# Communication

All requests and responses are send in json format. Game client (GC) sends request to game server (GS)and game server responds with the response. GC can send more than one request in single json, then GS responds for more than one request in single json. In this case game developer has to take attention for the order of the requests. GS processes the request in order defined in json file.

# Generic request command:

{

"requests": [{

"commandReq": "commandName\_A",

...,

"priority": 0

},

{

"commandReq": "commandName\_B",

...,

"priority": 2

},

{

"commandReq": "commandName\_C",

...,

"priority": 1

}

]

}

In example above we have 3 request in single json. Request are proceeded in order defined by *priority* object.

# Generic response command:

{

"responses": [{

"commandRes": " commandName\_A",

"errorCode": 0,

"errorMsg": "ok",

"priority": 0,

...

}, {

"commandRes": " commandName\_C",

"errorCode": 0,

"errorMsg": "ok",

"priority": 1,

...

}, {

"commandRes": " commandName\_B",

"errorCode": 0,

"errorMsg": "ok",

"priority": 2,

...

}]

}

Each response has *errorCode* and *errorMsg*. If command was executed successfully error code is 0 and error message is “ok”. In case of any error the *errorMsg* will contain short error description.

# Game server commands:

Currently game server supports an engine for simple slot games without free games. New game engines will be available soon. The GS has 3 game commands:

* CommInit
* CommPlay
* CommGetPaytable

and 2 debug commands:

* CommSetReels
* CommClearCheats

The game commands are available to game clients in production environment, debug commands are available only in test environment, and works only with special debug currency.

## Game commands:

Game commands are available at production and developer environment.

### CommInit:

The command ‘*CommInit*’, should be used when game client starts, and need to restore last game state. This command provides all information required to restore game state. This command does not change game state. It reads only static information. Hence the result of this command will not be written in history or blockchain.

#### CommInit request:

{

"requests": [{

"gameId": "DbUp",

"userId": "user00",

"currencyUnit": "dbg",

"commandReq": "CommInit",

"reelsLength": [10, 13, 16],

"priority": 0

}

]

}

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### userId

This is player account id. In dev environment it is your account id in our system. Use id used when you were buying an access to test environment.

##### currencyUnit

In test environment should be always ‘*dbg*’. In production system should be ‘*JoyToken*’

##### commandReq

must be ‘*CommInit*’

##### reelsLength

When a player presses ‘*Spin*’ button a *CommPlay* request should be send to the server and a game client should start to spin the reels. But the game client may not receive the response for the *CommPlay* on time and does not know which symbol should be visible during spin. In order to solve this problem *CommInit* response delivers to the GC the spin reels. These are reels with the same begin and end (later called *dummy* *spin reels*). GC should use these reels (spin in loop) until it receives response for the *CommPlay*. The *CommPlay* defines reels for reel spin with proper reel-end and reel-begin (later called *spin reels*). Take a note that the *dummy spins reels* begins and ends with start symbol combination, but the *spin reels* starts with start symbol combination and ends with end symbol combination, hence, when GC receives response for *CommPlay*, it should finish the spin with the *spin reels* and reel concatenation should not be visible to the player.

The *reelLength* table determinates sizes of ’*dummy spin reels*’ for the next spin. The minimum size is 9. Size of this table must be equal to number of reels.

##### priority

Determinates an order in json with multiple requests. This field must exist even in single request json.

#### CommInit response:

{

"responses": [{

"balance": 499380,

"commandRes": "CommInit",

"currencyUnit": "dbg",

"defaultStake": 200,

"denomination": 100,

"errorCode": 0,

"errorMsg": "ok",

"gameId": "this.wssM\_ulGameId",

"lines": [

[1, 1, 1]

],

"linesBet": [1],

"nextSpinReels": [

[7, 8, 7, 8, 6, 2, 7, 8, 7, 8],

[8, 7, 6, 9, 6, 8, 6, 7, 1, 8, 7, 6, 9],

[6, 8, 7, 6, 3, 8, 7, 2, 6, 8, 9, 5, 6, 8, 7, 6]

],

"priority": 0,

"stakes": [20, 60, 100, 200, 600, 800, 1000, 2000, 5000, 10000, 25000, 50000],

"tokenization": 100,

"userId": "user00"

}]

}

##### balance

This is player balance in balance units. There is some logic described below how to show balance in ‘*JoyToken chips*’ used in game client to visualize account balance and win. All bets and are always processed in the balance balance units.

##### denomination

Determinates how many decimal numbers has JoyToken chips. It is similar to dollars and cents. A value 100 means that 1 chip is 100 chip cents.

##### Tokenization

Determinates value of 1 chip, in balance units. A value 100 means that 1chip is a 100 balance units. In this example we have balance: 499380, denomination 100 and tokenization 100. It means that:

* 1 chip = 100 balance units
* 1 chip = 100 chip cents.
* 499380 should be displayed as 4993.80 chips

Another example. Balance 499380, denomination 1000, tokenization 10000 means:

* 1 chip = 10000 balance units
* 1 chip = 1000 chip cents
* 499380 should be displayed as 49.938 chips

##### commandRes

It is a name of the request command for the response.

##### currencyUnit

In test environment should be always ‘*dbg*’. In production system should be ‘*JoyToken*’

##### defaultStake

It is default stake when a player does not select any. A value is in balance units.

##### errorCode

Error code for the response. 0 if successful.

##### errorMsg

Error message for the response. ‘*ok*’ if successful.

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### lines

This is a table that describes lines in reel game. A table contains table of positions. In case of 3 reel game it is table of 3 elements tables. 0 means top symbol, 1 means middle symbol, 2 means bottom symbol. In example above we have only one line defined in the game and it selects middle symbols at all 3 reels.

##### linesBet

This is a table that determinates how number of lines bet can change. In example above we have only one line, and this table must be one element table with value ‘*1*’. But if we had 9 lines games (table ‘*lines*’ has 9 elements) and:

“LinesBet”:[1,3,5,7,9]

It means that lines bet can be changed only in set {1,3,5,7,9}, which means that is not possible to bet 2 lines, because it is not in the set.

##### nextSpinReels

These is table with *dummy reel spin*. Size of this table corresponds to number of reels in a game. Size of elements in this table corresponds to values in request (*reelLength*)

##### priority

This is the same number as *priority* in request.

##### stakes

This is a table with available stakes per line. Game server will accept only stakes from this set. Please note that it is not a total bet. Total bet is a product of bet per line (value from *stakes* table) and lines bet (value from *linesBet* table). Stakes are in balance unit.

##### userId

This is the same userId as in request.

### CommPlay

*CommPlay* is a command that should be send when a player presses ‘*spin*’ button. Before first usage of this command *CommInit* request should be send to the GS, for the *userId* and *currencyUnit*. Later *commPlay* can be send without *CommInit* request. This command makes a game for the player on GS. The command changes game state for the player, hence the results of this command will be written in game history and blockchain.

#### CommPlay request

{

"requests": [{

"gameId": "DbUp",

"userId": "user00",

"currencyUnit": "dbg",

"commandReq": "CommPlay",

"linesBet": 1,

"totalBet": 20,

"reelsLength": [25, 29, 37],

"nextReelsLength": [10, 13, 16],

"priority": 0

}]

}

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### userId

This is player account id. In dev environment it is your account id in our system. Use id used when you were buying an access to test environment.

##### currencyUnit

In test environment should be always ‘*dbg*’. In production system should be ‘*JoyToken*’

##### commandReq

must be ‘*CommPlay*’

##### linesBet

This is a number of lines bet. This value must be one of the value from table ‘*linesBet*’ in *CommInit* response.

##### totalBet

This is total bet in the game. This value is in balance units. This value must corresponds to ‘*linesBet*’ table and ‘*stakes*’ table in *CommInit* response. Please remember that:

total bet = bet per line (*stakes* table element) x lines bet (*linesBet* table element)

##### reelsLength

This is a table that contains length of *spin reels* (see *reelLenght* explanation for *CommInit* request), for the current spin.

##### nextReelsLength

This is a table that contains *dummy spin reels* *reels* (see *reelLenght* explanation for *CommInit* request) for the next reel spin (not for current reel spin).

##### priority

Determinates an order in json with multiple requests. This field must exist even in single request json.

#### CommPlay response

{

"responses": [{

"balance": 518800,

"commandRes": "CommPlay",

"currencyUnit": "dbg",

"denomination": 100,

"errorCode": 0,

"errorMsg": "ok",

"gameId": "DbUp",

"nextSpinReels": [

[7, 0, 6, 8, 7, 8, 7, 0, 6, 8],

[7, 0, 6, 8, 9, 6, 3, 7, 8, 7, 0, 6, 8],

[8, 0, 6, 8, 7, 8, 7, 6, 5, 8, 7, 6, 8, 0, 6, 8]

],

"priority": 0,

"signature": "",

"spinReels": [

[7, 0, 6, 8, 7, 8, 6, 7, 5, 8, 7, 6, 2, 8, 9, 7, 8, 4, 6, 7, 8, 7, 0, 6, 8],

[7, 0, 6, 8, 7, 8, 7, 5, 8, 7, 6, 8, 2, 6, 7, 8, 7, 6, 5, 7, 6, 7, 4, 6, 7, 6, 1, 7, 6],

[8, 0, 6, 8, 6, 5, 8, 7, 6, 8, 0, 6, 8, 9, 8, 1, 6, 7, 8, 6, 7, 5, 8, 7, 6, 8, 7, 6, 8, 7, 4, 6, 8, 7, 2, 6, 8]

],

"totalWin": 20000,

"userId": "user00",

"wins": [{

"animSymbol": [0, 0, 0],

"cnt": 3,

"lineNo": 0,

"pos": [1, 1, 1],

"winSymbol": 0,

"winType": 2,

"winValue": 20000

}]

}]

}

##### balance

This is player balance in balance units.

##### commandRes

It is a name of the request command for the response.

##### currencyUnit

In test environment should be always ‘*dbg*’. In production system should be ‘*JoyToken*’

##### denomination

Determinates how many decimal numbers has JoyToken chips. It is similar to dollars and cents. A value 100 means that 1 chip is 100 chip cents.

##### errorCode

Error code for the response. 0 if successful.

##### errorMsg

Error message for the response. ‘*ok*’ if successful.

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### nextSpinReels

This is a table that contains *dummy spin reels* for next spin (not the current one). Game client should use in current spin *dummy spin reels* delivered in last *CommInit* response or *CommPlay* response.

##### priority

This is the same number as *priority* in request.

##### spinReels

This is a table that contains *spin reels* for current spin.

##### totalWin

This is total win in the game in balance units.

##### userId

This is the same userId as in request.

##### wins

This is a table that contains object with win descriptors. There will be as many objects in the table as many wins is in the spin outcome.

###### animSymbol

This is a table that contains which information which symbol is animated at particular reel. This is not a position of the symbol on the screen. It is a symbol number from reel. In this example 0 is probably the highest win symbol.

###### cnt

This is how many winning symbols is in the win.

###### lineNo

This is at which line win was hit

###### pos

This is position of the winning symbols on the screen. *[1,1,1]* means middle symbol at all 3 reels.

###### winSymbol

This is the winning symbol. In our example we have *cnt*=3 and *winSymbol*=0, which means that we have 3xsymbol\_0 win.

###### winType

This is type of the win.

* 0 – unknown win
* 1 – scatter win
* 2 – line win

###### winValue

Win value in balance units.

##### signature

This is signature of hash of the response. The hash is calculated according to following formula:

#n(data, #n-1)

Hash algorithm is md5. Sign algorithm is sha256. Not all response string is hashed and signed, only part that contains data. In our example it is:

[{

"balance": 518800,

"commandRes": "CommPlay",

"currencyUnit": "dbg",

"denomination": 100,

"errorCode": 0,

"errorMsg": "ok",

"gameId": "DbUp",

"nextSpinReels": [

[7, 0, 6, 8, 7, 8, 7, 0, 6, 8],

[7, 0, 6, 8, 9, 6, 3, 7, 8, 7, 0, 6, 8],

[8, 0, 6, 8, 7, 8, 7, 6, 5, 8, 7, 6, 8, 0, 6, 8]

],

"priority": 0,

"spinReels": [

[7, 0, 6, 8, 7, 8, 6, 7, 5, 8, 7, 6, 2, 8, 9, 7, 8, 4, 6, 7, 8, 7, 0, 6, 8],

[7, 0, 6, 8, 7, 8, 7, 5, 8, 7, 6, 8, 2, 6, 7, 8, 7, 6, 5, 7, 6, 7, 4, 6, 7, 6, 1, 7, 6],

[8, 0, 6, 8, 6, 5, 8, 7, 6, 8, 0, 6, 8, 9, 8, 1, 6, 7, 8, 6, 7, 5, 8, 7, 6, 8, 7, 6, 8, 7, 4, 6, 8, 7, 2, 6, 8]

],

"totalWin": 20000,

"userId": "user00",

"wins": [{

"animSymbol": [0, 0, 0],

"cnt": 3,

"lineNo": 0,

"pos": [1, 1, 1],

"winSymbol": 0,

"winType": 2,

"winValue": 20000

}]

}

Note that multirequest header and *signature* object are not hashed.

### CommGetPaytable

The command ‘*CommGetPaytable*’is not obligatory. It provides information for game client how to build paytable. This command reads only static information, hence the result if this command will not be written in history or blockchain.

#### CommGetPaytable request

{

"requests": [{

"gameId": "DbUp",

"commandReq": "CommGetPaytable",

"linesBet": 1,

"totalBet": 20,

"priority": 0

}]

}

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### commandReq

must be ‘*CommGetPaytable*’

##### linesBet

This is a number of lines bet. This value must be one of the value from table ‘*linesBet*’ in *CommInit* response. Response will be recalculated for value provided in this object.

##### totalBet

This is total bet in the game. This value is in balance units. This value must corresponds to ‘*linesBet*’ table and ‘*stakes*’ table in *CommInit* response. Please remember that:

total bet = bet per line (*stakes* table element) x lines bet (*linesBet* table element)

Response will be recalculated for value provided in this object.

##### priority

Determinates an order in json with multiple requests. This field must exist even in single request json.

#### CommGetPaytable response

{

"responses": [{

"commandRes": "CommGetPaytable",

"errorCode": 0,

"errorMsg": "ok",

"gameId": "DbUp",

"priority": 0,

"winRule": [{

"quantitiy": 3,

"symbol": 0,

"value": 1000,

"winType": 2

}, {

"quantitiy": 3,

"symbol": 1,

"value": 500,

"winType": 2

}, {

"quantitiy": 3,

"symbol": 2,

"value": 500,

"winType": 2

}, {

"quantitiy": 3,

"symbol": 3,

"value": 500,

"winType": 2

}, {

"quantitiy": 3,

"symbol": 4,

"value": 200,

"winType": 2

}, {

"quantitiy": 3,

"symbol": 5,

"value": 100,

"winType": 2

}, {

"quantitiy": 3,

"symbol": 6,

"value": 50,

"winType": 2

}, {

"quantitiy": 3,

"symbol": 7,

"value": 4,

"winType": 2

}, {

"quantitiy": 3,

"symbol": 8,

"value": 2,

"winType": 2

}, {

"quantitiy": 3,

"symbol": 9,

"value": 1,

"winType": 2

}]

}]

}

##### commandRes

It is a name of the request command for the response.

##### errorCode

Error code for the response. 0 if successful.

##### errorMsg

Error message for the response. ‘*ok*’ if successful.

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### priority

This is the same number as *priority* in request.

##### winRule

This is a table with wins. Number of elements in the table is equal to number of wins in the game. This table contains json objects that describes wins in details.

###### quantitiy

number of winning symbols in the win

###### symbol

winning symbol

###### value

win value for the win

###### winType

Win type of the win. See *winType* in *CommPlay* response for details.

## Debug commands

Debug command are available only in developer environment. They provide functionality that simplifies debugging game client.

### CommSetReels

*CommSetReels* is a command that allows developer to get specific symbol combination on the screen during next spins. When a developer sends to game server *CommSetReels* command, GS stops responds on *CommPlay* request with random combination and starts responds with combination provided by the developer. Single *CommSetReels* command adds combination to the list. The list is executed in loop. It means that: If a developer sends to GS *CommSetReels* with comb\_A, *CommPlay* response will be returning comb\_A constantly. If a developer sends later to GS *CommSetReels* with comb\_B, comb\_B will be added to the list, and *CombPlay* will be constantly returning, comb\_A, comb\_B, in the infinite loop.

#### CommSetReels request

{

"requests": [{

"gameId": "DbUp",

"userId": "user00",

"currencyUnit": "dbg",

"commandReq": "CommSetReels",

"setReels": [

[-1, 0, -1],

[-1, 1, -1],

[-1, 2, -1]

],

"priority": 0

}]

}

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### userId

This is player account id. In dev environment it is your account id in our system. Use id used when you were buying an access to test environment.

##### currencyUnit

Must be always ‘*dbg*’.

##### commandReq

must be ‘*CommSetReels*’

##### setReels

This is a table that contains information what developer wishes to see. Elements of this table are another tables (later called *screen reel tables*). Size of the table must be equal to number of reels in the game. *Screen reel table* size should be a number of symbol for the appropriate reel visible on the game client screen. Numbers in *sreen reel table* are symbol numbers. *-1* is any symbol. In our case:

*[-1,2-1]* means:

* Any symbol at top position
* Symbol 2 at middle position
* Any symbol at bottom position

Developer does not to care does combination exists on reels or not. If the combination does not exists, the response will provide appropriate error message.

##### priority

Determinates an order in json with multiple requests. This field must exist even in single request json.

#### CommSetReels response

{

"responses": [{

"commandRes": "CommSetReels",

"currencyUnit": "dbg",

"errorCode": 0,

"errorMsg": "ok",

"gameId": "DbUp",

"numberOfRecords": 1,

"priority": 0,

"userId": "user00"

}]

}

##### commandRes

It is a name of the request command for the response.

##### currencyUnit

In test environment should be always ‘*dbg*’. In production system should be ‘*JoyToken*’

##### errorCode

Error code for the response. 0 if successful.

##### errorMsg

Error message for the response. ‘*ok*’ if successful.

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### numberOfRecords

This is information how many records is currently created for the developer (*userId*). All records are executed in infinite loop until developer sends clear command, which deletes all records, and game server starts to response with random combination again.

##### priority

This is the same number as *priority* in request.

##### userId

This is the same userId as in request.

### CommClearCheats

This command clears records created by *CommSetReels*. When developer sends this command to game server, the GS starts to respond on *CommPlay* with random combination instead of combinations set by developer using *CommSetReels* command.

#### CommClearCheats request

{

"requests": [{

"gameId": "DbUp",

"userId": "user00",

"currencyUnit": "dbg",

"commandReq": "CommClearCheats",

"priority": 0

}]

}

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### userId

This is player account id. In dev environment it is your account id in our system. Use id used when you were buying an access to test environment.

##### currencyUnit

Must be always ‘*dbg*’.

##### commandReq

must be ‘*CommClearCheats*’

##### priority

Determinates an order in json with multiple requests. This field must exist even in single request json.

#### CommClearCheats response

{

"responses": [{

"commandRes": "CommClearCheats",

"currencyUnit": "dbg",

"errorCode": 0,

"errorMsg": "ok",

"gameId": "DbUp",

"priority": 0,

"userId": "user00"

}]

}

##### commandRes

It is a name of the request command for the response.

##### currencyUnit

In test environment should be always ‘*dbg*’. In production system should be ‘*JoyToken*’

##### errorCode

Error code for the response. 0 if successful.

##### errorMsg

Error message for the response. ‘*ok*’ if successful.

##### gameId

This is unique string for the game engine. Currently GS implements one game engine and the string for them is ‘*DbUp*’.

##### priority

This is the same number as *priority* in request.

##### userId

This is the same userId as in request.