Overview

This document describes an API based on HTTP/1.1 protocol [RFC 2616 (https://www.ietf.org/rfc/rfc2616)].

Document version

1.0.7

Links

- RFC 2616 (https://tools.ietf.org/html/rfc2616), Hypertext Transfer Protocol -- HTTP/1.1
- ISO 4217 (http://www.iso.org/iso/home/standards/currency_codes.htm), Currency codes
- ISO 8601 (http://www.iso.org/iso/home/standards/iso8601.htm), Date and time format

Changelog

- `1.0.0 (2016-09-01, d6)`
    - documentation initialized
- `1.0.1 (2016-09-07, d2)`
    - games/init POST request format specified
- `1.0.2 (2016-09-15, d2)`
    - limits and self-validate endpoints added
- `1.0.3 (2016-09-23, d2)`
    - specify response status on duplicate requests
- `1.0.4 (2016-10-20, d2)`
    - "is_mobile" parameter added to games
- `1.0.5 (2016-11-15, d2)`
    - demo mode
- `1.0.6 (2017-02-17, d2)`
    - updated "/limits" response
- `1.0.7 (2017-03-21, d2)`
    - added "/jackpots" endpoint
- `1.0.8 (2017-03-30, d2)`
    - added "game_uuid" and "player_id" to "bet", "win" and "refund" requests

GR

Overview

Integration data provided by GR

1. Merchant ID
2. Merchant Key
3. Base API URL

Endpoints and Base API URL

For example Base API URL is *https://gr.com/api/v1 (https://gr.com/api/v1)* and Endpoint is */games/lobby*

Then calls from integrator to GR should be done to *https://gr.com/api/v1/games/lobby (https://gr.com/api/v1/games/lobby)*

Request format

Query parameters should be passed with `application/x-www-form-urlencoded` content type

Response format

Default response format is `json` with `Content-Type: application/json` header

List of used HTTP codes

- `200` : OK. Everything worked as expected.
- `201` : A resource was successfully created in response to a `POST` request. The `Location` header contains the URL pointing to the newly created resource.
- `204` : The request was handled successfully and the response contains no body content (like a `DELETE` request).
- `304` : The resource was not modified. You can use the cached version.
- `400` : Bad request. This could be caused by various actions by the user, such as providing invalid JSON data in the request body, providing invalid action parameters, etc.
- `401` : Authentication failed.
- `403` : The authenticated user is not allowed to access the specified API endpoint.
- `404` : The requested resource does not exist.
- `405` : Method not allowed. Please check the `Allow` header for the allowed HTTP methods.
- `415` : Unsupported media type. The requested content type or version number is invalid.
- `422` : Data validation failed (in response to a `POST` request, for example). Please check the response body for detailed error messages.
- `429` : Too many requests. The request was rejected due to rate limiting.
- `500` : Internal server error. This could be caused by internal program errors.

Error response

Generic errors

Generic error response contains a single object with following attributes:

- `name` , `string` , exception name
- `message` , `string` exception message
- `code` , `integer` , `default: 0` , exception code
- `status` , `integer` HTTP status code

Response example:

```
HTTP/1.1 404 Not Found
...

{
    "name": "Not Found Exception",
    "message": "The requested resource was not found.",
    "code": 0,
    "status": 404
}
```

Collections

Collection is a set of objects of the same type. There is an additional metadata for working with collections like pagination or sorting.

Pagination headers

By default, pagination metadata is available via HTTP headers:

- `X-Pagination-Total-Count` : The total number of resources;
- `X-Pagination-Page-Count` : The number of pages;
- `X-Pagination-Current-Page` : The current page (1-based);
- `X-Pagination-Per-Page` : The number of resources in each page;
- `Link` : A set of navigational links allowing client to traverse the resources page by page.

Collections enveloping

In case your client is incapable of working with HTTP headers you're able to receive this information within response body.

Response example:

```
HTTP/1.1 200 OK
...
X-Pagination-Total-Count: 1000
X-Pagination-Page-Count: 50
X-Pagination-Current-Page: 1
X-Pagination-Per-Page: 20
Link: <https://gr-url.com/endpoint?page=1>; rel=self,
      <https://gr-url.com/endpoint?page=2>; rel=next,
      <https://gr-url.com/endpoint?page=50>; rel=last
Content-Type: application/json; charset=UTF-8

{
    "items": [
        {
            "id": 1,
            ...
        },
        {
            "id": 2,
            ...
        },
        ...
    ],
    "_links": {
        "self": {
            "href": "https://gr-url.com/endpoint?page=1"
        },
        "next": {
            "href": "https://gr-url.com/endpoint?page=2"
        },
        "last": {
            "href": "https://gr-url.com/endpoint?page=50"
        }
    },
    "_meta": {
        "totalCount": 1000,
        "pageCount": 50,
        "currentPage": 1,
        "perPage": 20
    }
}
```

Game launch flow

Games should be stored/cached on the client side after retrieval. Game could be launched in several steps according to scenario based on lobby availability.

Games without lobby:

1. Call `/games/init`
2. Launch game by redirecting player to the provided URL

Games with lobby:

1. Call `/games/lobby`
2. Call `/games/init` with provided `lobby_data`
3. Launch game by redirecting player to the provided URL

> Info: More info on `/games` , `/games/lobby` and `/games/init` endpoints could be found in corresponding documentation sections. Note: Base API URL should be provided by manager.

Security

All requests should contain authorization headers (except **Launch** phase with player redirection).

Authorization headers

- `X-Merchant-Id` : Merchant ID provided by integration manager
- `X-Timestamp` : Request timestamp. If differ from current timestamp for more than 30 seconds - request considered expired
- `X-Nonce` : Random string
- `X-Sign` : Sign calculated with sha1 hmac

X-Sign calculation

1. Merge request array with authorization headers array
2. Sort resulting array by key in ascending order
3. Generate a URL-encoded query string from this array
4. Use sha1 hmac algorithm with Merchant Key (provided by integration manager) for signing

PHP example of the X-Sign calculation

```
$merchantKey = 'Merchant Key provided by integration manager';

$headers = [
    'X-Merchant-Id' => 'value',
    'X-Timestamp'   => time(),
    'X-Nonce'       => md5(uniqid(mt_rand(), true)),
];

$requestParams =  [
    'game_uuid' => 'abcd12345',
    'currency'  => 'USD',
];

$mergedParams = array_merge($requestParams, $headers);
ksort($mergedParams);
$hashString = http_build_query($mergedParams);

$XSign = hash_hmac('sha1', $hashString, $merchantKey);
```

Example

Request:

```
GET /games
...
X-Merchant-Id: ff955b5759b3885f08cf125d4454ceb4
X-Timestamp: 1471857411
X-Nonce: e115cf0f66a645aca08225c9c1b20b80
X-Sign: 1bb7e4cd5c43f9885ba6a1758ad30fc562f88821
...
```

Games

Endpoint URL

```
/games
```

[`GET /`] Retrieving games list

You will receive games collection available for your Merchant ID

Game item fields

- `uuid` : `string` , Game UUID that will be used in `/init` and `/lobby`
- `name` : `string` , Game name
- `image` : `string` , Game image url
- `type` : `string` , Game type
- `provider` : `string` , Game provider name
- `has_lobby` : `integer` , 1 or 0 - indicates if game has lobby
- `is_mobile` : `integer` , 1 or 0 - indicates if game used for mobile devices and should be opened in new window (not in iframe or some container)

Example

Request:

```
GET /games HTTP/1.1
...
```

Response:

```
HTTP/1.1 200 OK
...

{
    "items": [
        {
            "uuid": "abcd12345",
            "name": "Book of Ra",
            "image": "https://image-url.com",
            "type": "Slots",
            "provider": "abcd12345",
            "has_lobby": 0,
            "is_mobile": 0
        },
        {
            "uuid": "abcd12345",
            "name": "Baccarat",
            "image": "https://image-url.com",
            "type": "Baccarat",
            "provider": "abcd12345",
            "has_lobby": 1,
            "is_mobile": 0
        }
        ...
    ],
    "_links": {
        "self": {
            "href": "https://gr-url.com/endpoint?page=1"
        },
        "next": {
            "href": "https://gr-url.com/endpoint?page=2"
        },
        "last": {
            "href": "https://gr-url.com/endpoint?page=50"
        }
    },
    "_meta": {
        "totalCount": 1000,
        "pageCount": 50,
        "currentPage": 1,
        "perPage": 20
    }
}
```

Lobby

If game has lobby integrator should call this action to get lobby tables, so player can choose which table to play.

Endpoint URL

```
/games/lobby
```

[`GET /`] Returns list of tables for the selected game

Request fields

- `game_uuid` : `string` , `required` , Game UUID provided in `/games`
- `currency` : `string` , `required` , Player currency that will be used in this game session.

Response fields

- `lobby` : `array` , Contains lobby data of the selected game with following attributes:

- `lobbyData` : `string` , Data required on `/games/init` phase for *lobby_data* parameter
- `name` : `string` , Table name
- `isOpen` : `boolean` , True or false - indicates if game is open right now
- `openTime` : `string` , Lobby open time
- `closeTime` : `string` , Lobby close time
- `dealerName` : `string` , Dealer name
- `dealerAvatar` : `string` , Dealer avatar url
- `limits` : `array` , Table limits

Example

Request:

```
GET /games/lobby?game_uuid=abc123&currency=USD HTTP/1.1
...
```

Response:

```
HTTP/1.1 200 OK
...

{
    lobby: {
        lobbyData: "abcd12345",
        name: "Baccarat",
        isOpen: true,
        openTime: "11:00:00",
        closeTime: "12:00:00",
        dealerName: "abcd12345",
        dealerAvatar: "https://avatar-url.com",
        limits: {
            {
                currency:"USD",
                min: 1,
                max: 100
            }
        }
    }
}
```

Init

This action will prepare game for launch and return final url where player should be redirected to start playing.

Endpoint URL

```
/games/init
```

`[ POST / ]` Initializing game session

Request fields

- `game_uuid` : `string` , `required` , Game UUID provided in `/games`
- `player_id` : `string` , `required` , Unique player ID on the integrator side
- `player_name` : `string` , `required` , Player nickname that will be shown in some games
- `currency` : `string` , `required` , Player currency that will be used in this game session
- `session_id` : `string` , `required` , Game session ID on the integrator side
- `return_url` : `string` , `optional` , Redirect player to this url after game ends
- `language` : `string` , `optional` , Player language
- `email` : `string` , `optional` , Player email
- `lobby_data` : `string` , `optional` , Used only for games with lobby. Provided in `/lobby`

Example

Request:

```
POST /games/init HTTP/1.1
...

game_uuid=abcd12345&player_id=abcd12345&player_name=abcd12345&currency=USD& ....
```

Response:

```
HTTP/1.1 200 OK
...

{
    "url": "https://gr-url.com/endpoint"
}
```

Init demo game (only if provider has demo mode)

This action will prepare game for launch in demo mode and return final url where player should be redirected to start playing.

Endpoint URL

```
/games/init-demo
```

`[ POST / ]` Initializing game session

Request fields

- `game_uuid` : `string` , `required` , Game UUID provided in `/games`
- `return_url` : `string` , `optional` , Redirect player to this url after game ends
- `language` : `string` , `optional` , Player language

Example

Request:

```
POST /games/init-demo HTTP/1.1
...

game_uuid=abcd12345&language=en&return_url=....
```

Response:

```
HTTP/1.1 200 OK
...

{
    "url": "https://gr-url.com/endpoint"
}
```

Game launch

To launch the game redirect player to the URL returned by `/games/init` or `/games/init-demo` .

Integrator

Overview

Integrator should provide endpoint URL to communicate with GR during the game session

GR could send 4 type of calls to integrator

- Balance
- Win
- Bet
- Refund

Request format

All calls from GR to integrator will be done via `POST` and parameters will be passed with `application/x-www-form-urlencoded` content type

Response format

All integrator responses should have `Content-Type: application/json` header, `json` format and `HTTP/1.1 200 OK` status code.

Error format

In case of error integrator should return json object with following attributes and `HTTP/1.1 200 OK` status code.

- `error_code` : `string` , Error code (specific for every action)
- `error_description` : `string` , Human readable error description. Can be empty

```
HTTP/1.1 200 OK
...

{
    "error_code": "INSUFFICIENT_FUNDS",
    "error_description": "Not enough money to continue playing"
}
```

Error codes

There are only 2 error codes:

- `INSUFFICIENT_FUNDS` : code used in **bet** action when player has insufficient funds
- `INTERNAL_ERROR` : code used in all other cases meaning that action has not been executed: player not found, database or file system errors, etc

Security

All GR requests contains authorization headers.

Authorization headers

- `X-Merchant-Id` : Merchant ID provided by integration manager
- `X-Timestamp` : Request timestamp. If differ from current timestamp for more than 30 seconds - request considered expired
- `X-Nonce` : Random string
- `X-Sign` : Sign calculated with sha1 hmac

X-Sign calculation

1. Merge request array with authorization headers array
2. Sort resulting array by key in ascending order
3. Generate a URL-encoded query string from this array
4. Use sha1 hmac algorithm with Merchant Key (provided by integration manager) for signing

PHP example of the X-Sign validation

```
$merchantKey = 'Merchant Key provided by integration manager';

$headers = [
    'X-Merchant-Id' => 'Get header value',
    'X-Timestamp'   => 'Get header value',
    'X-Nonce'       => 'Get header value',
];

$XSign = 'Get header value';

$mergedParams = array_merge($_POST, $headers);
ksort($mergedParams);
$hashString = http_build_query($mergedParams);

$expectedSign = hash_hmac('sha1', $hashString, $merchantKey);

if ($XSign !== $expectedSign) {
    throw new \Exception('Invalid sign');
}
```

Example

Request:

```
POST <client_callback_endpoint> HTTP/1.1
Content-Type: application/x-www-form-urlencoded
...
X-Merchant-Id: ff955b5759b3885f08cf125d4454ceb4
X-Timestamp: 1471857411
X-Nonce: e115cf0f66a645aca08225c9c1b20b80
X-Sign: 1bb7e4cd5c43f9885ba6a1758ad30fc562f88821

param=value&param2=value2
...
```

Balance

GR will call this action to retrieve actual player balance

Endpoint URL

```
<client_callback_endpoint>
```

`[ POST / ]` Balance request

Request fields

- `action` : `string ["balance"]` , Action "balance"
- `player_id` : `string` , Unique player ID on integrator side
- `currency` : `string` , Balance currency

Response fields

- `balance` : `double` , `required` , Player's balance

Example

Request:

```
POST <client_callback_endpoint> HTTP/1.1
...

action=balance&player_id=123456&currency=USD
```

Response:

```
HTTP/1.1 200 OK
...

{
    "balance": 57.12
}
```

Bet

This action is called when player trying to make a bet

Bet types

- `bet` : default bet type
- `tip` : tip for a dealer

Request fields

- `action` : `string ["bet"]` , Action "bet"
- `amount` : `double` , Bet amount
- `currency` : `string` , Bet currency
- `game_uuid` : `string` , Game UUID from the list of games `/games`
- `player_id` : `string` , Unique player ID on integrator side
- `transaction_id` : `string` , Unique transaction ID on GR side
- `session_id` : `string` , Integrator game session ID, provided in `/games/init`
- `type` : `string` , "bet" or "tip"

Response fields

- `balance` : `double` , `required` , Player's balance after transaction
- `transaction_id` : `string` , `required` , Unique transaction ID on the integrator side

> Important: Bet with provided **transaction_id** should be processed only once. If you already processed this transaction, then return successful response with processed transaction ID on the integrator side

Example

Request:

```
POST <client_callback_endpoint> HTTP/1.1
...

action=bet&amount=10.00&currency=USD&transaction_id=abcd12345&session_id=abcd12345&type=bet
```

Response:

```
HTTP/1.1 200 OK
...

{
    "balance": 27.18,
    "transaction_id": "abcd12345",
}
```

Win

Action called when player win in a game

Win types

- `win` : default win type
- `jackpot` : player get a jackpot

Request fields

- `action` : `string ["win"]` , Action "win"
- `amount` : `double` , Win amount
- `currency` : `string` , Win currency
- `game_uuid` : `string` , Game UUID from the list of games `/games`
- `player_id` : `string` , Unique player ID on integrator side
- `transaction_id` : `string` , Unique transaction ID on GR side
- `session_id` : `string` , Integrator game session ID, provided in `/games/init`
- `type` : `string` , "win" or "jackpot"

Response fields

- `balance` : `double` , `required` , Player's balance after transaction
- `transaction_id` : `string` , `required` , Unique transaction ID on the integrator side

> Important: Win with provided **transaction_id** should be processed only once. If you already processed this transaction, then return successful response with processed transaction ID on the integrator side

Example

Request:

```
POST <client_callback_endpoint> HTTP/1.1
...

action=win&amount=100.00&currency=USD&transaction_id=abcd12345&session_id=abcd12345&type=win
```

Response:

```
HTTP/1.1 200 OK
...

{
    "balance": 170.21,
    "transaction_id": "abcd12345",
}
```

Refund

Refund is a cash back in case bet problems.

After receiving `refund` call integrator should cancel corresponding bet transaction and return funds to player. If such bet transaction does not exists then integrator should just save this refund transaction and respond with success.

Request fields

- `action` : `string ["refund"]` , Action "refund"
- `amount` : `double` , Refund amount
- `currency` : `string` , Refund currency
- `game_uuid` : `string` , Game UUID from the list of games `/games`
- `player_id` : `string` , Unique player ID on integrator side
- `transaction_id` : `string` , Unique transaction ID on GR side
- `session_id` : `string` , Integrator game session ID, provided in `/games/init`
- `bet_transaction_id` : `string` , GR bet transaction ID to be refunded

Response fields

- `balance` : `double` , `required` , Player's balance after transaction
- `transaction_id` : `string` , `required` , Unique refund transaction ID on the integrator side

> Important: Bet with provided **bet_transaction_id** should be refunded processed only once. If you already refunded this transaction, then in response return processed refund transaction ID on the integrator side

Example

Request:

```
POST <client_callback_endpoint> HTTP/1.1
...

action=refund&amount=10.00&currency=USD&transaction_id=abcd12345&session_id=abcd12345&bet_transaction_id=abcd1234
```

Response:

```
HTTP/1.1 200 OK
...

{
    "balance": 27.18,
    "transaction_id": "abcd12345",
}
```

Additional requests to GR

Merchant limits

Returns list of limits for merchant

Endpoint URL

```
/limits
```

[`GET /`] Returns list of limits for merchant

Response fields

- `amount` : `string` , Amount left
- `currency` : `string` , Limit currency
- `provider` : `array` , List of providers attached to this limit

Example

Request:

```
GET /limits
...
```

Response:

```
HTTP/1.1 200 OK
...

[
    {
        "amount": "1000.00",
        "currency": "USD",
        "providers":[
            "Provider1",
            "Provider2",
            "Provider3"
        ]
    },
    {
        "amount": "1000.00",
        "currency": "EUR",
        "providers":[
            "Provider1"
        ]
    }
]
```

List of jackpots

Returns list of jackpots for every game provider (if available) assigned to merchant key. List of jackpots is cached for 60 seconds.

Endpoint URL

```
/jackpots
```

[`GET /`] Returns list of jackpots assigned to merchant key

Response fields

- `name` : `string` , `null` , Jackpot name (string or null if game provider does not have names for jackpots)
- `amount` : `string` , Amount left
- `currency` : `string` , Limit currency
- `provider` : `string` , Game provider

Example

Request:

```
GET /jackpots
...
```

Response:

```
HTTP/1.1 200 OK
...

[
    {
        "name": "jackpot name",
        "amount": "1000.00",
        "currency": "USD",
        "provider": "Provider1"
    },
    {
        "name": null,
        "amount": "1000.00",
        "currency": "EUR",
        "provider": "Provider2"
    }
]
```

Integrator self validation

Integrator could check if implementation on his side is correct. To start validation integrator should have active game session (opened not longer than 15 minutes ago). GR will send set of requests ('bet', 'win', etc) during validation and return result in response.

Endpoint URL

```
/self-validate
```

`[ POST / ]` Self validation

Response fields

- `success` : `boolean` , true or false - indicates if validation is passed and implementation is correct
- `log` : `array` , Validation log

Example

Request:

```
POST /self-validate
...
```

Response:

```
HTTP/1.1 200 OK
...

{
    "success": true,
    "log": [
        "Log message",
        "Log message",
        ...
    ]
}
```