iSOFTBET

SIMPLY PLAY

PURSE
ADVANCED INTEGRATION

v4.4

TABLE OF CONTENTS

DOCUMENT HISTORY

v4.4

- in section 2.4.3 Request format, in the **Standard JSON parameters** table, removed the **timestamp** parameter
- added the **timestamp** parameter to the **Request** tables for the following commands: 5.3 bet, 5.4 win, 5.5 cancel
- changed title of section 10.0 TEST USERS to 10.0 TEST CASES

v4.3

- in section 2.3 REALITY CHECK (RTS13), added the two options to meet RTS13 requirements

v4.2

- in section 2.4.3 Request format, in the **Standard JSON parameters** table, added the **multiplier** parameter
- removed the **multiplier** parameter from the **Request** tables for the following commands: 5.3 bet, 5.4 win, 5.5 cancel

v4.1

- in section 2.3 REALITY CHECK (RTS13), added an important note
- in section 5.7 dialog updated comments for the **buttons** parameter to include and important note, details about the action value for the History button

v4.0

- added section 2.3 REALITY CHECK (RTS13)
- in section 2.4.3 Request format, in the **Standard JSON parameters** table, added the **timestamp** parameter
- in section 4.0 Commands list, added entry for a single-state **dialog** command
- added the **multiplier** parameter in the **Successful response** tables for all commands in section 5.0 Command details
- added the **multiplier** parameter in the **Successful response** tables for all commands in section 5.0 Command details
- added the **multiplier** parameter in the **Request** tables for the following commands: 5.3 bet, 5.4 win, 5.5 cancel
- added section 5.7 dialog
- in section 7.0 Error Response Format, added new example (Example 3) and added the **action** and the **dialog** parameters

- in section 7.1 List of possible Error Codes, added error code **D_01**
- in section 9.3 Bet Command Diagram, updated step 7 and the diagram

v3.3

- in section 11.0 REPORTS REQUESTED FROM WP FOR TEST USERS, updated the **Successful Response** example and the **Response details** table

v3.2

- for POST REQUEST, in section 2.4.3 Request format, in table **Standard JSON parameters**:
  - updated the details for the **sessionid** parameter
  - updated the description of the **currency** parameter
  - added the **allow_open_rounds** parameter
- for RESPONSE, in section 2.5.2 Response format, added the following note:
  All error responses should be returned with a HTTP 200 status code.
- in section 4.0 Commands list, removed the **sessionid** parameter from the single state **balance** command **request** example
- in section 4.0 Commands list, added the **currency** parameter to all **response** examples, except for the single state **init** command
- in section 5.0 Command details, for all commands, except **init**, to the **Successful Response** table, added the **currency** parameter
- in section 5.5 cancel, updated the command's description
- in section 6.1 Reconciliation process, split the details for **single bet transaction** into: **First bet within the round** and **Not first bet within the round**
- in section 7.1 List of possible Error Codes, for error code B_07 and in section 9.3 Bet Command Diagram, for step 7, added the following important note:
  If request parameter allow_open_rounds = true then WP should allow multiple rounds to be open. Error B_07 should never be returned when allow_open_rounds = true.
- in section 9.5 Cancel Command Diagram, updated step 6
- in section 10.0 Test Users, for SECTION 3, removed the following user behaviours:
  - B_04
  - B_05
  - B_06
  - B_07
  - W_03
  - W_06
  - C_03
  - C_04

- C_05

v3.1

- added the **country** parameter to the **init** command and updated all examples accordingly

v3.0

- updated in section 1.0 Introduction
- in section Glossary, added new terms: Session based integration, Real time seamless integration
- in section 4.0 Commands list, in the commands table, for the **win** command, on the **Request/Response Example** column, changed the **jpw** value to: 50
- in section 5.6 end, updated the description of the **sessionstatus** parameter
- added section 12.0 Session based integration

v2.0

- in section 4.0 Commands list, in the commands table, for the **win** command, on the **Request/Response Example** column, updated **jpc** to **jpw**
- in section 7.0 Error Response Format, in **Example 1**, changed B_02 to B_03 and in **Example 2**, changed I_02 to I_03
- in section 7.1 List of possible Error Codes:
  - removed the following error codes:
    - **R_01**
    - **R_04**
    - **R_05**
    - **R_06**
    - **R_07**
    - **R_08**
    - **R_12**
    - **I_01**
    - **I_02**
    - **I_05**
    - **A_01**
    - **B_01**
    - **B_02**
    - **W_01**
    - **W_02**
    - **W_04**

    - **W_05**
    - **C_01**
    - **C_02**
    - **E_01**
    - **E_02**
  - added error code **R_13**
  - updated the descriptions for several error codes
- in section 9.1 Request Flow Diagram, updated diagram and flow steps
- in section 9.2 Init Command Diagram, updated diagram and flow steps
- in section 9.4 Win Command Diagram, updated diagram and flow steps
- in section 10.0 Test Users, updated sections list

v1.4

- change the name of section 10.0 LIST OF SPECIAL USERS to 10.0 TEST USERS
- change the name of section 11.0 REPORTS REQUESTED FROM WP FOR EACH SPECIAL USER to 11.0 REPORTS REQUESTED FROM WP FOR TEST USERS
- updated introduction text for section 10.0 TEST USERS
- in section 10.0 TEST USERS:
  - updated the **USER** column header to **USER BEHAVIOUR**
  - updated the **USER BEHAVIOUR** column for **SECTION 1** and **SECTION 4**
- in section 11.0 REPORTS REQUESTED FROM WP FOR TEST USERS, for the **date_to** parameter, on the **Description** column, added an important note:

  Important note: For reports generation, do not take into consideration the session end date.

v1.3.1

- in section 11.0 REPORTS REQUESTED FROM WP FOR EACH SPECIAL USER:
  - updated the names of the **datefrom** and **dateto** parameters to: **date_from** and **date_to**
  - updated the **Example of JSON object returned in the body** in the case of a **Successful Response**
  - in the **Response details** table, updated the name of the following parameters:
    - **skin_ids** to **skindids**
    - **session_errors** to **sessions_errors**
    - **session_closed** to **sessions_closed**

- **bets_cancelled_amount** to **bets_amount_cancelled**

v1.3

- changed **gameid** to **skinid** throughout the document and changed type of the **skinid** parameter to INT 32
- in section 2.4.3 Request format, updated the type of the **sessionid** parameter to STRING 48
- in section 10.0 LIST OF SPECIAL USERS, updated the GM API ACTIONS details for SECTION 3 for the following users:
  - B_04 (for test case 00_EC)
  - B_05 (for test case 00_EC)
  - W_04 (for test case 00_EC)
- updated section 11.0 REPORTS REQUESTED FROM WP FOR EACH SPECIAL USER

v1.2

- removed section **10.0 ACCOUNTING AND RECONCILIATION TESTING DETAILS**
- added section 10.0 LIST OF SPECIAL USERS
- added section 11.0 REPORTS REQUESTED FROM WP FOR EACH SPECIAL USER

v1.1

- updated section 7.1 List of possible Error Codes
- in section 4.0 Commands list, removed "**,**" from last lines of code examples and updated **end** command response example
- updated **Successful Response** parameters in section 5.6 end
- in section 9.4 Win Command Diagram, updated diagram and steps
- in section 9.5 Cancel Command Diagram, updated diagram and step **5**
- in section 9.8 Database structure proposal diagram, updated diagram

v1.0

- First version

GLOSSARY

Term	Definition
Game Platform (GP)	**iSoftBet** Platform. GMAPI within an **iSoftBet** game server that allows 3rd party Wallets and their Players to be integrated with **iSoftBet** games.
Licensee, Licensee Page	Licensee of **iSoftBet** games.
Wallet, Wallet Platform (WP)	Licensee (OR licensee's 3rd party) Players details provider. Keeps Player's Balance. Makes required transactions of stakes and wins.
Player	User of Licensee Page. Plays games.
Game Client (GC)	Game GUI. Could be Flash, HTML5...
Session, Game Session	Single gameplay of Player. Session Id is a unique string generated by GP every time a Player opens **iSoftBet** game.
Round	Part of Game Session. Should have at least one bet and its settlement (win or more than one win in case of Free Spins).
Request	Should be understood in a general way. Could have one (Single-State) or more (Multi-State) actions.
Response	JSON encoded answer from the Wallet.
Action	Single operation in the communication process between GP and WP. Some of them could be Transactions.
Command	Command defines the action to be made, can have parameters.
Transaction	Type of an Action identified by transaction id. Difference between Transaction and any other Action is that Transaction makes a change to the Player's balance state. For now the only Transactions in use are **bet** and **win**.
Single-State Request	Type of integration in which GP communicates separately with the WP with every Session's action.
Multi-State Request	Type of integration in which GP communicates with the WP by sending more than one command in one call. Multi-State is now only used for sending together **bet** and **win** (transaction from one Round in one Request).
Real time seamless integration	A type of integration where GP communicates with Wallet Platform during the session on every bet and win. The balance that is being presented to the player after a spin is a real time balance that comes from the Wallet's response.

Term	Definition
Session based integration	A type of integration where GP communicates with the Wallet Platform only at the beginning and the end of the session, and not during the session. At the beginning of the session, GP asks WP to block the player's balance and, at the end of the session, GP sends a session summary and asks WP to update the player's balance accordingly.

1.0 INTRODUCTION

There are two types of integrations described in this document:

- **Real time seamless integration**
- **Session based integration**

Before starting development, the type of integration that best fits the client's requirements is decided. Based on the agreed solution, the Wallet needs to be prepared accordingly.

It is possible for a player to have a seamless integration, so that the player's balance on the Wallet is reflected fully and live, in the game. However, depending of the jurisdiction, the Wallet needs to make sure that if needed, a player cannot connect on two games at the same time.

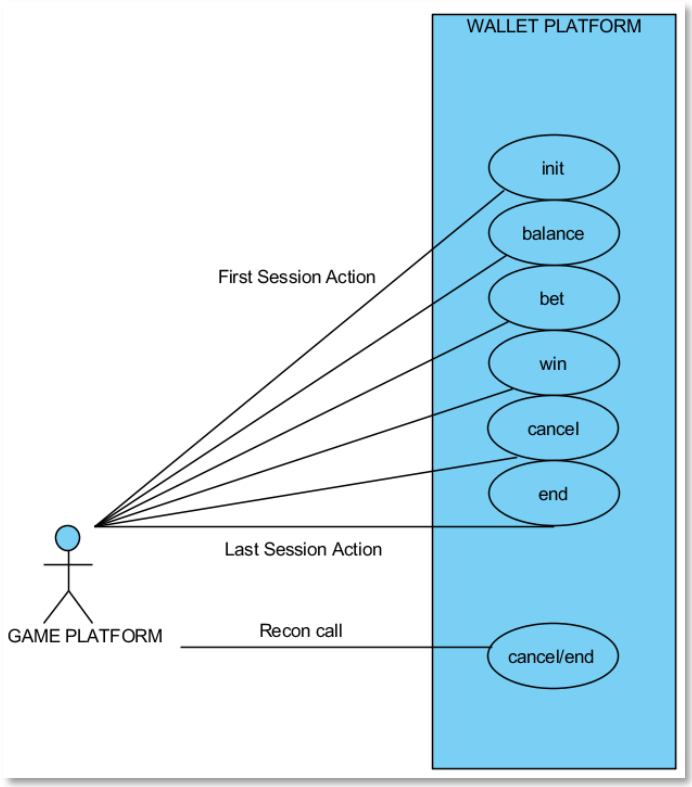
Some jurisdictions would prefer session based integrations, so the Wallet would need to be able to do it.

Note: For details about session based integration, please go to the following section of this document: 12.0 Session based integration.

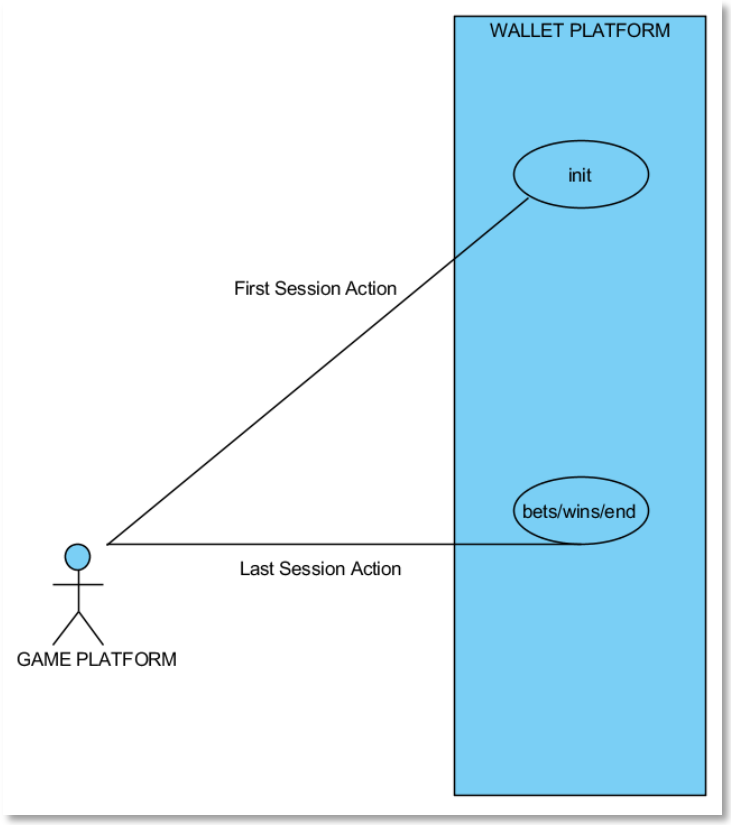
The web service created by the 3rd party has to ensure maximum speed while taking into consideration the resources and/or limitations of its database in order to make sure it can handle the calls from **iSoftBet** servers.

A REST web service allows the 3rd party to use any programming resources (PHP or Java for instance).

Real time seamless diagram:



Session based integration diagram:



2.0 API COMMUNICATION

2.1 Security

There are two levels of security. The Wallet Platform will be responsible for:

1. **Having an IP whitelisting system, to allow calls that come only from IPs of iSoftBet servers.**
2. **Validating every Request based on the HMAC hash sent using the POST method in the Request URL.**

2.2 Status types control & Objects hierarchy

The Wallet should store the following types of statuses for sessions, rounds and transactions:

SESSION:

- active (player is playing)
- inactive (session finished with success)
- error (session was interrupted with an error)

ROUND:

- active (before the last win within the round)
- inactive (round closed successfully)
- error (round was interrupted)

TRANSACTION:

- ok (successful transaction)
- cancelled (transaction has been cancelled)
- error (unsuccessful transaction)

Objects hierarchy

A Session includes a Round and a Round includes a Transaction.

Status types control is very important for security reasons. WP should not accept:

1. new transactions for inactive rounds
2. new rounds for inactive sessions
3. new rounds for a session that has active round

2.3 REALITY CHECK (RTS13)

„Customers should be able to set the frequency at which they receive and see on screen a reality check (display of time elapsed since the session began). The customer has to acknowledge the reality check for it to be removed from the screen.

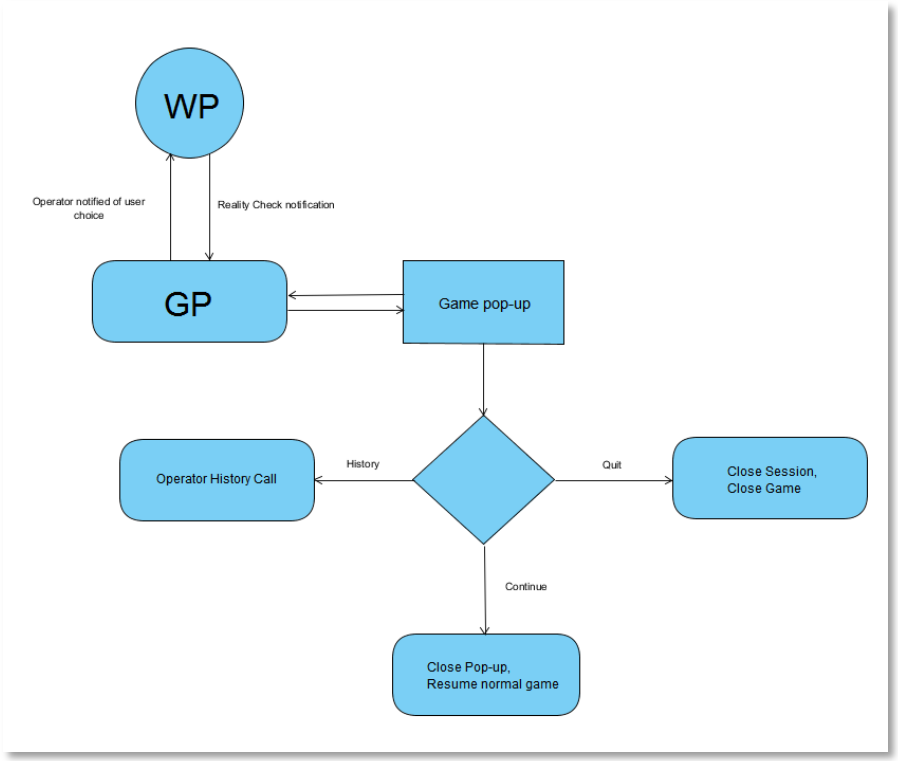
The customer should be able to set frequency prior to commencing the game, and the reality check should appear at that frequency until the session ends. The reality check should link to their account history and offer the facility to exit the gaming session."

There are two options to meet RTS13 requirements:

1. Use game clients (flash / html) to control session time based on the value of the parameter passed from the lobby. This option does not require interaction with WP.
2. WP can monitor sessions' time and send Reality Check pop-up content to GP (see section 7.0 Error Response Format).

Important:	**It should be possible to finalize rounds that are already opened and have a successful bet. WP should allow closing the rounds even if the session time is over. No popups related to the reality check should be returned in this case. Only new rounds can be controlled by RTS13 requirements.** **If a player is playing free rounds then the session time should not be validated by the WP. WP should accept transactions from free rounds even if the session time is over. No popups related to the reality check should be returned.**

High level Flow:



2.4 POST REQUEST

2.4.1 Request flow

FLOW Diagram: 9.1 Request Flow Diagram

2.4.2 Request headers

Content-Type: application/json; charset=utf-8

2.4.3 Request format

All Game Platform Requests will contain two parts:

1. **'hash' parameter sent using POST method in Request URL**, with Hash-based message authentication code (HMAC).

   Used algorithm: SHA256.
   Format: 64 char long hexadecimal string.

   Every implementation will have an individual secret key provided by **iSoftBet**. For hash implementation examples, please go to section 8.0 HMAC Generation Examples of this document.

2. **Raw JSON object sent in body by HTTP POST method.**

Standard JSON parameters:

Parameter	Required	Type	Description
sessionid	No	STRING 48	Session generated by GP ***Note**: This parameter is not required for **balance** request.*
playerid	Yes	STRING 64	Player Id passed in the launcher
currency	No	STRING 3	Currency returned from the Init action *Note: This parameter is not a part of the **init** request and it is optional for balance request, but will be sent for all the other requests to WP.*

Parameter	Required	Type	Description
skinid	Yes	INT 32	**iSoftBet** unique game id
allow_open_rounds	No	STRING 5	Possible values are "false" or "true". When this parameter is set up as **true**, then WP should allow placing of new bets even if there are open rounds in the current session. **Important:** **Error B_07 should never be returned when allow_open_rounds = true.**
state	Yes	STRING 16	Possible value: "single", "multi"
operator	No	STRING 250	Operator name passed in the launcher
action	Yes if STATE="single"	JSON OBJECT	JSON encoded action in format: `{ "command": "actionName", "parameters": { "param01": "value01", "param02": "value02" } }` Parameters aren't required for balance action.
actions	Yes if STATE="multi"	JSON ARRAY of OBJECTS	JSON encoded action in format: `[ { "command": "actionName", "parameters": { "param01": "value01", "param02": "value02" } }, { "command": "actionName2", "parameters": { "param2_01": "value2_01", "param2_02": "value2_02" } }]` Parameters are not required for balance action.

Parameter	Required	Type	Description
multiplier	No	INT 32	If this value is sent, it means that player is playing the game using tokens or a weak currency that is problematic to be displayed without denomination. All the bet/win transactions amounts should be multiplied with the multiplier value in order to obtain the bet/win amounts in the real currency. When this parameter is sent and has a value greater than **1** it means that GP expects WP to return player balance in tokens or after denomination.

2.4.4 Request examples

Example 1 – Single-State Request

URL:

```
WALLET_URL?hash=6ee986aa0e27906d5d3a3e68e95d57eb5e7647b76e6682383fe36b2d2912424a
```

Body content: JSON

```
{
   "sessionid": "3d7 … 19c1",
   "playerid": "player01",
   "currency": "EUR",
   "skinid": 1001,
   "state": "single",
   "operator": "operatorName",
   "action": {
       "command": "actionName",
       "parameters": {
              "param1": "value1",
              "param2": "value2"
       }
   }
}
```

Example 2 – Multi-State Request

URL:

```
WALLET_URL?hash=a96f619babfdd3c1cca2384c7728ec7a941a116726ef0797c6af04e5d44faeb7
```

Body content: raw JSON

```
{
   "sessionid": "3d7 … 19c1",
   "playerid": "player01",
   "currency": "EUR",
   "skinid": 1001,
   "state": "multi",
   "operator": "operatorName",
   "actions": [
              {
```

```
            "command": "actionName",
            "parameters": {
                  "param1": "value1",
                  "param2": "value2"
            }
        },
        {
            "command": "actionName2",
            "parameters": {
                  "param2_01": "value2_01",
                  "param2_02": "value2_02"
                  }
        }
   ]
}
```

2.5 RESPONSE

2.5.1 Response headers

Content-Type: application/json; charset=utf-8

2.5.2 Response format

application/JSON

Standard JSON parameters:

Parameter	Required	Type	Description
status	Yes	STRING 32	Should have value "success" for every correctly processed Request and "error" in case of some Error on the Wallet side.

Every Response is composed as follows:
status(success) **+ Action Response** (if there is more than one action, than Wallet should return response from the last action in the Request)
OR
status(error) **+ Error Response** (if something went wrong)

Note: All error responses should be returned with a **HTTP 200** status code.

2.5.3 Response examples

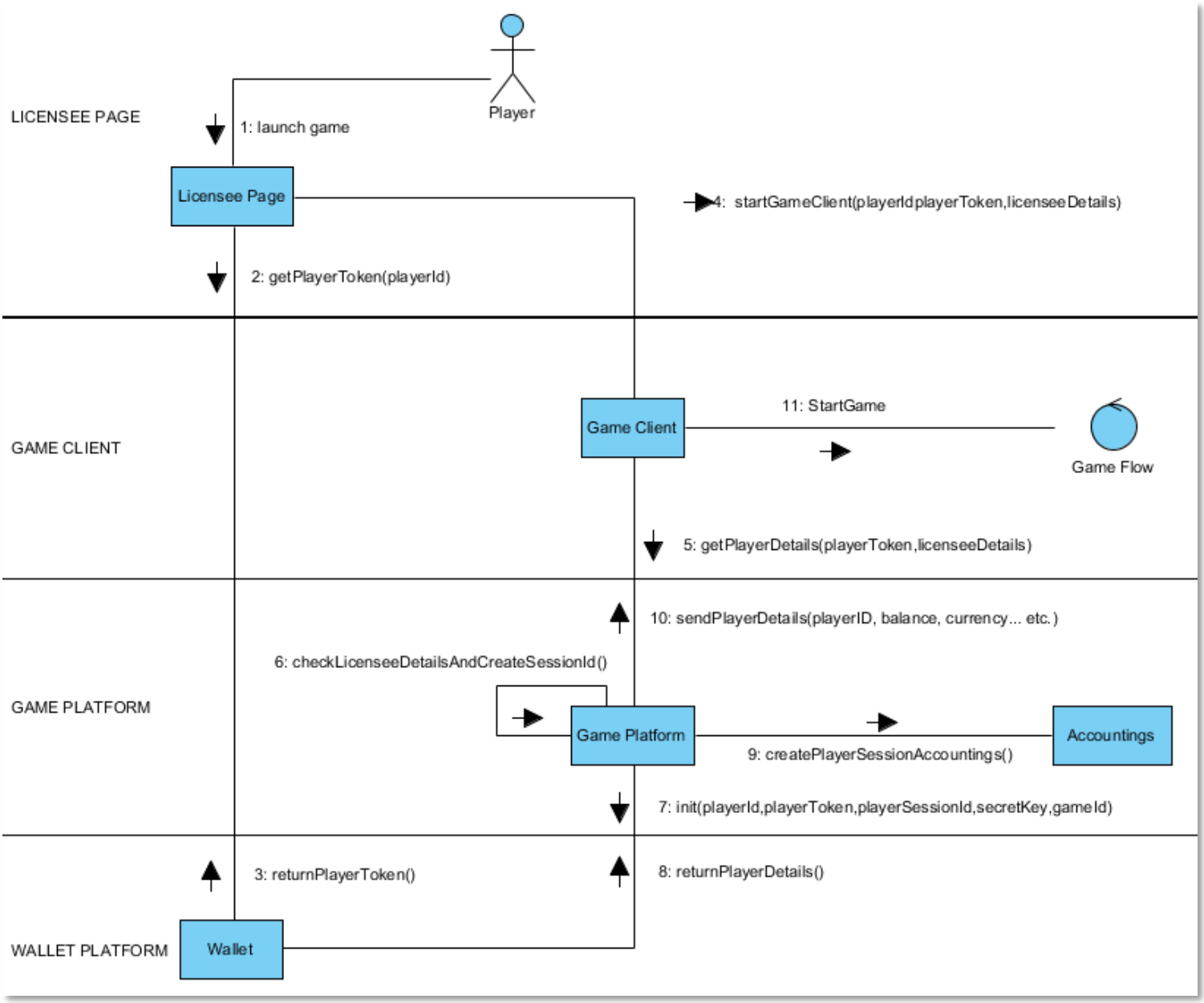
Example 1 successful Response of ***init*** Action

```
{
   "status":         "success",
   "sessionid":      "3d7 … 19c1",
   "playerid":       "00001",
   "currency":       "EUR",
   "balance":        100000
}
```

Example 2 Error Response

```
{
   "status":         "error",
   "code":           'ER01',
   "message":        "Some optional information for The Player or Error Info",
   "action":         "continue", //optional
   "display":        true //optional
}
```

3.0 GAME START COMMUNICATION PROCESS



3.1 Token retrieving process walkthrough

1. The player launches the game.
2. Licensee Page asks Wallet for player's token.
3. Wallets returns token.
4. Token is passed to Game Client as launch parameter.
5. The Game Client communicates with the Game Platform to retrieve the Player's details (player's balance, currency, username etc.).
6. The Game Platform checks if Licensee details are ok, then creates new Session Id to be used and passes it to the Wallet side.
7. If step **6.** is ok, GP sends ***init*** call to WP.
8. The Wallet Platform validates the token, creates Session on its side, and returns player's details.
9. The Game Platform creates new Session in GP Accountings.
10. GP communicates with Game Client to pass player's details.
11. Game starts.

4.0 COMMANDS LIST

Note: Communication in this table is as minimal as possible and the following are examples only; see each command details for additional parameters.

State	API Command	Description	Request/Response Example

single - state	**init**	First command of the Player's Session. Returns details needed for the next calls, like: playerid, currency, balance etc.	**Request:**
			{
			"sessionid": "3d7 … 19c1",
			"playerid": "player01",
			"skinid": 1001,
			"state": "single",
			"action": {
			"command": "init",
			"parameters": {
			"token": "2d7 … 19c1"
			"country": "GB"
			}
			}
			}
			Response:
			{
			"status":"success",
			"sessionid":"3d7 … 19c1",
			"playerid":"00001",
			"currency":"EUR",
			"balance": 100000
			}

single - state	**balance**	Gets current Player's balance.	**Request:**
			{
			"playerid": "player01",
			"currency": "EUR",
			"skinid": 1001,
			"state": "single",
			"action": {
			"command": "balance"
			}
			}
			Response:
			{
			"status": "success",
			"balance": 10000,
			"currency": "EUR"
			}

<table>
<tr><th>State</th><th>API Command</th><th>Description</th><th>Request/Response Example</th></tr>
<tr><td>single - state</td><td>bet</td><td>Called when the Player makes a bet. Request amount value should be subtracted from Player's balance. Response should contain balance after successful operation.</td><td>
Request:
<pre>{
   "sessionid": "3d7 … 19c1",
   "playerid": "player01",
   "currency": "EUR",
   "skinid": 1001,
   "state": "single",
   "action": {
        "command": "bet",
        "parameters": {
        "transactionid": "2d7 … 19c1",
        "roundid": "2d7 … 19c1",
        "amount": 50,
        "jpc": 199999.9999
        }
   }
}</pre>
Response:
<pre>{
   "status   "success",
   "balance   9990,
   "currency": "EUR"
}</pre>
</td></tr>
<tr><td>single - state</td><td>win</td><td>Called when the Player's bet is settled. Request amount value should be added to the Player's balance. Response should contain balance after successful operation.</td><td>
Request:
<pre>{
   "sessionid": "3d7 … 19c1",
   "playerid": "player01",
   "currency": "EUR",
   "skinid": 1001,
   "state": "single",
   "action": {
        "command": "win",
         "parameters": {
         "transactionid": "2d7..c1",
         "roundid": "2d7 … 19c1",
         "amount": 50,
         "jpw": 50
        }
   }
}</pre>
Response:
<pre>{
   "status":   "success",
   "balance": 10000,
   "currency": "EUR"
}</pre>
</td></tr>
</table>

<table>
<tr><th>State</th><th>API Command</th><th>Description</th><th>Request/Response Example</th></tr>
<tr><td>multi - state</td><td>bet + win</td><td>A Multi-State request is composed of all the transactions within one round (bets + wins).

A Multi-State request should be made sequentially, in order on the list and always have Response of the last Command.

A Multi-State Request should be made atomic as well. If one of the commands will fail, that means that all the previous ones shouldn’t make any effect on Player’s balance as well.</td><td>Request:
<pre>{
   "sessionid": "3d7 … 19c1",
   "playerid": "player01",
   "currency": "EUR",
   "skinid": 1001,
   "state": "multi",
   "actions":
        [
                {
                "command": "bet",
                "parameters": {
                  "transactionid": "2d7..c1",
                  "roundid": "2d7 … 19c1",
                  "amount": 50,
                  "jpc": 199999.9999

                  }
                }, {
                "command": "win",
                "parameters": {
                  "transactionid": "2d7 …c1",
                  "roundid": "2d7 … 19c1",
                  "amount": 50,
                  "jpw": 199999.9999,
                  "closeround": true
                  }
                 }
                }

        ]

}</pre>Response:
<pre>{
  "status":   "success",
  "balance":   10000,
  "currency": "EUR"
}</pre></td></tr>
<tr><td>single - state</td><td>cancel</td><td>Cancel is called if GP has unknown status of bet on its side because of a timeout or GP needs to cancel successful transaction because of the internal GP error.</td><td>Request:
<pre>{
   "sessionid": "3d7 … 19c1",
   "playerid": "player01",
   "currency": "EUR",
   "skinid": 1001,
   "state": "single",
   "action": {
        "command": "cancel",
        "parameters": {
          "transactionid": "2d7 … 19c1",
          "roundid": "2d7 … 19c1",
          "amount": 50,
          "jpc": 199999.9999
        }
   }
}</pre>Response:
<pre>{
   "status":   "success",
   "balance": 9990,
   "currency": "EUR"
}</pre></td></tr>
</table>

<table>
<tr><th>State</th><th>API Command</th><th>Description</th><th>Request/Response Example</th></tr>
<tr><td>single - state</td><td>end</td><td>Last call within the session.
WP should change Session status to the given by GP.</td><td>Request:
<pre>{
   "sessionid": "3d7 … 19c1",
   "playerid": "player01",
   "currency": "EUR",
   "skinid": 1001,
   "state": "single",
   "action": {
        "command": "end",
        "parameters": {
        "sessionstatus":"INACTIVE"
        }
   }
}</pre>Response:
<pre>{
   "status":   "success",
   "balance": 10000
   "currency": "EUR"
}</pre></td></tr>
<tr><td>single - state</td><td>dialog</td><td>Method is being used to inform WP about a player’s decision chosen within the popup.</td><td>Request:
<pre>{
   "sessionid": "3d7 … 19c1",
   "playerid": "player01",
   "currency": "EUR",
   "skinid": 1001,
   "state": "single",
   "action": {
        "command": "dialog",
        "parameters": {
           "action":"continue",
          "choiceIndex":2
        }
   }
}</pre>Response:
<pre>{
   "status":   "success",
   "balance": 10000
   "currency": "EUR"
}</pre></td></tr>
</table>

5.0 COMMAND DETAILS

5.1 init

This is the first call of every session. **Requires token** from the Game Start Communication Process. Required parameters are mandatory to start Game Client.

FLOW Diagram: 9.2 Init Command Diagram

Action parameters:

Parameter	Required	Type	Description
token	Yes	STRING 32	Player Token given in the launcher
country	Yes	STRING 2	ISO 3166-1 alpha-2. Country form where the player has opened the game.
game_type	No	STRING 32	Optional parameter describing the type of the game; E.g. Flash, HTML

Successful Response:

Parameter	Required	Type	Description
playerid	Yes	STRING 64	Player Identifier
sessionid	Yes	STRING 32	Session Identifier given in general Request part
currency	Yes	STRING 3	**ISO 4217** currency code
balance	Yes	INT 32	Player's balance in cents
multiplier	No	INT 32	This value should be sent when the balance amount must be multiplied with the multiplier value in order to obtain the real balance in the currency.

Error Response:

See section 7.0 Error Response Format

5.2 balance

Gets current Player's balance.

Action parameters:

NONE

Successful Response:

Parameter	Required	Type	Description
balance	Yes	INT 32	Player's balance in cents
currency	No	STRING 3	ISO 4217 currency code
multiplier	No	INT 32	This value should be sent when the balance amount must be multiplied with the multiplier value in order to obtain the real balance in the currency.

Error Response:

See section 7.0 Error Response Format

5.3 bet

Called when the Player makes a bet. Request amount value should be subtracted from the Player's balance. Response should contain balance after successful operation.

FLOW Diagram: 9.3 Bet Command Diagram

Request:

Parameter	Required	Type	Description
transactionid	Yes	STRING 32	Identifies every transaction that is changing the Player's balance ***Note: If the Wallet receives the same transaction more than once, should return the same response without perform it again.***
roundid	Yes	STRING 48	Id used to identify game round
amount	Yes	INT 32	Stake value in coins for bet. Should be subtracted from Player's balance.

Parameter	Required	Type	Description
jpc	Yes	DOUBLE 30,10	Jackpot contribution, with 10 decimals, in cents
froundid	No	INT 32	When *froundid* > 0 means that the round is free for the Player and **WP should not deduct money from the Player balance**. *froundid* is an identification of the Free Round package to be used in the current session. WP should mark all the bets and wins from free rounds in WP accounting. ***It is the WP responsibility not to consider or record any jackpot information if the free round is in use.***
fround_coin_value	No	INT 32	Coin value used by the Player for the current bet (in cents). This value will be sent only for free rounds. If WP wants to store detailed information about free rounds it should save this value in WP database.
fround_lines	No	INT 32	Number of lines used by the Player for the current bet. This value will be sent only for free rounds. If WP wants to store detailed information about free rounds it should save this value in WP database.
fround_line_bet	No	INT 32	Bet per line used. This value will be sent only for free rounds. If WP wants to store detailed information about free rounds it should save this value in WP database.
timestamp	No	INT 32	Exact time of the transaction being sent from GP to WP.

Successful Response:

Parameter	Required	Type	Description
balance	Yes	INT 32	Player's balance in cents.
currency	No	STRING 3	ISO 4217 currency code
multiplier	No	INT 32	This value should be sent when the balance amount must be multiplied with the multiplier value in order to obtain the real balance in the currency.

Error Response:

See section 7.0 Error Response Format

5.4 win

Called when the Player's bet is settled. Request amount value should be added to the Player's balance. Response should contain the balance after successful operation.

FLOW Diagram: 9.4 Win Command Diagram

Request:

Parameter	Required	Type	Description
transactionid	Yes	STRING 32	Identifies every transaction that is changing the Player's balance ***Note: If the Wallet receives the same transaction more than once, should return the same response without perform it again.***
roundid	Yes	STRING 48	Id used to identify game round
closeround	Yes	BOOLEAN	**true** or **false**; Determines if it was a last win on current round
amount	Yes	INT 32	Win value in coins. Should be added to the Player's balance
jpw	Yes	DOUBLE 30,10	Jackpot winning, with 10 decimals, in cents; the jackpot winning is already included in the win value
froundid		INT 32	When *froundid* > 0 means that the round is free for the Player and bet for this win was free. *froundid* is an identification of the Free Round package that is used in the current session. WP should mark all the bets and wins from free rounds in the WP accounting.
timestamp	No	INT 32	Exact time of the transaction being sent from GP to WP.

Successful Response:

Parameter	Required	Type	Description
balance	Yes	INT 32	Player's balance in cents.
currency	No	STRING 3	ISO 4217 currency code
multiplier	No	INT 32	This value should be sent when the balance amount must be multiplied with the multiplier value in order to obtain the real balance in the currency.

Error Response:

See section 7.0 Error Response Format

5.5 cancel

cancel is called if GP has unknown status of bet on its side because of a timeout or GP needs to cancel a successful transaction because of an internal GP error.

If GP calls WP with the "cancel" method for a single **bet** transaction, then WP should refund money to a Player's balance if the bet has been processed successfully on WP side in the past.

WP should return successful response from the **cancel** method even if the transaction has been cancelled in the past and doesn't need to be cancelled again.

GP will never call the "cancel" method for a single win transaction and for multi-state transactions.

GP will also never call the "cancel" method for a single bet transaction if it is not the first bet within the round.

Important: **WP should close the round after receiving the *cancel* call.**

FLOW Diagram: 9.5 Cancel Command Diagram

Request:

Parameter	Required	Type	Description
transactionid	Yes	STRING 32	Identifier of cancelled Transaction ***Note: It is the Wallet's responsibility to not allow cancellation of a bet of closed round.***
roundid	Yes	STRING 48	Must be the same as in cancelled bet
amount	Yes	INT 32	Must be the same as in cancelled bet
jpc	Yes	DOUBLE 30,10	Must be the same as in cancelled bet
froundid	No	INT 32	Must be the same as in cancelled bet
fround_coin_value	No	INT 32	Must be the same as in cancelled bet
fround_lines	No	INT 32	Must be the same as in cancelled bet
fround_line_bet	No	INT 32	Must be the same as in cancelled bet
timestamp	No	INT 32	Exact time of the transaction being sent from GP to WP.

Successful Response:

Parameter	Required	Type	Description
balance	Yes	INT 32	Player's balance in cents.
currency	No	STRING 3	ISO 4217 currency code
multiplier	No	INT 32	This value should be sent when the balance amount must be multiplied with the multiplier value in order to obtain the real balance in the currency.

Error Response:

See section 7.0 Error Response Format

5.6 end

Last call within the session. WP should change Session status to the given by GP.

FLOW Diagram: 9.6 End Command Diagram

Request:

Parameter	Required	Type	Description
sessionstatus	Yes	STRING 32	Possible values: **INACTIVE** or **ERROR**. If status is INACTIVE, it means that the session was closed properly and WP should change the session status to INACTIVE on its side. If status is ERROR, it means that there was some interruption during the session and WP should change session status to ERROR on its side.

Successful Response:

Parameter	Required	Type	Description
balance	Yes	INT 32	Player's balance, in cents.
currency	No	STRING 3	ISO 4217 currency code
multiplier	No	INT 32	This value should be sent when the balance amount must be multiplied with the multiplier value in order to obtain the real balance in the currency.

Error Response:

See section 7.0 Error Response Format

5.7 dialog

This method is used to inform WP about a player's choice within the popup.

Request:

Parameter	Required	Type	Description
action	Yes	STRING 32	Possible values: **continue** or **void**.
choiceIndex	NO	INT	Index of a choice selected by the player. 0 is a first element in the array.

Successful Response:

Parameter	Required	Type	Description
balance	Yes	INT 32	Player's balance, in cents.
currency	No	STRING 3	ISO 4217 currency code
multiplier	No	INT 32	This value should be send when the balance amount must be multiplied with the multiplier value in order to obtain the real balance in the currency.

Error Response:

See section 7.0 Error Response Format

6.0 RECONCILIATION

6.1 Reconciliation process

If a request times out because of internet connection problems, then GP will follow a process to reconcile the action:

a. **single bet transaction**

First bet within the round:

The bet request will be retried 3 (three) times with an interval of 10 seconds between each try. If a request is still not processed after the third try, GP will call the **cancel** method to cancel this single bet on WP side. If the **cancel** method times out then the **cancel** call is added to a Recon queue and an error message is displayed to the player on the game interface. The game client is closed. The reconciliation mechanism kicks in from there.

Not first bet within the round:

If not a first bet within the round, then the bet request will be retried 3 (three) times with an interval of 10 seconds between each try. If a request is still not processed after the third try then an error message is displayed to the player on the game interface. The game client is closed. When the player will open the same game again, the same single bet call will be sent to WP. Same scenario will continue until a successful WP response will not reach GP.

b. **single win transaction**

The win request will be retried 3 (three) times with an interval of 10 seconds between each try. If a request is still not processed after the third try then an error message is displayed to the player on the game interface. The game client is closed. When the player will open the same game again, the same single win call will be sent to WP. Same scenario will continue until a successful WP response will not reach GP.

c. **multi state call (all the transactions from one round in one call)**

The multi-state request will be retried 3 (three) times with an interval of 10 seconds between each try. If a request is still not processed after the third try then an error message is displayed to the player on the game interface. The game client is closed. When the player will open the same game again, the same multi state call will be sent to WP. Same scenario will continue until a successful WP response will not reach GP.

d. **end call**

The end request will be retried 3 (three) times with an interval of 10 seconds between each try. If a request is still not processed after the third try then the end call is added to a Recon queue. The reconciliation mechanism kicks in from there.

6.2 Reconciliation calls

6.2.1 Cancel

WP should accept "cancel" calls that are run independently of the game session in order to ensure the two systems are kept in sync.

Such a call can occur if GP has unknown response from the **cancel** call on its side because of a timeout (so it will try to run it in the background until the Wallet response from **cancel** will not reach GP).

WP should return successful response from the **cancel** method even if the transaction has been cancelled in the past and does not need to be cancelled again.

6.2.2 End

WP should accept "end" calls that are run independently of the game session in order to ensure the two systems are kept in sync. Such a call can occur only if GP has unknown status on its side because of a timeout on the **end** call, so it will try to run it in the background until the Wallet response will not reach GP.

6.3 Reconciliation mechanism details

The reconciliation mechanism ensures the two systems (GP and WP) are kept in sync.

From the GP perspective, the Recon queue is a table within the database, which records any "cancel" and "end" requests that timed-out.

GP has put in place an automated process (cron job) that reads the Recon table from the database and runs Recon requests within the queue.

For any new entry, a recon request will be sent as follow:

- Every minute for the first 10 minutes
- Every 10 minutes for the first 1 hour
- Every hour for the first 24 hours

After 24 hours, the requests that are still unsuccessful are marked for manual reconciliation.

7.0 ERROR RESPONSE FORMAT

An Error Response is returned in case of any failure on the Wallet side. In most cases the game will crash instantly (**Example 2**) but in some predicted cases it will be possible to inform the Player about a cause of an unsuccessful transaction by showing a popup message with content of "message" parameter (**Example 1**). **Example 3** shows the interaction with a player where that player can choose from the options and buttons given by WP.

Parameter	Required	Type	Description
code	Yes	STRING 4	Identification code of the Error
message	Yes	STRING 255	UTF-8 text about internal Wallet Errors/Notices to inform **iSoftBet** or to inform the Player
action	Yes	STRING 10	Possible values: - "**void**" - session cannot continue. - "**continue**" - session can continue after a player was informed about the error. For example, "insufficient funds" - "**dialog**" - interaction between a player and the wallet is needed in order to continue the current session. If the value of **action** is **dialog** then the value of the **display** element must be true and the value of the **dialog** element cannot be empty.
display	No	BOOLEAN	Not Required; by default ***false***, should be ***true*** in order to display info from 'message' parameter for the Player in the Game.
dialog	No	JSON object (string max 2000 chars)	Dialog element must be JSON composed of the elements you can see in the **dialog parameter elements** table.

dialog parameter elements:

Parameter	Required	Type	Description
choices	No	ARRAY	Interval extension choices (ordered array of values). Ex: ["30","60","120"]
default choice	No	INT	Default choice (index of default choice to select, 0 is first element).

buttons	Yes	JSON object	**Important:** **Max number of buttons is 3. If there are more than 3 buttons returned then the popup will not pass the validation and the game will return an error.**
			Buttons (as ordered array of objects with the following properties):
			● Display text
			● Action to execute as the following format: "action_type:action_param" where action type can be:
			○ "js" - in this case the code after: is executed as javascript code, for example:
			`"js:showPlayerHistory(2556)"`
			○ "url" - in this case the specified url will be opened in a new tab, for example:
			`"url:www.supercasino.com/history/2556"`
			Buttons **Continue**, **Quit** and **History** must always be present, and their labels are also configured through the buttons array:
			● the action value for the **Continue** button shall be 'continue'
			● the action value for the **Quit** button shall be 'quit'
			● the action value for the **History** button shall be 'history' or js/url to be executed; if the action value is 'history' then the game will use default history page from the lobby
			Example 1:
			`"buttons": [ { "text": "Continue", "action": "continue", },{ "text": "Quit", "action": "void", },{ "text": "History", "action":`

Parameter	Required	Type	Description

			"js:showHistory()",
			}
			]
			Example 2:
			buttons": [
			{
			"text": "Continue",
			"action": "continue",
			},{
			"text": "Quit",
			"action": "void",
			},{
			"text": "History",
			"action": "history",
			}
			]

Example1:

```
{
    "status":           "error",
    "code":             "B_03",
    "message":          "Insufficient Funds",
    "action":           "continue",
    "display":          true
}
```

Example2:

```
{
    "status":           "error",
    "code":             "I_03",
    "message":          "Invalid token",
}
```

Example3:

```
{
    "status":           "error",
    "code":             "D_01",
    "display":          true,
    "action" :          "dialog",
    "message":          "You have been playing for 3 hours",
```

```
    "dialog": {
           "choices":    ["30","60","120"],
           "defaultChoice":     0,
           "buttons":    [
                                 {
                                 "text": "Continue",
                                 "action": "continue",
                                 },{
                                 "text": "Quit",
                                 "action": "void",
                                 },{
                                 "text": "History",
                                 "action": "js:showHistory()",
                                 }
                           ]
                  }

}
```

7.1 List of possible Error Codes

Error Code	Type/Relation	Description
ER***nn***	General	Technical Error. Used in case of some Wallet general error and could not specify troublemaker. Could be used to define WP errors. ***nn*** should be 2 digit individual number code. E.g. ER01 - Server overload.
R_02	Request	Invalid request. This error can be returned if required parameters are missing or they have incorrect format.
R_03	Request	Invalid HMAC.
R_09	Request	Player not found.
R_10	Request	Game is not configured for this licensee.
R_11	Request	Invalid Session Id or (can occur only on **init** call) Session not unique
R_13	Request	Invalid Currency.
I_03	init	Invalid Token.
I_04	init	Token already used. [optional]
B_03	bet	Insufficient Funds.
B_04	bet	Duplicate Transaction Id.
B_05	bet	Transaction has been cancelled.
B_06	bet	Round is closed.
B_07	bet	There is an active Round in current Session. **Important:** **If request parameter *allow_open_rounds = true* then WP should allow multiple rounds to be open. Error B_07 should never be returned when *allow_open_rounds = true*.**
W_03	win	Invalid Round.
W_06	win	Duplicate Transaction Id.
W_07	Win	Operator should remain the same during the round. [only for licensees that use operators]
C_03	cancel	Invalid cancel, Transaction does not exist.

Error Code	Type/Relation	Description
C_04	cancel	Trying to cancel a bet from an already closed round.
C_05	cancel	Transaction details do not match.
D_01	dialog	This error means that the dialog with a player is needed in order to continue the session. This type of error returned from a BET method can be used for the reality check (RTS13).

8.0 HMAC GENERATION EXAMPLES

8.1 PHP

```
<?php
$key = 'verySecretKey';
$message = '{"message":"Hello World!"}';

print hash_hmac('SHA256',$message,$key);
```

8.2 Python

```
#!/usr/local/bin/python2.7
import hmac
import hashlib

secretKey = 'verySecretKey'
message = '{"message":"Hello World!"}'

print hmac.new(secretKey, msg=message, digestmod=hashlib.sha256).hexdigest()
```

8.3 Java

```
package javaapplication1;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class JavaApplication1 {

   final protected static char[] hexArray = "0123456789abcdef".toCharArray();

   public static void main(String[] args) throws Exception {
       String secretKey = "verySecretKey";
       String message = "{\"message\":\"Hello World!\"}";
       SecretKeySpec keySpec = new SecretKeySpec(secretKey.getBytes(),
"HmacSHA256");

       Mac mac = Mac.getInstance("HmacSHA256");
       mac.init(keySpec);
       byte[] rawHmac = mac.doFinal(message.getBytes());

       System.out.println(JavaApplication1.bytesToHex(rawHmac));
   }

   private static String bytesToHex(byte[] bytes) {
       char[] hexChars = new char[bytes.length * 2];
       for (int j = 0; j < bytes.length; j++) {
              int v = bytes[j] & 0xFF;
              hexChars[j * 2] = hexArray[v >>> 4];
              hexChars[j * 2 + 1] = hexArray[v & 0x0F];
       }
       return new String(hexChars);
   }
}
```

8.4 C#

```
using System;
using System.IO;
using System.Security.Cryptography;

class Program {

   static void Main() {
        string key = "verySecretKey";
        string message = "{\"message\":\"Hello World!\"}";
        System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
        byte[] keyByte = encoding.GetBytes(key);

        HMACSHA256 hmacmd5 = new HMACSHA256(keyByte);
        byte[] messageBytes = encoding.GetBytes(message);
         byte[] hashmessage = hmacmd5.ComputeHash(messageBytes);

        Console.WriteLine(ByteToString(hashmessage).ToLower());
   }

   static string ByteToString(byte[] buff) {
        string sbinary = "";

        for (int i = 0; i < buff.Length; i++) {
               sbinary += buff[i].ToString("X2"); // hex format
        }
        return (sbinary);
   }
}
```

8.5 Correct HMAC

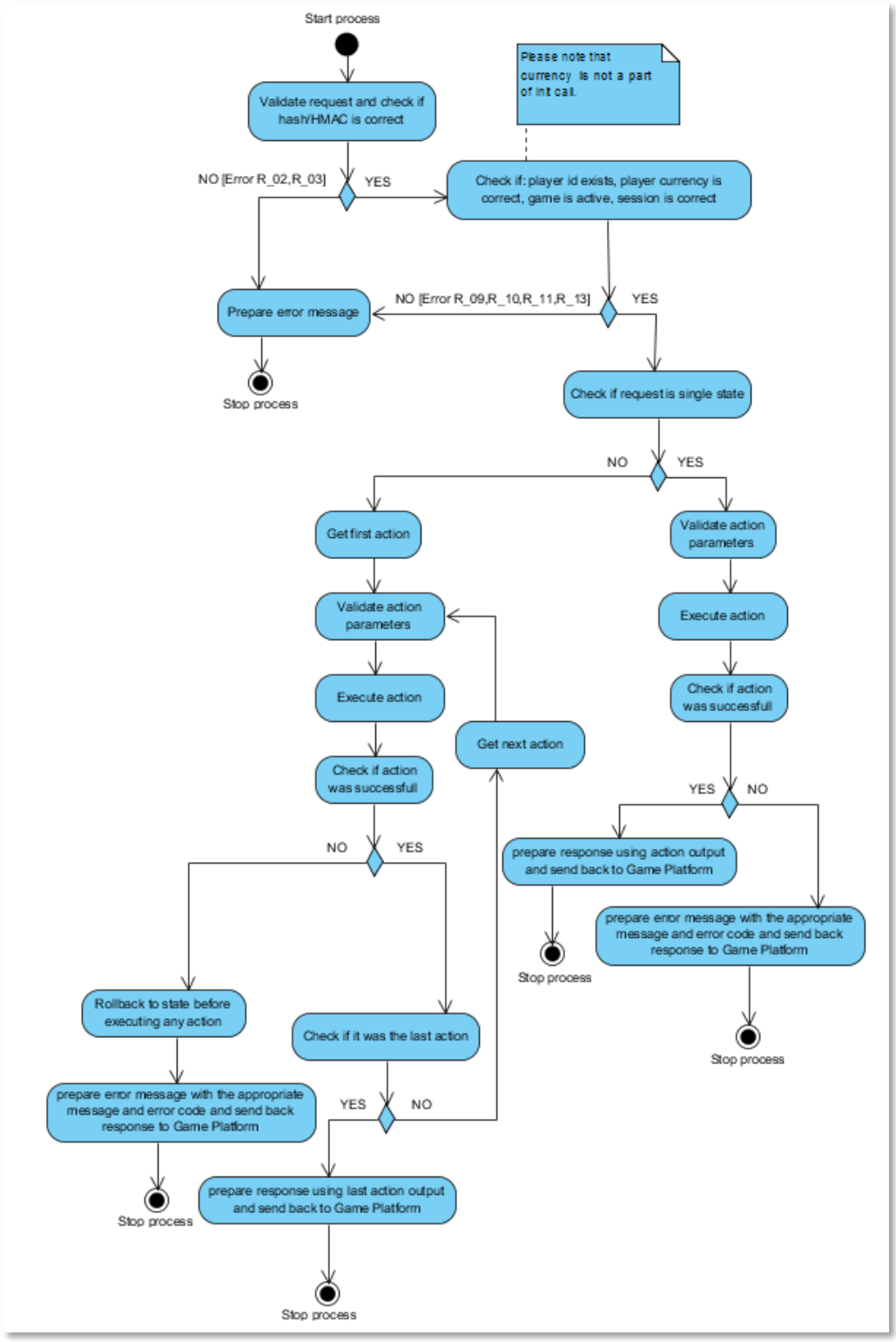
This hash string for given values:

- secretKey: verySecretKey
- message: {"message":"Hello World!"}

```
514acab8e3ff3f6b7562e8b6ef44141d21e90d7aa19002a38314ecc9fbce2553
```

9.0 DIAGRAMS

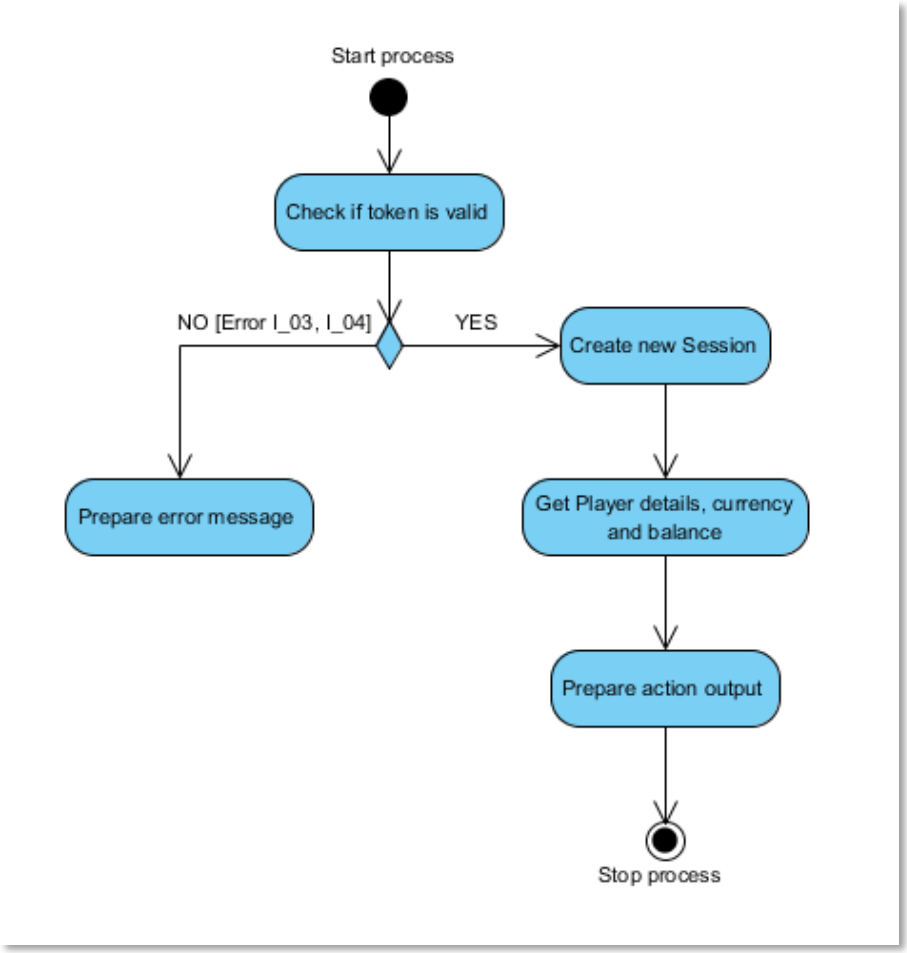
9.1 Request Flow Diagram



1. Check if hash is correct. If not - return the **Error R_03**.
2. Check if request has valid format. If not - return the **Error R_02**.
3. Check if Game for given skinid exists on Wallet Platform and is enabled. If not - return the **Error R_10**.
4. Check if Player id exists. If not - return the **Error R_09**.
5. Validate Session.
   If a first Action call of the Request is 'Init' then check if session has unique value. If not return the **Error R_11**.
   If a first Action call of the Request is not 'Init' then check if a session exists on WP side and has status different from 'INACTIVE'. If not - return the **Error R_11**.
6. Check if Player currency is correct. If not - return the **Error R_13**.
7. Execute action.
8. Check if execution was successful.
9. Response preparation:
   - **STATE 'single'**: Prepare and send successful or Error Response based on an action execution result.
   - **STATE 'multi'**: Error at any action should trigger immediate break of processing other actions. There also should be a rollback process to get back to the state before the request (or actions should take effect after processing all of them)

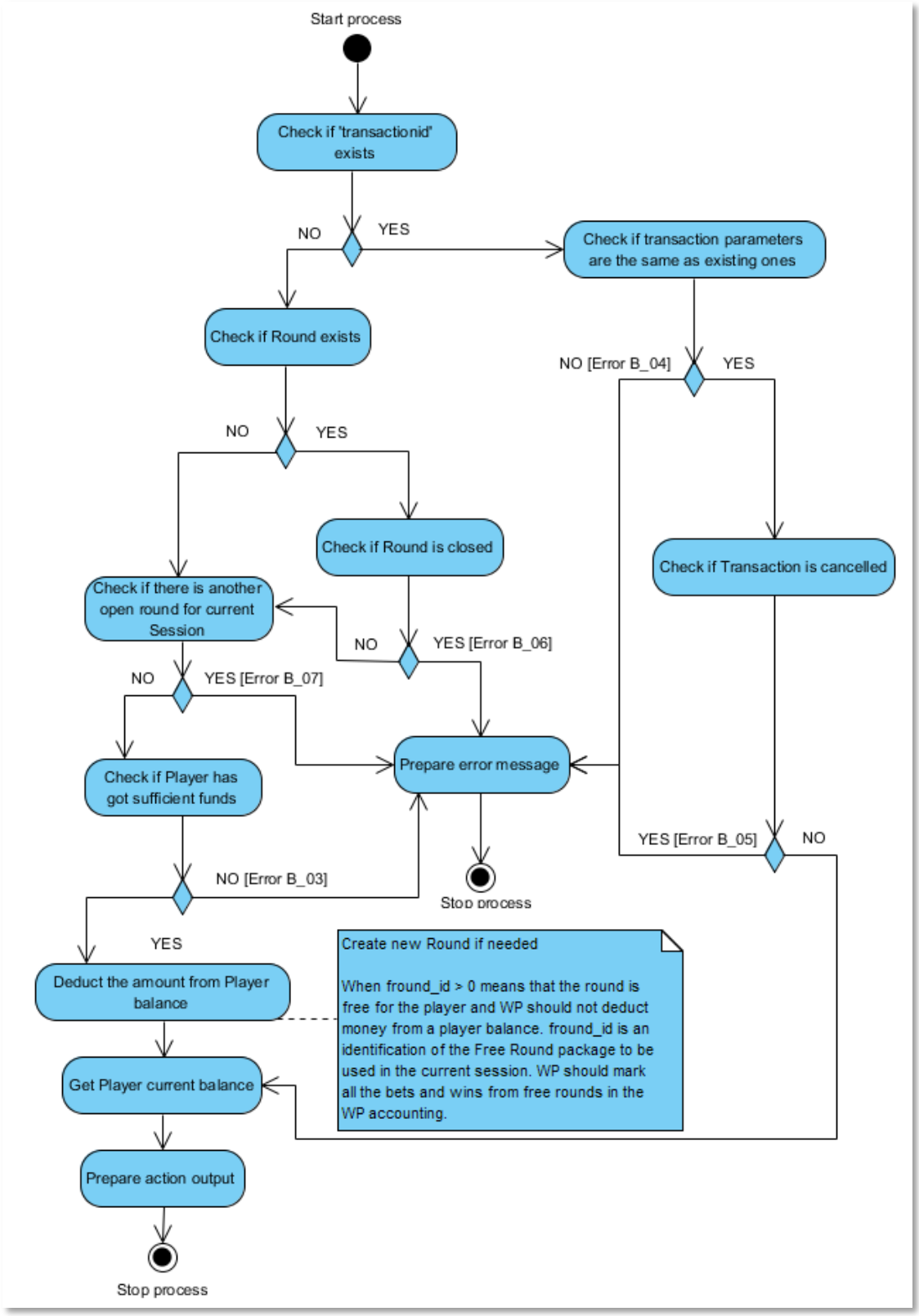
     If there are no Errors, successful Response should be prepared based on the last action message.

9.2 Init Command Diagram



1. Check if Token is valid for a Player. If no - return Error (codes: **I_03**, **I_04**).
   I_04 error is an optional one and can be used for integrations that expect the token passed in the launcher to be a unique value.
2. Create new Session.
3. Get Player details, currency and balance.
4. Prepare an output with gathered information.

9.3 Bet Command Diagram

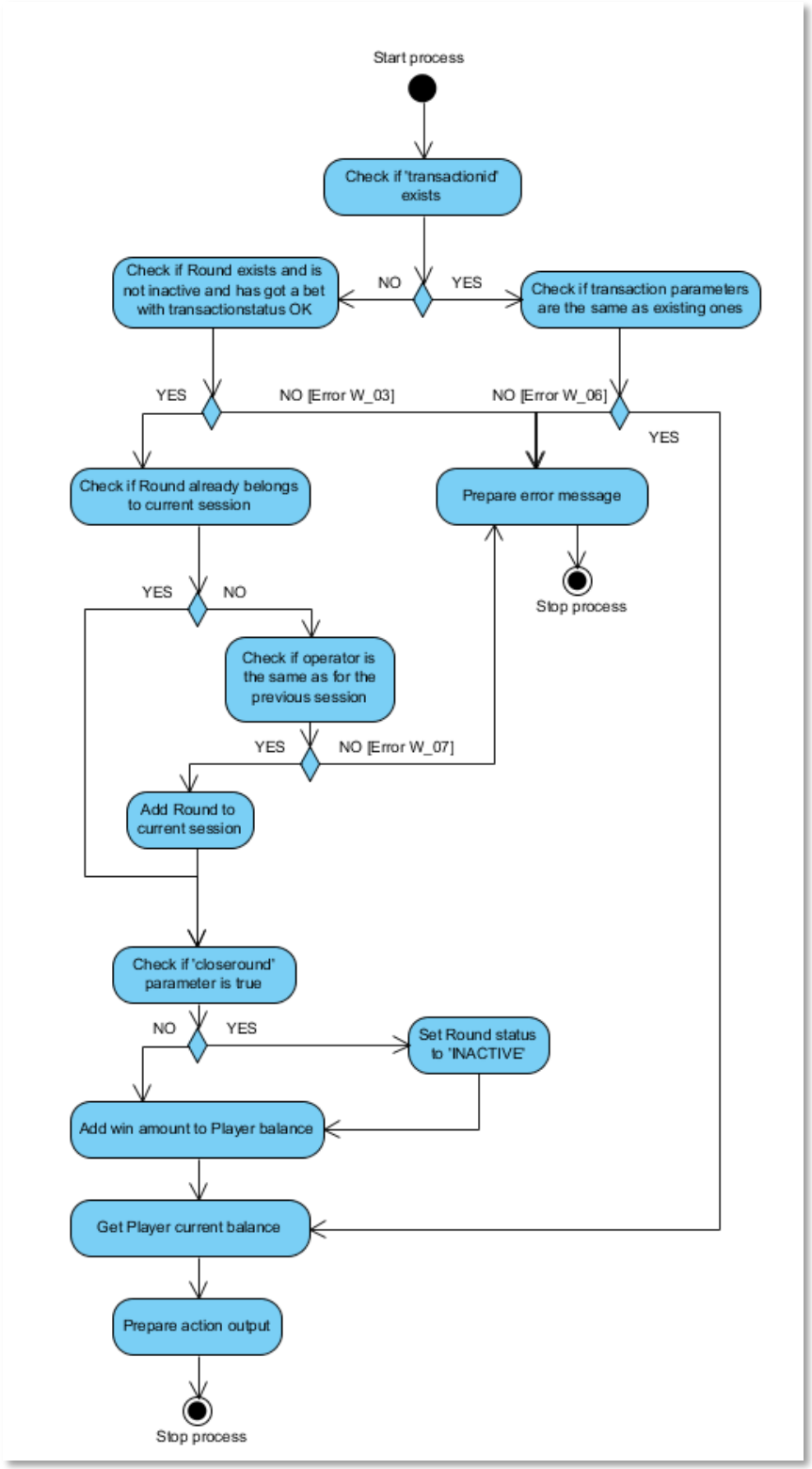


1. Check if 'transactionid' exists. If it exists go to step **2**; if it doesn't go to step **3**.
2. Check if Transaction parameters are the same in Request (see bet parameters) as they are in the system. If yes go to step **4**; if not go to step **5** with Error **B_04**.
3. Check if Round exists. If yes, go to step **6**; if not go to step **7**.
4. Check if Transaction is cancelled. If not go to step **10**; if yes go to step **5** with Error **B_05**.
5. Prepare proper error message and stop process.
6. Check if Round is closed. If not go to step **7**; if yes go to step **5** with Error **B_06**.
7. Check if there is another open Round for the current Session. If no, go to step **8**; if yes go to step **5** with Error **B_07**.

Important:	**If request parameter *allow_open_rounds = true* then WP should allow multiple rounds to be open. Error B_07 should never be returned when *allow_open_rounds = true*.**

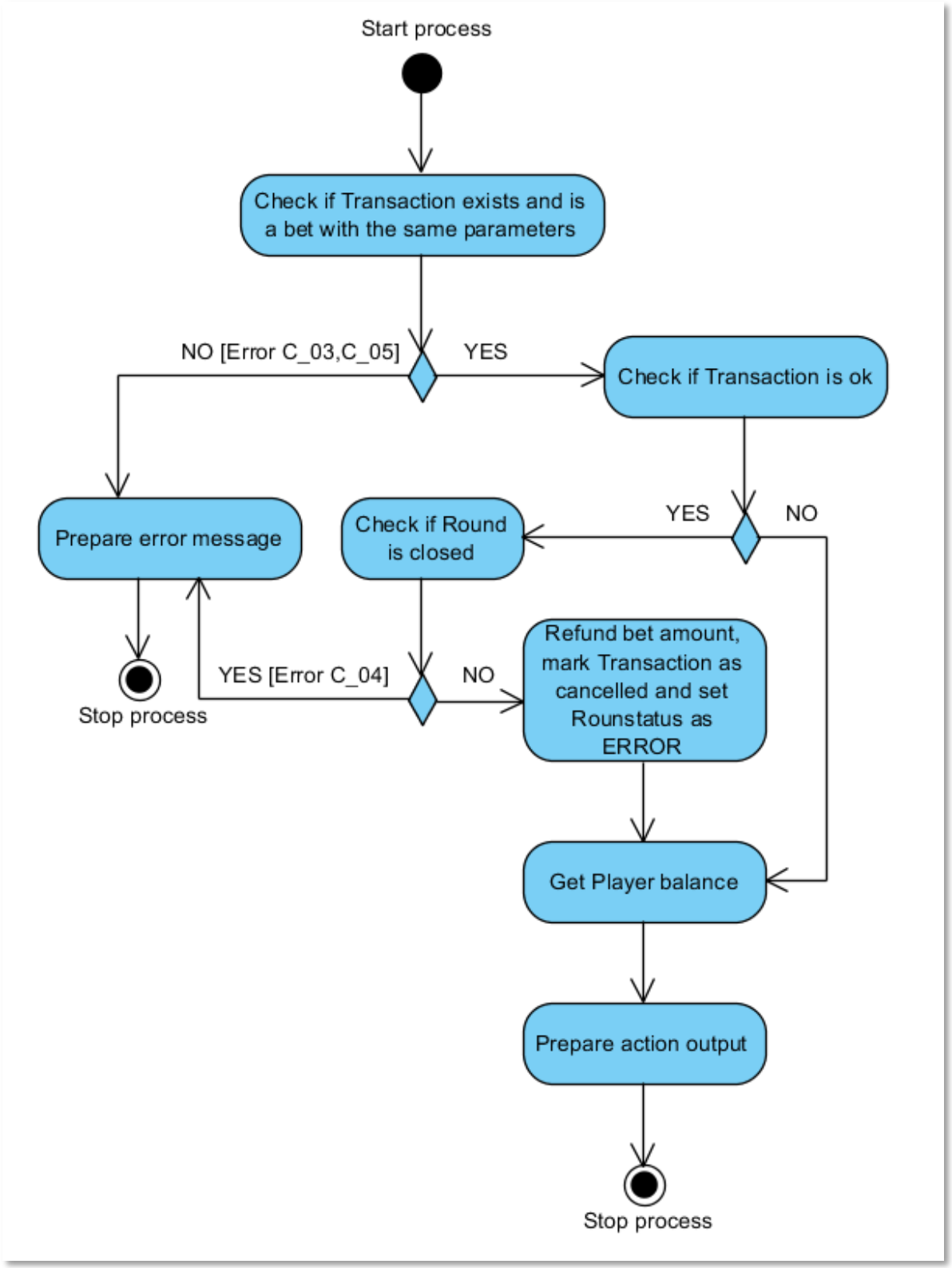
8. Check if Player has got sufficient funds to place a bet. If yes go to step **9**; if not go to step **5** with Error **B_03**.
9. Deduct the amount of the bet from the Player's balance. Go to step **10**.
10. Get the Player's current balance. Go to step **11**.
11. Prepare action output and stop process.

9.4 Win Command Diagram



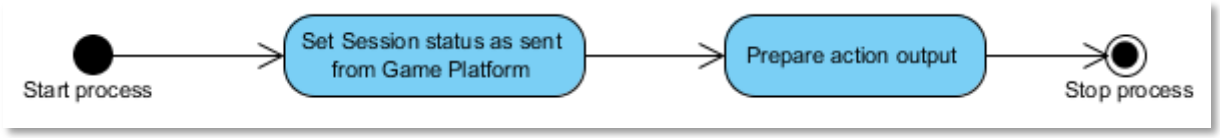
1. Check if 'transactionid' exists. If it exists, go to step **2**; if it doesn't go to step **3**.
2. Check if Transaction parameters are the same in the Request (see win parameters) as existing in the system. If yes go to step **11**, if not go to step **5** with Error **W_06**.
3. Check if Round exists and is active and has already a bet with transactionstatus OK. If yes go to step **4**, if not go to step **5** with Error **W_03**.
4. Check if Round already belongs to current Session. If yes, go to step **7**, if not, go to step **6**.
5. Prepare proper error message and stop process.
6. Check if operator is the same as for the previous session. If yes, go to step **7**, if not, go to step **5** with Error **W_07.**
7. Add Round to current session. Go to step **8**.
8. Check if 'closeround' parameter of action is true. If yes, go to step **9**, if not go to step **10**.
9. Set Round status to 'INACTIVE'. Go to step **10**.
10. Add win amount to Player balance. Go to step **11**.
11. Get Player current balance. Go to step **12**.
12. Prepare action output and stop process.

9.5 Cancel Command Diagram



1. Check if Transaction exists in the system and has a type 'bet' and has got exactly the same parameters. If yes go to step **2**, if not go to step **3** with Error **C_03** (if Transaction doesn't exists) or **C_05** (if details don't match).
2. Check if Transaction result is 'ok'. If yes go to step **4**, if not go to step **6**.
3. Prepare proper error message and stop process.
4. Check if Round is closed. If no, go to step **5**, if yes go to step **3** with Error **C_04**.
5. Refund bet amount to the Player's balance and mark Transaction as cancelled and Round as error. Go to step **6**.
6. Get the Player's current balance. Close the round. Go to step **7**.
7. Prepare action output and stop process.

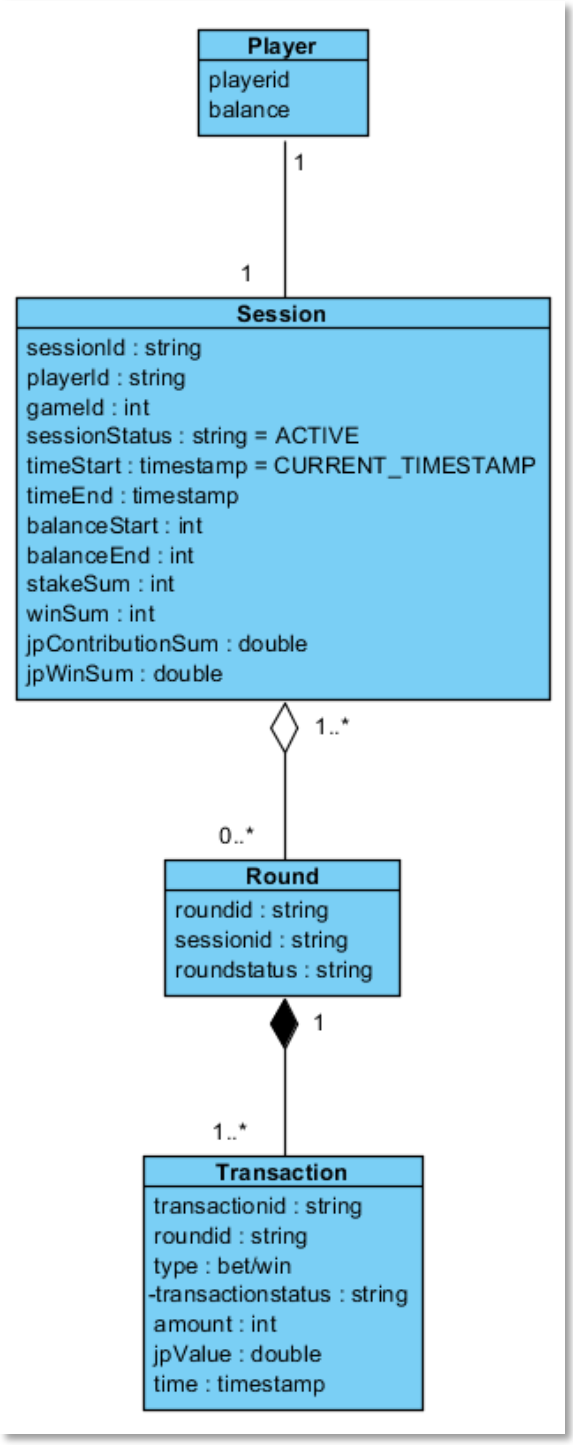
9.6 End Command Diagram



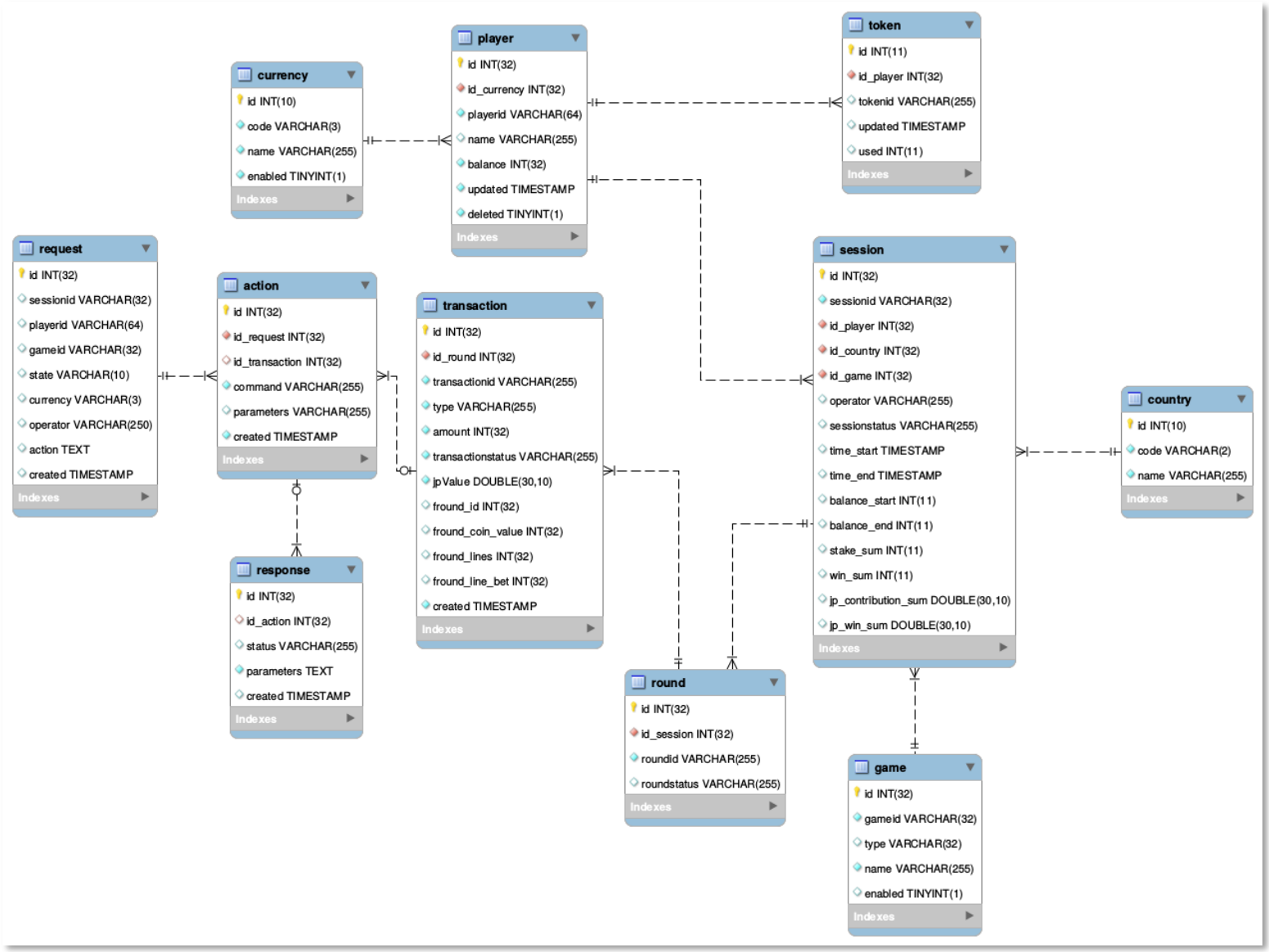
1. Set Session status to the one sent by the Game Platform in 'sessionstatus' parameter. For a session without any errors the status will be 'INACTIVE' and for a session that was interrupted somehow the status will be 'ERROR'.
2. Prepare action output and stop process.

9.7 Classes Dependency Diagram

This diagram should be treated as a demonstration of dependencies and required properties of each class/object on WP. Required fields are crucial for accounting comparison part of testing integration.



9.8 Database structure proposal diagram



10.0 TEST CASES

Test cases are divided into the following sections:

- **SECTION 1** – "happy flow"
- **SECTION 2 /SECTION 3** – test cases
- **SECTION 4** – HV automatic tests for testing the reconciliation mechanism

SECTION NUMBER	TEST CASES	Test Case Description
SECTION 1	happy flow	Game should not have errors while playing.
SECTION 2	10_DT_B (for test case 10_DT)	Player presses spin. GP sends bet to the Wallet with transaction id XXX and amount YYY. Wallet accepts the bet and returns a successful answer. Then GP simulates the same bet with the same transaction id XXX and the same amount YYY. Wallet should return the current player balance from the second bet attempt without deducting this second bet amount again.
SECTION 2	10_DT_W (for test case 10_DT)	Player presses spin. GP sends bet to the Wallet and Wallet accepts the bet and returns a successful answer. GP sends win to the Wallet with transaction id XXX and amount YYY. Wallet accepts the win and returns a successful answer. Then GP simulates the same win with the same transaction id XXX and the same amount YYY. Wallet should return the current player balance from the second win attempt without adding this second win amount again.
SECTION 2	11_SS (for test case 11_SS) (ISB GAMES ONLY)	Player presses spin. GP sends bet to the Wallet with session id AAA and round id BBB. Wallet accepts the bet and returns a successful answer. GP sends win to the Wallet with session id AAA, round id BBB and transaction id CCC. Wallet accepts the win and returns a successful answer, however GP does not accept this answer and game crashes with the fake error. (at this point, reports must have differences) Player opens the same game again. GP sends to the Wallet the win with round id BBB, transaction id CCC but with a new session DDD. Wallet should return the current player balance, without adding the win amount again.

SECTION NUMBER	TEST CASES	Test Case Description
SECTION 2	40_CSSA_1 (for test case 40_CSSA)	Player presses spin. GP sends 4 identical bet calls to the Wallet one after another. Wallet deducts bet amount only once and returns successful answer from all the 4 attempts. However, GP simulates timeouts from all the 4 attempts. GP sends cancel call to the Wallet. Wallet returns successful answer from the cancel. GP registers cancel call into the recon table with the status DONE.
SECTION 2	40_CSSA_2 (for test case 40_CSSA)	Player presses spin. GP sends bet to the Wallet. Wallet accepts the bet and returns a successful answer. However, GP does not accept this answer and game will crash with the fake error. Before game crashes, GP sends 3 identical cancel calls to the Wallet. First cancel call should refund the bet amount on the Wallet side and the 2 next cancel calls should just return the player balance without any operations on the player's money. GP registers one cancel call into the recon table with the status DONE.
SECTION 2	50_RC (for test case 50_RC)	Player presses spin. GP sends 4 identical bet calls to the Wallet one after another. Wallet deducts bet amount only once and returns successful answer from all the 4 attempts. However, GP simulates timeouts from all the 4 attempts. GP sends cancel call to the Wallet. Wallet returns successful answer from the cancel. However, GP simulates timeout from the cancel in order to save cancel call to the recon table. GP sends cancel call again (in the background process) and Wallet returns a successful answer. This cancel call will have its status changed (in the recon table) from PENDING to DONE after a successful Wallet answer.
SECTION 2	51_RE (for test case 51_RE)	Player closes the game. GP sends end call to the Wallet. Wallet accepts the end and returns a successful answer. However, GP simulates timeout from end method in order to save end call to the recon table. GP sends end call again (in the background process) and Wallet returns a successful answer. Session is being closed successfully.
SECTION 3	R_03 (for test case 00_EC)	Player opens a game. GP will generate invalid HASH as a part of the INIT request.

SECTION NUMBER	TEST CASES	Test Case Description
SECTION 3	R_11 (for test case 00_EC)	Player opens a game. To simulate, a game has to be opened twice for this player. On second attempt, GP will send INIT request with session that was already used in the first attempt.
SECTION 3	B_03 (for test case 00_EC)	Player presses spin. GP will send BET request with bet higher than the current balance.
SECTION 3	W_07 (ISB GAMES ONLY) (for test case 00_EC)	Player opens the game for operator XXX and press the spin. Player won free spins but closed the game before free spin round has been finished. Player opens the same game for operator YYY. Wallet should return the error on the first free win for new session after restore.
SECTION 4	high volume tests	Those players will be used only for high volume tests. For those players, GP will simulate random timeouts on the following calls to the Wallet: **bet**, **win**, **cancel**, **end**.

11.0 REPORTS REQUESTED FROM WP FOR TEST USERS

WP should provide an URL to be available for the **iSoftBet** platform during the tests in order to compare the reports that are registered on the **WP** side with the reports that were registered on the **GP** side.

Request:

Example of the URL with the **GET** parameters:
WALLET_URL/report.php?playerid=xxx&date_from=xxx&date_to=xxx&operator=xxx

Parameter	Required	Type	Description
playerid	Yes	STRING 64	Player Id
operator	No	STRING 250	Operator name. Optional. If an operator is not provided, then please return the report including all the operators.
date_from	No	ISO-8601	Report start date. Optional. If the start date is not provided, then please return the report since the beginning. **Example**: 2015-05-10T15:52:01+0000
date_to	No	ISO-8601	Report end date. Optional. If the end date is not provided, then please return the report including all the recent sessions. **Example**: 2015-05-10T15:52:01+0000 **Important:** **For reports generation, do not take into consideration the session end date.**

Successful Response:

Example of JSON object returned in the body:

```
{
   "session": 1,
   "rounds": 2,
   "transactions": 3,
   "bets": 2,
   "bets_cancelled": 1,
   "bets_amount": 100,
   "bets_amount_cancelled": 50,
```

```
   "wins": 1,
   "wins_amount": 300,
   "fround_bets": 1,
   "fround_wins_amount": 300,
   "jp_contribution": 0.0375,
   "jp_wins": 0
}
```


Response details:

Name	Type	Description
sessions	INT	Total count of player sessions (including all the sessions with status: ACTIVE, INACTIVE, ERROR)
rounds	INT	Total count of player's rounds (including all the sessions with status: ACTIVE, INACTIVE, ERROR)
transactions	INT	Total count of bet and win transactions made on player's account
bets	INT	Total count of player bets
bets_amount	INT	Sum of successful (with **transactionstatus** OK) bets amount in coin value; bets with **fround_id** greater than 0 should not be included here
bets_amount_cancelled	INT	Sum of cancelled bets amount in coin value
wins	INT	Total count of player wins
wins_amount	INT	Sum of wins in coin value
fround_bets	INT	Count of bets with **fround_id** greater than 0
fround_wins_amount	INT	Sum of wins with **fround_id** greater than 0
jp_contribution	DOUBLE	Sum of all the **JP** contributions that are sent in the **JPC** parameter for successful bets with **fround_id paramer** equal to 0
jp_wins	DOUBLE	Sum of all the **JP** wins for successful wins

All the amounts in the report should be presented in **cents**.

12.0 SESSION BASED INTEGRATION

Important:	**If the type of integration chosen for you is Real time seamless integration, then please disregard this section of the document.**

12.1 Overview

This type of integration considers only two calls form the **Game Platform** to the **Wallet Platform** – one on session open and second on session end.

The Wallet Platform is responsible for blocking the player's balance when it receives the **init** call (for more details about the **init** call, please go to 5.1 init).

The Wallet Platform must keep the player's balance blocked (not allow withdrawals, deposits and play on any other game) until it receives a second call for the session with information about the session's end.

The second call uses a multi-state request format, where in one call it sends all the bets and wins that occurred during the session plus the **end** command which informs the Wallet Platform about a status of the session (if any error occurred during the session or not).

12.2 Requests

12.2.1 FIRST CALL on session start

This example is of a single state request when the player opens a game.

Example:

```
{
   "sessionid": "3d7 … 19c1",
   "playerid": "player01",
   "skinid": 1001,
   "state": "single",
   "action": {
        "command": "init",
        "parameters": {
               "token": "2d7 … 19c1"
               "country": "GB"
        }
   }
}
```

Note: You can see the format of the parameters in table **Standard JSON parameters** within section 2.4.3 Request format and in table **Action parameters** within section 5.1 init.

12.2.2 SECOND CALL on session end

This is an example of a multi-state request for a player that placed **two** bets and had **two** winnings where one of the winnings includes the JP win.

The Wallet Platform can save each transaction separately in its database or just sum them up and deduct/add money to the player's balance.

Example:

```
{
"sessionid": "3d7 … 19c1",
"playerid": "player01",
"currency": "EUR",
"skinid": 1001,
"state": "multi",
"actions":
     [
          {
                    "command": "bet",
                    "parameters": {
                              "transactionid": "2d7..c1",
                              "roundid": "2d7 … 19c1",
                              "amount": 50,
                              "jpc": 0.9999
                    }
          } , {
                    "command": "win",
                    "parameters": {
                              "transactionid": "2d7 …c1",
                              "roundid": "2d7 … 19c1",
                              "amount": 50,
                              "jpw": 50,
                              "closeround": true
                    }
          } , {
                    "command": "bet",
                    "parameters": {
                              "transactionid": "2d7..c1",
```

```
                                "roundid": "2d7 … 19c1",
                                "amount": 250,
                                "jpc": 0.9999
                        }
                } , {
                        "command": "win",
                        "parameters": {
                                "transactionid": "2d7 …c1",
                                "roundid": "2d7 … 19c1",
                                "amount": 150,
                                "jpw": 0,
                                "closeround": true
                        }
                } , {
                        "command": "end",
                        "parameters": {
                                "sessionstatus":"INACTIVE"
                        }
                }
        ]
}
```

Note: You can see the format of the parameters in table **Standard JSON parameters** within section 2.4.3 Request format, in table **Request** within section 5.3 bet, in table **Request** within section 5.4 win and in table **Request** within section 5.6 end.

12.3 Wallet response

12.3.1 Wallet response on first call

The Wallet should respond with the player's info and the player's current balance.

Successful Response:

Parameter	Required	Type	Description
status	Yes	STRING 32	"success"
playerid	Yes	STRING 64	Player Identifier
sessionid	Yes	STRING 32	Session Identifier given in general Request part

Parameter	Required	Type	Description
currency	Yes	STRING 3	**ISO 4217** currency code
balance	Yes	INT 32	Player's balance in cents

Example:

```
{
   "status":          "success",
   "sessionid":       "3d7 … 19c1",
   "playerid":        "00001",
   "currency":        "EUR",
   "balance":         100000
}
```

Error Response

Parameter	Required	Type	Description
status	Yes	STRING 32	"error"
code	Yes	STRING 4	Identification code of the Error
message	Yes	STRING 255	UTF-8 text about internal Wallet Errors/Notices to inform **iSoftBet**

Example:

```
{
   "status":          "error",
   "code":            'ER01',
   "message":         "Error description",
}
```

12.3.2 Wallet response on second call

The Wallet should respond with the player's info and the player's new updated balance (after bets and wins execution).

Successful Response:

Parameter	Required	Type	Description
status	Yes	STRING 32	"success"
playerid	Yes	STRING 64	Player Identifier
sessionid	Yes	STRING 32	Session Identifier given in general Request part
currency	Yes	STRING 3	**ISO 4217** currency code
balance	Yes	INT 32	Player's balance in cents

Example:

```
{
   "status":         "success",
   "sessionid":      "3d7 … 19c1",
   "playerid":       "00001",
   "currency":       "EUR",
   "balance":        100000
}
```

Error Response

Parameter	Required	Type	Description
status	Yes	STRING 32	"error"
code	Yes	STRING 4	Identification code of the Error
message	Yes	STRING 255	UTF-8 text about internal Wallet Errors/Notices to inform **iSoftBet**

Example:

```
{
   "status":         "error",
   "code":           'ER01',
   "message":        "Error description",
}
```

Important:	**If the Wallet responds with an error, then all the bets and wins are cancelled on the Game Platform side.**

iSOFTBET

SIMPLY PLAY

1-7 Boundary Row,
London SE1 8HP

T: +44 (0) 208 133 2907
F: +44 (0) 207 117 1220

E: sales@isoftbet.com
www.isoftbet.com