Table of Contents

[Brief integration information 1](#_heading=h.gjdgxs)

[Administrative panel and Server 1](#_heading=h.3znysh7)

[Basic information 2](#_heading=h.1t3h5sf)

[Command: getBalance - (get user balance from side site) 3](#_heading=h.4d34og8)

[Command: writeBet - (result BET-WIN) 4](#_heading=h.2s8eyo1)

[Get Games List 7](#_heading=h.17dp8vu)

[**Games List Sorting 8**](#_heading=h.34tifch1bofm)

[**Open Games 9**](#_heading=h.26in1rg)

[Games images preload functionality 11](#_heading=h.lnxbz9)

[Adding fullscreen functionality into iframe 12](#_heading=h.35nkun2)

[Close games: 17](#_heading=h.1ksv4uv)

[Save current page location after close game 17](#_heading=h.44sinio)

[Add browser back button functionality 18](#_heading=h.2jxsxqh)

[Hide open game links and games picture (for game lobby) with RewriteEngine 18](#_heading=h.z337ya)

[Create hall and Change settings through API: 19](#_heading=h.3j2qqm3)

[**Jackpots 20**](#_heading=h.i842wm3e11mi)

[Games Logs 22](#_heading=h.4i7ojhp)

[**All type response actions 24**](#_heading=h.dzjye2ys9awq)

# Brief integration information

Command **getBalance** send POST JSON request to the site server to check player balance;

Command **writeBet** send POST JSON request to the site server to change player balance;

1. Create API file from the site connected to real database for processing commands **getBalance** and **writeBet**
2. Under the Hall Settings in our Administrative Panel use **TEST API** to check response from site server
3. TEST API - Get games list
4. TEST API - Open game
5. Close games, example implementation [demo.gamesapi.biz](http://demo.gamesapi.biz) in pc and mobile

# 

# 

# Administrative panel and Server

1. Create hall (one hall can only support one currency type)
2. Setup hall settings

RTP: win/bet

MIN TOTAL BET, MAX TOTAL BET (change the bets settings in all slots games except EGT)

Hall key: setup manually

Callback URL: address to the API file from the side site

1. TEST API getBalance and writeBet

request = from game server

response = from site side

If response is empty need to check API with Postman request POST method form-data

1. Tab GAMES / CONFIG

4.1 **NOVO, IGRO, ARISTO**

Bet Limit: Limitation of possible multiplication for 1 spin by total bet

Example: if Bet Limit 4000, then the player will never win multiplication by 5000.

Example: 5 MAN symbols in the Novo game BOOK OF \*\* will never win.

Risk Limit: Limitation of possible winning in the risk game.

4.2 **IG, EG, AMAT, ARISTO** Additionally, supported multiple input of maximum multiplication separated by comma:

Example: 100,100,100,100,50 (important without spaces)

Under above settings, the players with small bets will get max multiplication by 100 within 0 to 80%. Players with high bets will get max multiplication no more than 50 within 80 to 100%.

It’s also supported to input 4, 5 or 10 combinations of Bet Limit, they will proportionally affect bets multiplication.

1. To view players statistics in the hall, click on computers icon  near hall name, or in the hall settings button USERS, On the right side of the player, Joystick  icon displays game logs in pictures

Game history logs are saved for 10 Days

# 

# 

# 

# Basic information

| **METHOD** | **POST** |
| --- | --- |
| **REQUEST** | **json** |
| **RESPONSE** | **json** |
| **TIME ZONE** | **GTM + 0 (Server time zone)** |

# Command: getBalance - (get user balance from side site)

| **Key** | **Value** | **Description** |
| --- | --- | --- |
| cmd | getBalance |  |
| hall | 3 | Hall id |
| key | 123456 | Hall key |
| login | letst1001 | User login on side site |

EXAMPLE REQUEST FROM GAME SERVER

{

"cmd":"getBalance",

"hall":"3",

"key":"123456",

"login":"test"

}

RESPONSE FROM SITE(response speed should be less than 80 ms):

{

"status":"success",

"error":"",

"login":"test",

"balance":"1000.00", // Balance value must be returned in Data Type: Decimal 12,2

"currency":"LAK"

}

Example fail response:

{

"status":"fail",

"error":"ERROR CODE"

}

\*Balance value must be returned in Data Type: Decimal 12,2 (Example: 1000.00)

| **ERROR CODE** | **Description** |
| --- | --- |
| user\_not\_found | User not found on the side site |

# Command: **writeBet** - (result BET-WIN)

In the writeBet command, only the "bet" and "win" parameters need to be processed.

The **"bet"** value needs only to check if the balance is enough before processing the **"win"** value.

If balance is enough, then subtract **"bet"** from the balance and add **"win"** value.

if balance not enough for bet:

{

"status":"fail",

"error":"fail\_balance"

}

Additionally, depending on your platform architecture, you can check the **sessionId** and **tradeId** parameters.

**sessionId** is the unique id of the game session, which comes in the openGame response when you request to open a new game session.

**tradeId** is the unique id of the current request of the writeBet command.

**round\_finished** - a parameter indicating the completion of the round, available values:

**true** - the round is finished

**false** - the round is active

**null** - the parameter is not supported by the game/provider, in this case, it should be assumed that each writeBet is completed and the results of bet and win come together in one request.

All other parameters are additional and the information is available depending on the provider, you can **only store their value**:

| betInfo | Additional information, such as the number of lines selected and the bet per line: "lines":20,"bet":"0.05" |
| --- | --- |
| matrix | Spin matrix |
| date | Date and time |
| winLines | The win lines |
| action | The current transaction type, for example "spin" or "freeSpin", each provider has its own description, therefore a complete list of possible "actions" is not available, as it differs depending on the rules of the game |

ONLY FOR Sport Betting API

On receiving **refund** parameter, do not check balance, in this case recalculation of bet value can cause user balance to become negative.

This can happen if there was a recalculation from a win to a refund, and funds were paid out during the time between the first calculation and the refund.

The **refund** parameter comes in "betInfo", example: "betInfo":"refund"

EXAMPLE REQUEST FROM GAME SERVER:

{

"cmd":"writeBet",

"login":"letst1001",

"hall":"3",

"key":"123456",

"sessionId":"22333427",

"bet":"100",

"win":"100",

"tradeId":"9999",

"action":"spin",

"betInfo":"bet",

"gameId":"1",

"matrix":"",

"date":2015-02-11 16:35:02,

"WinLines":""

}

RESPONSE FROM SITE(response speed should be less than 80 ms):

{

"status":"success",

"error":"",

"login":"letst1001",

"balance":"900.00",

"currency":"LAK"

}

Example fail response:

{

"status":"fail",

"error ":"ERROR CODE"

}

| **ERROR CODE** | **Description** |
| --- | --- |
| user\_not\_found | User not found on the side site |
| fail\_balance | Not enough balance player for bet |

| **if bet** | **win** | **Balance before** | **Balance after** |
| --- | --- | --- | --- |
| 100 | 100 | 1000.00 | 1000.00 |
| 100 | 0 | 1000.00 | 900.00 |
| 100 | 10 | 1000.00 | 910.00 |

| **if bet** | **win** | **Balance before** | **Balance after** |
| --- | --- | --- | --- |
| 100 | 100 | 50.00 | fail\_balance |
| 100 | 0 | 50.00 | fail\_balance |
| 100 | 10 | 50.00 | fail\_balance |

| **Key** | **Value** | **Description** | **Data types** |
| --- | --- | --- | --- |
| cmd | writeBet | Command name | string |
| hall | 3 | Hall id | number |
| key | 123456 | Hall key | string |
| login | letst1001 | User login id | string |
| bet | 100 | Bet value | decimal 12,2 |
| win | 100 | Win value | decimal 12,2 |
| tradeId | 9999 | Spin trade id | string |
| betInfo | bet | Bet info | string |
| gameId | 1 | Game id | number |
| matrix |  | Screen matrix | string |
| date |  | Spin date and time | datetime |
| WinLines |  | Win lines description | string |
| sessionId | 2222 | Session id | number |
| action | bet | Game action | string |
| round\_finished | null | Round bool or null | boolean || null |

# 

| OPTIONAL additional encrypted signature for **getBalance** and **writeBet** commands    Current functionality is enabled only by request.  If activated, in commands **getBalance** and **writeBet** instead of **"key"** parameter, our server will send **"sign"** in sha256 format, you will need to check this signature on your side.  **"sign"** parameter must not be sent to our server.  The **"key"** parameter is still used in the openGame command.    Example function to check the signature, where you need to pass our **writeBet** data and Hall Key:    function sign($data,$key){  ksort($data, SORT\_STRING);  array\_push($data, $key);  $data = implode(':', $data);  $sign=hash('sha256',$data);  return $sign;  }    $data - parameters that came in writeBet  $key - hall key |
| --- |

# Get Games List

It is recommended to automate get **getGamesList** on member login or save it in Database and update maximum 1 time per 1 hour. In this case you will always have our up-to-date games list.

url="http://SERVER/API/";

{  
 "hall": "hall\_id",  
 "key": "hall\_key",  
 "cmd": "getGamesList",  
 "img":"game\_img\_2", // Optional parameter

"cdnUrl":"http://domain.com/resources" // Optional parameter  
}

The parameter "img" is optional and specifies the desired thumbnail style to be returned in the response. Currently, the following 3 styles are supported:

• **game\_img\_2**: 288x210px (Horizontal design)

• **game\_img\_5**: 268x418px (Vertical design)

• **game\_img\_6**: 300x300px (Square design)

If no "img" parameter is provided, the default thumbnail style is **game\_img\_2** (Horizontal design).

The "**cdnUrl**" parameter is optional and serves as a helper to direct the game path URL to your CDN. It is particularly useful when utilising rewrite rules to proxy our resources through your CDN infrastructure.

In response you will receive a games list grouped by providers.

Example response:

| {  "status": "success",  "microtime": 0.00280014991760254,  "dateTime": "2024-04-12 12:12:14",  "error": "",  "content": {  "provider\_title": [  {  "id": "game\_id",  "name": "game\_name",  "img": "game\_img\_thumbnail\_url",  "device": "device\_support",  "title": "game\_provider\_title",  "categories": "game\_category",  "bm": "free\_spin\_support\_flag",  "demo": "demo\_mode\_support\_flag",  "rewriterule": "rewriterule\_support\_flag",  "exitButton": "exit\_button\_flag"  },  ]  } } |
| --- |

# Games List Sorting

In getGamesList response you receive each game as following:

| **Parameter** | **Value** | **Description** |
| --- | --- | --- |
| id | Game id | The game unique ID |
| name | Game name | The game name |
| img | Thumbnail URL | The URL to game icon |
| device | 0 - only supports Desktop PC  1 - only supports Mobile devices  2- supports both PC and Mobile | This parameter indicated the game client supported devices |
| title | Game Provider | The game provider name |
| categories | slots  live\_dealers  fast\_games | The game category |
| bm | 0 - false  1 - true | Promo free spins support flag |
| demo | 0 - false  1 - true | Demo (fun) mode, this flag indicates if demo mode is supported by this game |
| rewriterule | 0 - false  1 - true | Rewrite rules are used to proxy game resources via operator servers |
| exitButton | 0 - false  1 - true | Indicates if you need to place a custom close button to close the game, in case if in-game exit button do not redirect to "exitUrl" on game launch |

# Open Games

url="http://SERVER/API/openGame/";

{

"cmd":"openGame",

"hall":"hall\_id",

"domain":"https://domain",

"exitUrl":"https://domain/[close.php](https://api.gamesapi.biz/API/IntegrationFiles.zip)",

"language":"en",

"key":"hall\_key",

"login":"player\_login",

"gameId":"1",

"cdnUrl":"http://domain.com/resources",

"demo":"0"

}

**"cdnUrl"** – parameter in **openGame** to replace the link with your domain for the games that support **rewriterule**

**"login"** – player's login **cannot be less than 2 characters long** and also contain spaces or symbols **‘:’;#**

**"domain"** - parameter is required for protocol identification

if send as **http** -> the game URL will be returned as **http**, for **https** -> please send domain parameter with **https**

Example: **"domain":"https://your\_domain.com/"**

Additionally, **domain** parameter is used to close some games, the game client redirects to **domain/close.php**, game close events are described further in the Close Game section.

You can use **exitUrl** parameter in case you need an individual link to close the games without **close.php**. On close event, the game client will redirect to appointed **exitUrl**

**"demo":"0"** (demo mode for registered users, "0"=false "1"=true)

RESPONSE FROM GAME SERVER ON OPENGAME

{

"status":"success",

"error":"",

"content":{

"game":{

"url":game\_url, game url

"iframe":"1" iframe support flag

"width":"1"

"withoutFrame":"1"

"exitButton\_mobile":"1" flag for add in game iframe exit button on mobile

"exitButton":"1" flag for add in game iframe exit button on pc

"disableReload":"1"

"wager":"1"

"bonus":"1"

﻿"rewriterule":"1" RewriteEngine support flag

},

"gameRes":{

"sessionId":session\_id game session id

}}}

**"width":** for pc

if **"width":"0"** the game supports 3:4 screen ratio

if **"width":"1"** the game only supports 16:9 screen ratio

**"withoutFrame"**: for mobile

If **"0"** the game must be opened in iframe on the same page, this helps to handle close game events and stay at the same position on the page

If the game do not support iframe, you will receive **"withoutFrame":"1"**, in this case you need to redirect to our server response link

**"wager":"1"** (supported wager bonus mode)

**"bonus":"1"** (supported deposit bonus mode (cashback))

# Games images preload functionality

For images preload is used [Intersection Observer API](https://developer.mozilla.org/ru/docs/Web/API/Intersection_Observer_API). In the following example we create observer object with callback function and settings, where:

· rootMargin – viewport margin (by default is used browser viewport)

· threshold – a value, that indicates the visibility percentage of observed element to invoke callback function

Then we select required elements and apply observer in a loop

In callback function we check, if an element is in viewport by entries[i].isIntersecting and if it was loaded

var observer = new IntersectionObserver(function (entries, observer) {

for (var i = 0; i < entries.length; i++) {

var $target = $(entries[i].target);

if (entries[i].isIntersecting && !$target.hasClass('loaded')) {

var img = new Image();

img.target = $target;

img.target.addClass('loading');

img.src = $(entries[i].target).attr('data-img');

img.onload = function () {

this.target.css('background-image', 'url(' + this.src + ')');

this.target.removeClass('loading');

this.target.addClass('loaded');

}

img.onerror = img.onload;

observer.unobserve(entries[i].target);

}

}

}, {

rootMargin: '0px',

threshold: 0.1

});

document.querySelectorAll('.game-element').forEach(game=>{

observer.observe(game);

})

On games list layout we indicate a link to image in data-attribute data-img='${game['img']} to prevent premature loading. Once an element is entered into viewport and after completed loading, in the observer callback function data-attribute is changed into background-image .

# Adding **fullscreen** functionality into iframe

All modern browsers, except Safari on iOS, support [Fullscreen API](https://developer.mozilla.org/en-US/docs/Web/API/Fullscreen_API). To add fullscreen is required:

1) add fullscreen function:

function toggleFullScreen(isFull) {

if ((document.fullScreenElement && document.fullScreenElement !== null) ||

(!document.mozFullScreen && !document.webkitIsFullScreen)) {

if (isFull===false) {

return;

}

if (document.documentElement.requestFullScreen) {

document.documentElement.requestFullScreen();

} else if (document.documentElement.mozRequestFullScreen) {

document.documentElement.mozRequestFullScreen();

} else if (document.documentElement.webkitRequestFullScreen) {

document.documentElement.webkitRequestFullScreen();

}

} else {

if (isFull===true) {

return;

}

if (document.cancelFullScreen) {

document.cancelFullScreen();

} else if (document.mozCancelFullScreen) {

document.mozCancelFullScreen();

} else if (document.webkitCancelFullScreen) {

document.webkitCancelFullScreen();

}

}

}

2) add a call of fullscreen function on the required element (in current example a click event is bind to **$(document)** and will be invoked in any webpage location). Additionally, you can add a check on device type – desktop, mobile, tablet.

$(document).click(() => {

toggleFullScreen(true);

});

Since **Safari on iOS** does not support Fullscreen API, to enter in fullscreen mode you can add an overlay with height greater than device height. In this case on scroll will be invoked a native fullscreen mode. When native fullscreen mode is activated – height/width of the website is equal to height/width of the device, an overlay is hidden. An invoke event is similar to resize event.

IMPORTANT: custom overlay on iOS must be enabled only in landscape mode, do not show it in portrait mode.

1) In HTML you need to add overlay element, responsible for scroll

<div class="safarihelper-bg">

<div id="safarihelper"></div>

</div>

2) Then add css styles

body.core-scrollcheck {

width: 100%;

height: 3000px;

position: absolute;

overflow: auto;

}

body.core-scrollcheck .core-bonus-message {

position: fixed !important;

}

.notFullscreenSafari {

overflow: auto !important;

}

.fullscreenSafari {

overflow: hidden !important;

touch-action: none;

-ms-touch-action: none;

position: relative !important;

}

#safarihelper{

color: white;

font-size: 26px;

font-family: 'tekomedium','serif';

text-align: center;

touch-action: none;

pointer-events: none;

position: fixed;

height: 304px;

width: 90px;

top: 10px;

right: 40px;

transform-origin: center top;

z-index: 10000;

cursor: pointer;

background-image: url(../images/fullscreenanim\_ios\_hand\_move.png);

background-repeat: no-repeat;

background-position: 0 -310px;

}

.safarihelper-bg {

position: fixed;

top: 0;

left: 0;

right: 0;

bottom: 0;

width: 100%;

height: 100vh;

background-color: rgba(0,0,0,0.5);

z-index: 999999;

}

.safarihelper-bg, #safarihelper {

opacity: 0;

visibility: hidden;

transition: opacity .5s linear, visibility .5s linear;

}

.safarihelper-bg.active, #safarihelper.active {

opacity: 1;

visibility: visible;

}

.safarihelper-bg:not(.active), #safarihelper:not(.active) {

pointer-events: none;

}

3) Add a function to enter into fullscreen mode in iOS

function windowResize() {

var w=window.innerWidth;

var h=window.innerHeight;

var isFullScreen = false;

var ua = window.navigator.userAgent,

iOS = !!ua.match(/iPad/i) || !!ua.match(/iPhone/i),

webkit = !!ua.match(/WebKit/i),

iOSSafari = iOS && webkit && !ua.match(/CriOS/i);

if (window.orientation!==undefined && ((w>h && window.orientation==0 )|| (w<h && window.orientation!=0))) {

setTimeout(function() {windowResize();},1000);

return;

}

if ((device.mobile() || device.tablet()) && window.orientation==0) {

$('.device-rotate').addClass('active');

} else {

$('.device-rotate').removeClass('active');

}

if (device.mobile() || device.tablet()) {

$('body').attr('orientation',window.orientation==0?'portrait':'landscape');

}

if (device.ios() && !device.desktop() && !device.tablet() && iOSSafari) {

if (($('body').hasClass('mobile-game') && !$('body').hasClass('game-without-scrollcheck')) {

$('body').addClass('core-scrollcheck');

} else {

$('body').removeClass('core-scrollcheck');

}

if (window.screen.width === h && window.orientation!=0) {

isFullScreen = true;

} else {

isFullScreen = false;

}

if (!isFullScreen && window.orientation!=0 && (!$('body').hasClass('game-without-scrollcheck') && $('body').hasClass('mobile-game'))) {

$('html, body').removeClass('fullscreenSafari').addClass('notFullscreenSafari');

} else if (!$('body').hasClass('game-without-scrollcheck') && $('body').hasClass('mobile-game')) {

$('html, body').addClass('fullscreenSafari').removeClass('notFullscreenSafari');

}

}

}

4) Add a function invoke on website loading and resize

windowResize()

$(window).resize(function() {windowResize()});

The above code is working correctly only in Safari on iOS, because another browser’s address tab is not hidden completely.

Disable scroll in game iframe on iOS 15

Since iOS 15 does not work standard methods to disable scroll on the page when modal windows are opened (overflow:hidden or event.preventDefault() on touchmove event), you can use an example from [here](https://pqina.nl/blog/how-to-prevent-scrolling-the-page-on-ios-safari/).

var syncWindowHeight = () => {

if (pageYOffset!=0) {

localStorage.pageScroll = Math.round(pageYOffset);

}

if (device.portrait()) {

document.querySelector('body').classList.add('disable-scroll');

document.querySelector('html').classList.add('disable-scroll');

} else {

document.querySelector('body').classList.remove('disable-scroll');

document.querySelector('html').classList.remove('disable-scroll');

}

setTimeout(()=>{

document.documentElement.style.setProperty(

"--window-inner-height",

`${window.innerHeight}px`

);

}, 100)

}

We invoke a function on modal window open and add listener on window resize

syncWindowHeight();

window.addEventListener("resize", syncWindowHeight);

We delete resize listener on modal window close

window.removeEventListener("resize", syncWindowHeight);

window.scrollTo(0, localStorage.pageScroll);

We also add css style to disable scroll

.disable-scroll {

height: calc(var(--window-inner-height) - 1px);

overflow: hidden;

}

We also recommend adding a check on device type and calling syncWindowHeight(); only for iphone if (device.iphone()) - you can use library [device.js](https://danilin.biz/device-js.htm).

# Close games:

2 types of events close, which occur at the close game

(For each game one of two types):

1. The page sends **postMessage** request, with request body **"closeGame", "close" or "GAME\_MODE:LOBBY"**

2. All other games: Game page redirected to **domain/close.php**, where **domain** - Website address, sent to **gameOpen**.

Recommendations and examples to handle all 2 types of game closing events in one function **closeGame**:

1. Processing example postMessage on javascript:

window.onmessage=function(event) {

if (event.data=='closeGame' || event.data=='close' || event.data=='notifyCloseContainer' || (event.data.indexOf && event.data.indexOf("GAME\_MODE:LOBBY")>=0)) {

window.gclose();

} else if (event.data=='closeWheel') {

\_this.closeWheel();

}

if (event.data.closeGame!==undefined) {

\_this.closeGame(event.data.closeGame);

}

}

2. Example [close.php](https://api.gamesapi.biz/API/IntegrationFiles.zip) working when game open used **window.open** and when game in iframe

# Save current page location after close game

To persist the current page location, you can use the [beforeunload](https://developer.mozilla.org/ru/docs/Web/API/Window/beforeunload_event) event. Before the user leaves the current page, we save Y coordinates:

window.onunload = window.onbeforeunload = function () {

localStorage.pageScroll = Math.round(pageYOffset);

};

On page open, we scroll the page to saved position

window.scrollTo(0, localStorage.getItem('pageScroll'));

# 

# Add browser back button functionality

To add back button close functionality, you need to:

1) Listen [popstate](https://developer.mozilla.org/ru/docs/Web/API/Window/popstate_event) event

window.addEventListener('popstate', function (e) {

if (document.querySelector('body').classList.contains('game-open')) {

/\* close game function here \*/

}

})

2) When the game opens, we change the record in history by calling [history.pushState()](https://developer.mozilla.org/en-US/docs/Web/API/History_API#The_pushState()_method). In following example, we add new history record with gameId

history.pushState(null,null,window.location.search+'&id='+id);

3) For proper [popstate](https://developer.mozilla.org/ru/docs/Web/API/Window/popstate_event) event workflow it is required to remove iframe element from the page when game closed

document.getElementById('gameFrame').innerHTML = '';

# Hide open game links and games picture (for game lobby) with RewriteEngine

In open game replace link with domain from which games are launched, games will be loaded from your website **domain/resources**

**"cdnUrl"**:"http://domain.com/resources" – parameter in **openGame** to replace the link with your domain for the games that support **rewriterule**

Configure the site server from which the game opens, **linux**:

a2enmod rewrite headers proxy proxy\_http ssl status

Example **apache**:

Header set Access-Control-Allow-Origin "\*"

**RewriteEngine** on

ProxyPass /resources http://cdn.(ask support).net/resources

ProxyPassReverse /resourceshttp://cdn.(ask support).net/resources

Example **nginx**:

minimum configuration

server {

root /var/www/resources;

index index.html index.php;

server\_name resources.youdomain.com;

location /resources {

proxy\_pass http://cdn.(ask support).net/resources;

}

}

Exception for this type of openGame: rewriterule=0

# Create hall and Change settings through API:

host: http://api\_server/API/

Create hall

{

"cmd":"createHall",

"api\_key":"tbs2\_agent\_api\_key",

"agent":"tbs2\_agent\_Id",

"key":"hall\_key",

"host":"http://callback\_url/",

"hall":"",

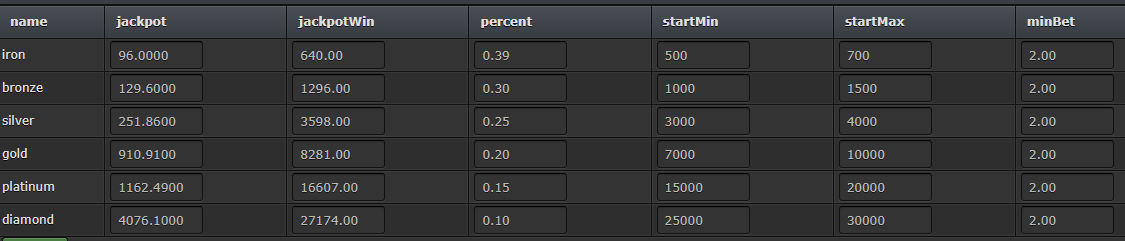
"login":id hall in you system,

"currency":"USD"

}

| **Parameter** | **Description** |
| --- | --- |
| api\_key | Key from your agent ask manager or support about it creation Its not available by default |
| agent | Your agent id **can be found in BO** or ask support or manager about it |
| key | Hall key **created by user**. Needed as additional checking. Its string can contain any value. |
| host | Endpoint to **user server** where processing commands |
| hall | Leave empty when creating a hall with command - **createHall** Fill with data when changing hall with - **changeHallConfig** |
| login | Using with command “createHall” to create your hall login It will be displayed in your BO when created |
| currency | Using with command “createHall” Currency should match your region usage and **can’t be changed after creation**. If you use several currencies you need to create several halls. |

# Jackpots



Name - jackpot type name

Jackpot - current jackpot value

JackpotWin - amount at which the jackpot is hit

Percent- percentage of the bet that increases the Jackpot

startMin - minimum jackpot value

startMax - maximum jackpot value

minBet - minimum bet that will increase the Jackpot value

JackpotWin is recommended to be set between startMin and startMax

EXAMPLE REQUEST TO SERVER

url="http://SERVER/API/

{

"hall":"HallId",

"key":"KeyHall",

"cmd":"jackpots"

}

SERVER - URL of your back office server

Receiving current jackpot values ​​is possible no more than once every 1 minute due to server load limitation

EXAMPLE RESPONSE FROM SERVER

{

"status": "",

"microtime": 0.003556966781616211,

"dateTime": "2025-01-01 00:00:00",

"error": "",

"content": [

{

"name": "iron",

"jackpot": "96.0000",

"user": "0",

"jackpotWin": "0.00",

"activate": false},

{

"name": "bronze",

"jackpot": "129.6000",

"user": "0",

"jackpotWin": "0.00",

"activate": false},

{

"name": "silver",

"jackpot": "251.8600",

"user": "0",

"jackpotWin": "0.00",

"activate": false},

{

"name": "gold",

"jackpot": "910.9100",

"user": "0",

"jackpotWin": "0.00",

"activate": false},

{

"name": "platinum",

"jackpot": "1162.4900",

"user": "0",

"jackpotWin": "0.00",

"activate": false},

{

"name": "diamond",

"jackpot": "4076.1000",

"user": "0",

"jackpotWin": "0.00",

"activate": false}

]

}

Name - name of the jackpot type

Jackpot - current jackpot value

User - last login that won the jackpot

JackpotWin - jackpot amount

For processing and displaying on the site, it is necessary to take into account 2 parameters

“name” - jackpot name and “jackpot” - current jackpot value.

# Games Logs

url="http://SERVER/API/";

{

| ERROR CODE | Description |
| --- | --- |
| session\_not\_found | Game session is not found |
| session\_closed | Game session is closed |
| writeBet:fail\_bet | Bet is greater than balance |
| writeBet:fail\_balance | Fail balance response from site |
| writeBet:mtwpu | Hall MAXIMUM MINUS setting error |
| writeBet:actionId\_exist | The action already exists |
| writeBet:prePayment | Prepayment error, insufficient credits |
| writeBet:fail\_response | Fail response from the site |

"cmd":"gameSessionsLog",

"hall":"hall\_id",

"key":"hall\_key",

"sessionsId":"session id(string)",

"count":" row count in page(int)",

"page":"page number(int)"

}

Example response:

"status": "success",

"error": "",

"content": {

"info": {

"page": 1, //page number

"pages": 1 //number of pages

},

"log": [{

"id": "3091109", //bet id

"status": "1", //status

"BetInfo": "|bet", //bet info

"gameId": "2", //game id in tbs2

"gameName": "Tornado", //game name in tbs2

"cdn": "https://resources/thumbnails/568",

//folder of images

"before": "447.38", // balance before bet

"bet": "0.20", // bet value

"win": "0.00", //win value

"dateTime": "2020-09-01 12:06:52", //bet datetime

"matrix": {...} //matrix Info }, ...]

Session logs are sent only by session id

Download and connect the files: **matrix.js** and **matrix.css**

Script in **matrix.js** works with jquery, if you don't have it connected to the site, you also need to connect it.

If necessary, you can change **matrix.css** to match the style of the site.

**matrix.js** should not be changed.

Parse response content and display response ["content"]["log"] array as table.

Then in js for each matrix (response["content"]["log"][i]["matrix"])

call matrixDraw(elem, matrix), once ["content"]["log"].

where:

elem - element

jquery - element, where the matrix will be placed,

matrix - bet matrix in json or string format.

You can download the files for working with the matrix, as well as other necessary files, from this link: <https://api.gamesapi.biz/API/IntegrationFiles.zip>

| **ERROR CODE** | **Description** |
| --- | --- |
| session\_not\_found | Session not found |
| session\_closed | Session closed |
| writeBet:fail\_bet | Higher balance rate |
| writeBet:fail\_balance | Partner balance error |
| writeBet:mtwpu | Error "Maximum minus hall settings" |
| writeBet:actionId\_exist | Event already processed |
| writeBet:prePayment | Prepayment error, no credits |
| writeBet:fail\_response | Incorrect partner response |

# All type response actions

These are all possible types of actions that can come in response from the game, they are needed to process the bet, namely to understand what event occurred in the game. Whether the player made a bet or received a win, for example.

Link to [documentation about actions](https://docs.google.com/document/d/1nbu2n7ZONMiJtyoK-z2Ak-0Z4vMCFdOWEoWApVcw1c0/edit?usp=drive_web&ouid=108756315214196403253).

* spin
* SpinNormal
* collect
* freeReSpin
* bet\_and\_wi
* SpinFree
* reSpin
* CollectWin
* freeSpin
* win
* bet
* gamble
* LuckyWheel
* bonus
* drop
* takeWin
* pick
* extra
* cancel
* loss
* PLAY
* FINISH\_GAM
* jackpot
* mystery
* LADDER\_PLA
* RISK\_DIVID
* PLAY\_FREE
* RISK\_BLACK
* place\_bet
* add\_wins
* half
* hit
* checkWin
* end
* stand
* insurance
* split
* refund
* freespins
* spin/risk
* jpSpin
* sale
* LOTTO
* diamonds
* xWin
* Picker
* double
* closeBonus
* Deal
* Exchange
* jackpot ma
* bonusactio
* open
* super jack
* spin/colle
* cashBack
* gamble\_sta
* BUY
* endbonus
* gamble\_tak
* rollback
* choice
* ""
* dice