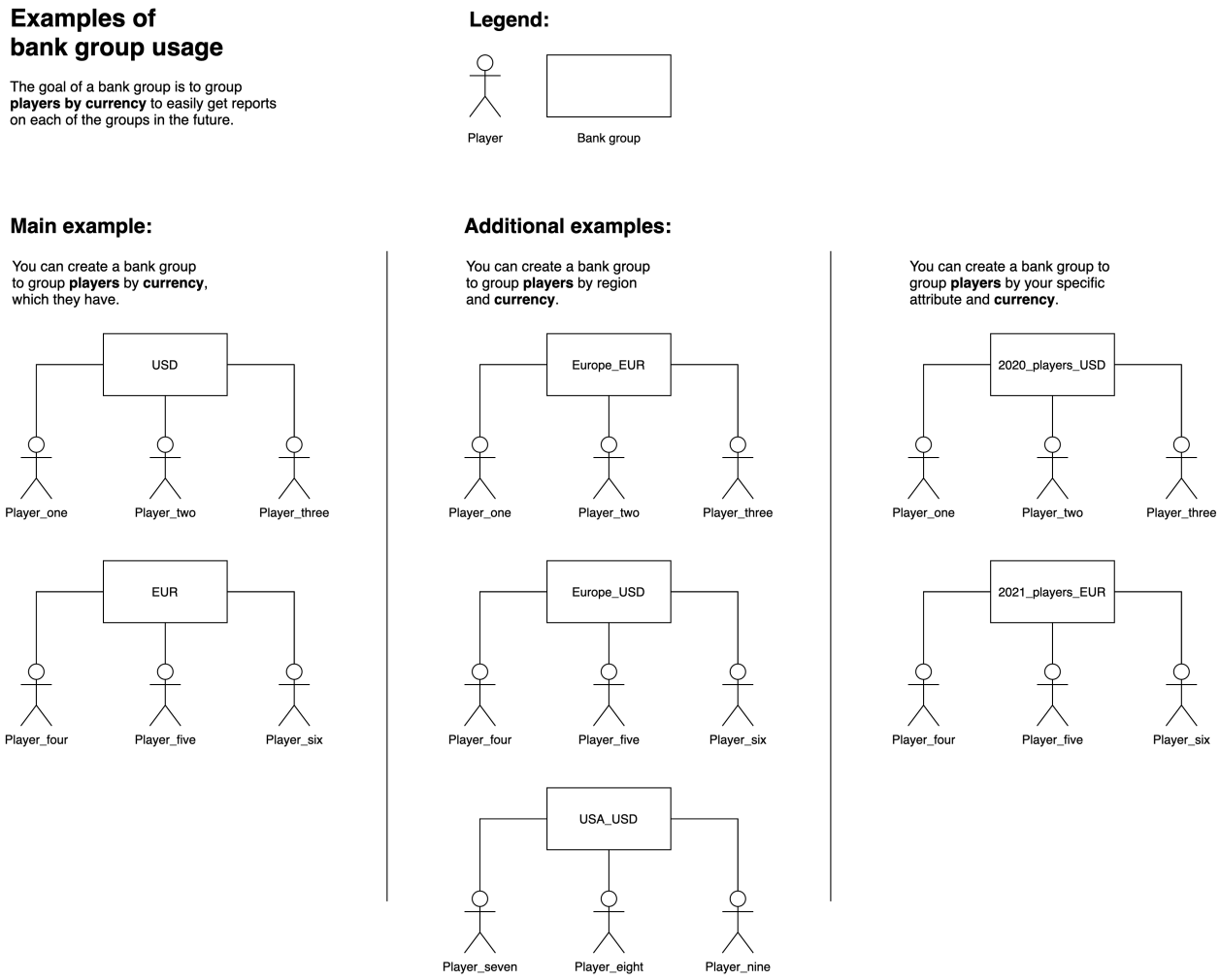
The server part of the Operator, which serves and modifies players balances from the Game provider API.

7.4.5 Back-office

Game provider's Admin panel.

7.4.6 Bank Group

This is a special group for players, which classify players by currencies.
Bank group has two parameters: currency and settings patch.
You can set a currency for the bank group only once. Nevertheless, you can apply settings patch.
You can read more about bank group's parameters *here*.



7.4.7 Spin

To make spin means to play one bet in a game with a win or lose.

7.4.8 Free round

This is a bonus game session, where a player can spend his/her free spins.

7.4.9 Bonus balance

This is a separate type of balance that does not overlap with a player's balance.

That balance is replenished by wining in free rounds.

The player or operator can execute a bonus balance to the player's balance.

7.4.10 Series of sessions

This is a set of sessions that are combined in order to be able to transfer the state of the game from one session to another for a specific player.