Kafka topics

This page features all Kafka topics to which Event Streaming produces messages. Here, you will find:

- Outlines of each topic
- Examples of message structures
- Comprehensive payload field definitions

This documentation corresponds to version 1.6.0. See Changelog to keep up with the latest updates.

On this page:

Balane transactions

Subscribe to the `es.{casino_name}.balance_transaction` Kafka topic to consume any balance transaction of the player. A balance transaction can be:

- Deposit or cashout, refund or chargeback
- Bonus issue
- Balance corrections
- Various sportsbook activities
- Other operations

Explore the full list of types and actions in the table below.

Example: Player placed bet

```
{
  "id": 555,
  "client_name": "Best-casino",
  "user_id": 123,
  "reference_id": 101,
  "reference_type": "Game",
  "action": "bet",
  "currency": "EUR",
  "amount_cents": 700,
  "bonus_amount_cents": 0,
  "balance": 0,
  "created_at": "2020-02-02T00:00:00Z",
  "balance_before": 700,
  "game_info": {
        "game_table_id": 22,
    "name": "netent:devil_sw",
    "categories": ["slots"],
    "tx_number": 0
  },
  "bonus": {
    "bonus_issue_id": 15,
    "amount_wager_requirement_cents": 0,
    "amount_wager_cents": 0,
    "amount_locked_cents": 0
  },
  "version": 1640682537,
  "msg_id": "01ARZ3NDEKTSV4RRFFQ69G5KNA",
  "api_version": "1.0.0",
  "account_id": 147422
}
```

Field definitions

Field	Type	Description
`id`	number	Unique identifier for the balance transaction.
`client_name`	string	Name of the casino in the Casino Platform.
`user_id`	number	Unique identifier for the player. This field serves as a partition key.

`reference_type`	string	Type of the balance transaction. Each type has a set of possible actions defined in `action`. The possible reference types are: <ul><li>`BalanceAdjustment`: Balance accrual and balance withdrawal transaction. Balance accrual and balance withdrawal are transactions in which a player spends money on buying or selling items in the casino.</li><li>`BalanceCorrection`: Manual balance correction is made. In case of a transaction error, admins can adjust the player's balance manually (for example, add money or subtract it).</li><li>`BonusIssue`: Bonus is issued to the player.</li><li>`Game`: Player placed a bet in a game or won money.</li><li>`Gift`: Player has received a gift.</li><li>`GovernmentWithdrawal`**:** Romanian license. Money is confiscated from the balance of the player who was inactive for two years to the Romanian State Treasury.</li><li>`NationalTax`: Romanian license. Withholding tax transaction. The withholding tax is charged from the Romanian players' deposits and cashouts.</li><li>`Payment`: Player made a payment (for example, deposit or cashout).</li><li>`SportsbookBet`: Player placed a bet in Sportsbook.</li><li>`SportsbookBonusIssue`: Sportsbook bonus is issued to the player.</li><li>`SportsbookFreebet`: Sportsbook freebet is issued to the player.</li><li>`SportsbookJackpotWin`: Player hit a jackpot in Sportsbook.</li><li>`UserSession`: Greek license. Player's wins within a session have been taxed.</li></ul>Message structure is always the same regardless of `reference_type`. See examples in the Balance transaction examples section.
`reference_id`	number	Unique identifier for the balance transaction of `reference_type`. For example, if reference type is `BonusIssue`, `reference_id` points to a particular bonus. You can view bonus IDs on the **Issued bonuses** page in the Casino Platform. Or, if reference type is `Game`, `reference_id` points to particular bet. You can view bet IDs on the **Bets** page in the Casino Platform.

`action`	string	Specific player's action performed in the casino or Sportsbook. The possible values vary depending on the balance transaction type defined in `reference_type`. Expand the full list of possible values below: • `bet`: Player placed a bet in a game round. • `win`: Player won a game round. Bet and win of the same game round have the save `reference_id`. In this case, reference ID is a bet identifier from the **Bets** page in the Casino Platform. • `gift`: Gift is issued to the player. • `refund`: Casino operator refunded money to the player. • `rollback_bet`: Player's bet in casino is rolled back. Bet amount has been returned to the player. • `rollback_win`: Player's win in casino is rolled back. • `deposit`: Player has made a deposit. • `cashout`: Player has requested a cashout. • `chargeback`: Player has received a deposit chargeback. • `reversal`: Payment system has reversed the player's cashout. • `affiliate_payment`: Player's affiliate has received the money. • `addition`: Manual balance correction upward. • `subtraction`: Manual balance corrections downward. • `trade_accrual`: Funds have been added to the player's account via Balances API. • `trade_withdrawal`: Funds have been deducted from the player's account via Balances API. • `sportsbook_bet`: Player has placed a bet (for Sportsbook). • `sportsbook_win`: Player has won the bet (for Sportsbook). • `sportsbook_reject`: Player's bet has been rejected. Bet amount has been returned to the player (for Sportsbook). • `sportsbook_cancel`: Player's bet has been canceled. Bet amount has been returned to the player (for Sportsbook). • `sportsbook_rollback`: Transaction has been rolled back (for Sportsbook). • `sportsbook_rollback_cancelled`: Player's rollback has been canceled (for Sportsbook). • `sportsbook_gift`: Player has received a gift. A player can cashout the gift money with no need to wager them (for Sportsbook). • `sportsbook_subtraction`: Manual balance correction downward (for Sportsbook). • `sportsbook_addition`: Manual balance correction upward (for Sportsbook). • `sportsbook_issued_comboboost`: Comboboost bonus has been issued to the player (for Sportsbook). • `sportsbook_issued_freebet_no_risk`: No risk free bet has been issued to the player (for Sportsbook). • `sportsbook_freebet_win`: Player has won the free bet (for Sportsbook). • `sportsbook_issued_comboboost_rollback`: Issued comboboost has been rolled back from the player (for Sportsbook). • `sportsbook_issued_freebet_no_risk_rollback`: Issued no risk free bet has been rolled back (for Sportsbook). • `sportsbook_freebet_win_rollback`: Freebet win has been rolled back from the player (for Sportsbook). • `exchange_deposit`: Crypto to fial conversion has been made for a deposit. • `exchange_withdrawal`: Crypto to fiat conversion has been made for a withdrawal. • `issued_bonus`: Player has received bonus money. • `canceled_bonus`: Player has lost not wagered bonus money. • `bet_tax`: Tax has been charged for the player's bet. • `rollback_bet_tax`: Tax charge has been canceled, the player has received the charged money back. • `country_win_tax`: Tax has been imposed on the player's win. Valid for the casinos operating under the Greek license.
`currency`	string	A three-letter ISO currency code (for example, `"EUR"`).

`amount_cents`	number	Transaction amount in the smallest target currency unit. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.

`bonus_amount_cents`	number	Amount of the bonus money in the smallest currency unit. `0` if there have been no bonuses. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.
`balance`	number	Account balance in the smallest currency unit when a transaction has been made.
`created_at`	string	Date and time when a transaction has been made in the ISO 8601 format (YYYY-MM-DDThh:mm:ssZ). **Note** that this field never changes. For example, in the case of data re-push, the value of this field remains intact and stores its initial value.
`balance_before`	number|null	Account balance in the smallest currency unit before a transaction has been made. In games with several rounds this field will be updated on each player's bet and win. **Note** that this field is always `null` unless a transaction isn't associated with a game activity (`reference_type: "Game"`).
`game_info`	object	Information on the game the player is playing in. **Note** that this field is always `{}` (an empty object) unless `reference_type: "Game"`.
`game_info.game_table_id`	number	Unique identifier for the game table. Game table is a game launched with a specific currency.
`game_info.name`	string	Game title. Includes the game title and its provider separated by a semicolon. For example, `"netent:devil_sw"`.
`game_info.categories`	array [string]	An array of strings where each element is a game category the game belongs to.

`game_info.tx_number`	number	Sequence number of the transaction within the same game. For example, if a player places a bet, it will be the first transaction, and if they win – it will be the second. **Note** that for the first game transaction, the value will be `0`.
`bonus`	object	Information on the bonus. **Note** that this field is always `{}` (an empty object) unless `reference_type: "Game"` and the bonus issue itself is present.
`bonus.bonus_issue_id`	number	ID of the bonus issue if a bonus issue has been made. This field stores the same value as in `game_info.bonus_issue_id`.
`bonus.amount_wager_requirement_cents`	number	An amount of money – in the smallest target currency unit – the player needs to wager to receive the bonus. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.

`bonus.amount_wager_cents`	number	An amount of money – in the smallest target currency unit – the player has already wagered. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.

`bonus.amount_locked_cents`	number	Locked amount of money in the smallest currency unit. While the player profits and hasn't wagered the bonus yet, they can't withdraw this locked amount. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.
`version`	number	Unix representation of `created_at`. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while msg_id will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).
`account_id`	number	Unique numeric identifier for the player's account where the balance transaction occurred.

Balance transaction examples

See examples for balance transactions of some `reference_type`. All the rest types have the same structure. Only values of the properties differ.

BalanceCorrection

```
{
        "id": 8853,
        "client_name": "Best-casino",
        "user_id": 5344,
        "account_id": 61399,
        "reference_id": 583,
        "reference_type": "BalanceCorrection",
        "action": "subtraction",
        "currency": "BRL",
        "amount_cents": -15090,
        "bonus_amount_cents": 0,
        "balance": 0,
        "balance_before": null,
        "created_at": "2023-03-06T04:16:58Z",
        "game_info": {},
        "bonus": {},
        "version": 1678076218,
        "msg_id": "01GTTJ63N4W2V1P9S7Y1TTH9PE",
        "api_version": "1.3.0"
}
```

BonusIssue

```
{
        "id": 8853,
        "client_name": "Best-casino",
        "user_id": 5344,
        "account_id": 6107231,
        "reference_id": 3112,
        "reference_type": "BonusIssue",
        "action": "issued_bonus",
        "currency": "USD",
        "amount_cents": 785,
        "bonus_amount_cents": 0,
        "balance": 1570,
        "balance_before": null,
        "created_at": "2023-03-06T04:02:29Z",
        "game_info": {},
        "bonus": {},
        "version": 1678075349,
        "msg_id": "01GTTHBKB24BY9RFRWPPQ7Z5WQ",
        "api_version": "1.3.0"
}
```

Payment

```
{
        "id": 8849841742,
        "client_name": "Best-casino",
        "user_id": 5086571,
        "account_id": 5887114,
        "reference_id": 18763080,
        "reference_type": "Payment",
        "action": "deposit",
        "currency": "BTC",
        "amount_cents": 202122,
        "bonus_amount_cents": 0,
        "balance": 202496,
        "balance_before": null,
        "created_at": "2023-03-05T20:32:28Z",
        "game_info": {},
        "bonus": {},
        "version": 1678048348,
        "msg_id": "01GTSQKJVBDRS0R6F69E0119CH",
        "api_version": "1.3.0"
}
```

SportsbookBet

```
{
        "id": 5403,
        "client_name": "Best-casino",
        "user_id": 16398,
        "account_id": 21029,
        "reference_id": 29965,
        "reference_type": "SportsbookBet",
        "action": "sportsbook_win",
        "currency": "USD",
        "amount_cents": 1,
        "bonus_amount_cents": 0,
        "balance": 100432,
        "balance_before": null,
        "created_at": "2023-02-28T18:11:19Z",
        "game_info": {},
        "bonus": {},
        "version": 1677607879,
        "msg_id": "01GTCKHGMPWDKRS47YKG9XR6KH",
        "api_version": "1.3.0"
}
```

SportsbookBonusIssue

```
{
        "id": 54104,
        "client_name": "Best-casino",
        "user_id": 30964,
        "account_id": 32689,
        "reference_id": 300,
        "reference_type": "SportsbookBonusIssue",
        "action": "sportsbook_issued_comboboost",
        "currency": "EUR",
        "amount_cents": 297,
        "bonus_amount_cents": 0,
        "balance": 15159,
        "balance_before": null,
        "created_at": "2023-03-01T03:59:49Z",
        "game_info": {},
        "bonus": {},
        "version": 1677643189,
        "msg_id": "01GTDN733JNBAS1STTYQEA3CJT",
        "api_version": "1.3.0"
}
```

SportsbookFreebet

```
{
        "id": 541,
        "client_name": "Best-casino",
        "user_id": 5161,
        "account_id": 5441,
        "reference_id": 441,
        "reference_type": "SportsbookFreebet",
        "action": "sportsbook_freebet_win",
        "currency": "AUD",
        "amount_cents": 9750,
        "bonus_amount_cents": 0,
        "balance": 9750,
        "balance_before": null,
        "created_at": "2023-03-01T13:04:19Z",
        "game_info": {},
        "bonus": {},
        "version": 1677675859,
        "msg_id": "01GTEMC3E4V2NZND24XESRXK5M",
        "api_version": "1.3.0"
}
```

UserSession

```
{
        "id": 162,
        "client_name": "Best-casino",
        "user_id": 4531,
        "account_id": 5015,
        "reference_id": 2553,
        "reference_type": "UserSession",
        "action": "country_win_tax",
        "currency": "EUR",
        "amount_cents": -207,
        "bonus_amount_cents": 0,
        "balance": 34675,
        "balance_before": null,
        "created_at": "2022-02-22T12:38:44Z",
        "game_info": {},
        "bonus": {},
        "version": 1645533524,
        "msg_id": "01GXBHDH9RC747E4B5KN97PZ1B",
        "api_version": "1.3.1"
}
```

Gift

```
{
        "id": 1144,
        "client_name": "Best-casino",
        "user_id": 6932,
        "account_id": 7320,
        "reference_id": 3100,
        "reference_type": "Gift",
        "action": "gift",
        "currency": "EUR",
        "amount_cents": 10000,
        "bonus_amount_cents": 0,
        "balance": 10000,
        "balance_before": null,
        "created_at": "2023-03-22T17:18:00Z",
        "game_info": {},
        "bonus": {},
        "version": 1679505480,
        "msg_id": "01GW557PM2DW3PR3EB1MHF6607",
        "api_version": "1.3.0"
}
```

Bonus issue

Subscribe to the `es.{casino_name}.bonus_issue` Kafka topic to consume events associated with the bonus issue flow, for example, status updates. Explore the message structure and field definitions below:

Example: Player activated bonus

```
{
  "id": 5704,
  "client_name": "Best-casino",
  "operation": "update",
  "user_id": 16279,
  "stage": "handle_bets",
  "title": "Signup Bonus for everybody",
  "currency": "EUR",
  "created_at": "2023-06-20T09:13:36Z",
  "activated_at": "2023-06-21T10:12:00Z",
  "finished_at": "2023-06-22T10:12:00Z",
  "activatable_until": null,
  "valid_until": "2022-04-27T09:13:36Z",
  "amount_cents": 1000,
  "amount_wager_requirement_cents": 5000,
  "amount_wager_cents": 0,
  "strategy": "registration",
  "payment_id": null,
  "freespin_id": null,
  "version": 1650446016,
  "msg_id": "01G133Z8G4CGZR3XZ5NVZDRATR",
  "api_version": "0.2.0",
  "account_id": 147422,
  "updated_at": "2022-04-20T09:20:36Z"
}
```

Field definitions

Field	Type	Description
`id`	number	Unique identifier for the bonus issue.
`client_name`	string	Name of the casino in the Casino Platform.
`operation`	string	Name of the operation with the bonus issue. Has the only value: • `update` — bonus status has been changed.
`user_id`	number	Unique identifier for the player. This field serves as a partition key.
`stage`	string	Current bonus stage. The possible values are: • `issued` — bonus has been issued to the player. The player can either activate or cancel the bonus. • `handle_bets` — player has activated the bonus. • `lost` — bonus money has been wagered and lost. • `expired` — the bonus time frame has expired. Bonus money and related winnings have been removed from the player's account. • `wager_done` — the player has met the wagering requirements. • `canceled` — the player or operator has canceled the bonus.
`title`	string	Bonus title. This title is displayed to the player.
`currency`	string	Three-letter ISO currency code. Indicates the player's account to which the bonus has been issued. For example, `"EUR"`.
`created_at`	string	Date and time when the bonus has been issued, in the ISO 8601 format (YYYY-MM-DDThh:mm:ssZ). **Note** that this field never changes. For example, in the case of data re-push, the value of this field remains intact and stores its initial value.
`activated_at`	string|null	Date and time when the player activated the bonus. The date is in the UTC format. If `null`, the bonus isn't activated yet.

`finished_at`	string|null	Date and time when the bonus expired. Shown in the UTC format. The field is `null` in the following cases: • The bonus isn't activated. • The bonus is activated and not expired.
`activatable_until`	string|null	Date and time before which the bonus must be activated, in the ISO 8601 format (YYYY-MM-DDThh:mm:ssZ). If the bonus hasn't been activated in time, it becomes expired. `null`, if the bonus can be activated without any time limits.
`valid_until`	string|null	Bonus expiry date, in the ISO 8601 format (YYYY-MM-DDThh:mm:ssZ). If wagering requirements haven't been met within the period of bonus validity, both bonus money and winnings will be removed from the player's balance. `null`, if the bonus is permanently valid.
`amount_cents`	number	Bonus amount in the smallest currency unit. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.

`amount_wager_requirement_cents`	number	Amount of money the player needs to wager before they will be able to withdraw the winnings. In the smallest currency unit. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.

`amount_wager_cents`	number	Amount of money the player has already wagered, in the smallest currency unit. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.
`strategy`	string	Bonus issuing strategy. The reason the player has received the bonus. The possible values are: • `deposit` • `exchange` • `freespins_result` • `groups_updated` • `input_coupon` • `jackpot_award` • `manual` • `prize` • `prize_award` • `registration` • `scheduler` To read more about bonus strategies, go to the **Issues histories** page.
`payment_id`	number|null	Unique identifier for the player's payment that triggered the bonus issue. For example, for a deposit (`strategy: "deposit"`). `null`, if the bonus issue isn't associated with a player's payment.
`freespin_issue_id`	number|null	Unique identifier for the free spin issue that triggered the bonus issue (`strategy: "freespins_result"`). `null`, if the bonus issue isn't associated with a free spin.
`version`	number	Unix representation of `created_at`. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.

`api_version`	string	API version in the semantic versioning format (major.minor.patch).
`account_id`	number	Unique numeric identifier for the player's account to which the bonus was issued.
`updated_at`	string	Date and time when the bonus `stage` was updated in the UTC format.

Free spin issue

Subscribe to the `es.{casino_name}.freespin_issue` Kafka topic to consume events associated with the free spin issue flow, for example, status updates. Explore the message structure and field definitions below:

Example: Freespin bonus expired

```
{
  "id": 31791,
  "client_name": "Best-casino",
  "operation": "update",
  "user_id": 396251,
  "stage": "expired",
  "title": "Freespin coupon",
  "currency": "EUR",
  "created_at": "2022-04-12T08:32:31Z",
  "activatable_until": "2022-04-19T08:32:31Z",
  "valid_until": "2022-04-19T08:32:38Z",
  "freespins_total": 20,
  "freespins_performed": null,
  "win_amount_cents": 0,
  "strategy": "input_coupon",
  "provider": "softswiss",
  "games": ["softswiss:AztecMagic"],
  "version": 1650357607,
  "msg_id": "01G130X6FR19FNACZEP6EZ0ANA",
  "api_version": "0.2.0"
}
```

Field definitions

Field	Type	Description
`id`	number	Unique identifier for the free spins issue.
`client_name`	string	Name of the casino in the Casino Platform.
`operation`	string	Name of the operation with the free spins issue. The only possible value: • `update` – the free spins status has been changed.
`user_id`	number	Unique identifier for the player. This field serves as a partition key.
`stage`	string	Current free spins stage. The possible values are: • `issued` – the free spins bonus have been issued to the player. The player can either activate or cancel it. • `activated` – the player has activated the free spins bonus. • `canceled` – the player has canceled the free spins bonus. • `expired` – the free spins bonus has expired. • `finished` – the player has used all the free spins.
`currency`	string	Three-letter ISO currency code. Indicates the player's account to which the free spins bonus has been issued. For example, `"EUR"`.
`title`	string	Free spins bonus title. This title is displayed to the player.

`created_at`	string	Date and time when the free spins bonus has been issued in the ISO 8601 format (YYYY-MM-DDThh:mm:ssZ). **Note** that this field never changes. For example, in the case of data re-push, the value of this field remains intact and stores its initial value.
`activatable_until`	string|null	Date and time before which the free spins can be activated, in the ISO 8601 format (YYYY-MM-DDThh:mm:ssZ). If the free spins haven't been activated in time, they become expired. `null`, if the free spin can be activated without any time limits.
`valid_until`	string|null	Date and time before which the free spins are valid, in the ISO 8601 format (YYYY-MM-DDThh:mm:ssZ). `null`, if the free spins are permanently valid.
`freespins_total`	number	Total number of free spins that the player has.
`freespins_performed`	number|null	Number of free spins the player has already used. `null`, if the player hasn't used any free spin yet.
`strategy`	string|null	Free spins issuing strategy. The reason the player has received these free spins. The possible values are: • `deposit` – the free spins are issued to the player upon a successful deposit. • `groups_updated` – the free spins are issued to the player when they join a group. • `input_coupon` – the free spins are issued upon registration, if a player specified a promo code. • `registration` – the free spins are issued to the player immediately upon registration. • `scheduler` – the free spins are scheduled to be issued at a certain time. `null`, if there's no strategy to issue the free spins.
`provider`	string|null	Game provider. Players can use free spins in games of this provider.
`games`	array [string]	An array of slot games where the free spins can be played. Each element is a string with the game provider and title separated by a colon. For example, `"softswiss:AztecMagic"`.

`win_amount_cents`	number	Amount of money the player has won from the free spins, in the smallest currency unit. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.
`version`	number	Unix representation of `created_at`. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Random bonus

Subscribe to the `es.{casino_name}.random_bonus` Kafka topic to consume events associated with random bonuses. Event Streaming sends messages to the topic when a random bonus is issued to the player or when its status is updated.

Random bonus is issued to player

```
{
    "client_name": "Best-casino",
    "operation": "update",
    "id": 5280,
    "stage": "issued",
    "strategy": "lootbox_item",
    "title": "Lootbox on deposit with empty bonus",
    "valid_until": "2023-05-24T08: 15: 42Z",
    "created_at": "2023-05-23T08: 15: 42Z",
    "updated_at": "2023-05-23T08: 15: 42Z",
    "issue_history_id": 5283,
    "version": 1684829742,
    "user_id": 51115,
    "account_id": 53506,
    "currency": "USD",
        "api_version": "1.5.0"
    "msg_id": "01H13TV9EAYHDETP6ZKS4TP3M4"
}
```

Field definitions

Field	Type	Description
`client_name`	string	Name of the casino in the Casino Platform.
`operation`	string	Name of the operation with an empty bonus. Always `update`.
`id`	number	Unique identifier for the random bonus issue.
`user_id`	number	Unique identifier for the player.
`account_id`	number	Unique identifier for the account to which the bonus is issued.
`currency`	string	Currency of the player's account to which the bonus is issued. The currency is in three-letter ISO code.
`stage`	string	Current stage of the random bonus. Possible values: • `issued`: Bonus is issued to a player. • `activated`: Bonus is activated by a player or issued as active. • `canceled`: Player or an admin user canceled the bonus. • `expired`: Bonus is expired. Bonus money and related winnings are removed from the player's account.
`strategy`	string	Strategy via which the bonus is issued. Always `lootbox_item`.
`title`	string	Title of the random bonus.
`valid_until`	string	Date and time when the bonus expires. The date is in the UTC format.
`created_at`	string	Date and time when the random bonus was issued to the player. The date is in the UTC format.
`updated_at`	string	Date and time when the bonus stage was updated. The date is in the UTC format.
`issue_history_id`	number	Unique identifier for the bonus issue in the issue history.
`version`	number	Unix representation of `created_at`. Measured in seconds since the Unix epoch.

api_version	string	API version in the semantic versioning format (major.minor.patch).
msg_id	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id`, while `msg_id` will always be unique. You can use the latter to differentiate between the duplicates.

Empty bonus

Subscribe to the `es.{casino_name}.empty_bonus` Kafka topic to consume events associated with empty bonuses. Event Streaming sends messages to the topic when an empty bonus is issued to the player or when its status is updated.

Empty bonus is issued to player

```
{
    "id":43,
    "client_name":"Best-casino",
    "operation":"update",
    "user_id":12,
    "account_id":13,
    "currency":"EUR",
    "title":"Empty bonus",
    "created_at":"2023-05-22T11:41:48Z",
    "updated_at":"2023-05-22T11:41:48Z",
    "issue_history_id":243,
    "version":1684755708,
        "api_version": "1.5.0",
    "msg_id":"01H11M7YZZHNRJ1GFCN9KZNM9P"
}
```

Filed definitions

Field	Type	Description
id	number	Unique identifier for the empty bonus.
client_name	string	Name of the casino in the Casino Platform.
operation	string	Name of the operation with an empty bonus. Always `update`.
user_id	number	Unique identifier for the player.
account_id	number	Unique identifier for the account to which the bonus is issued.
currency	string	Currency of the player's account that received the empty bonus. Shown in three-letter ISO code.
title	string	Title of the empty bonus.
created_at	string	Date and time when the empty bonus was issued to the player.
updated_at	string	Date and time when the empty bonus was updated.
issue_history_id	number	Unique identifier for the bonus issue in the issue history.
version	number	Unix representation of `created_at`. Measured in seconds since the Unix epoch.
api_version	string	API version in the semantic versioning format (major.minor.patch).

`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id`, while `msg_id` will always be unique. You can use the latter to differentiate between the duplicates.

CompPointTransaction

Subscribe to the `es.{casino_name}.comp_point_transaction` Kafka topic to consume creations of the complimentary point transactions. Explore the message structure and field definitions below:

Example: Player received chargeable comp points

```
{
  "id": 123,
  "client_name": "Best-casino",
  "user_id": 444,
  "account_type": "chargeable",
  "points_delta": 2.0,
  "balance": 5.0,
  "target_type": "BonusIssue"
  "created_at": "2020-02-02T00:00:00Z",
  "version": 1578009600,
  "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
  "api_version": "0.3.0",
}
```

Field definitions

Field	Type	Description
`id`	number	Unique identifier for the comp point transaction.
`client_name`	string	Name of the casino in the Casino Platform.
`user_id`	number	Unique identifier for the player. This field serves as a partition key.
`account_type`	string	Type of the player's account to which the transaction has been made. The possible values are: • `chargeable`: The player can redeem this type of comp points for rewards (cash, lottery tickets, free spins, and others). • `persistent`: The player can't exchange this type of comp points. They define the level that a player achieved within an online casino.
`points_delta`	number	The value by which the player's balance has been increased. For example, if the player's balance is 5 and the transaction delta is 2, then the player's balance after the transaction will be 7.
`balance`	number	Balance of the player's account **after** the transaction has been conducted. **Note** that for each account type (chargeable and persistent) there is a separate balance value. See examples in the Comp Point balances section.
`target_type`	string	The reason for the comp point transaction. The possible values are: • `Game` - the player has received the comp points for placing a bet. • `AdminUser` - an admin user has allocated comp points to the player. • `BonusIssue` - the player has received a bonus and comp points. • `CompPointBooster` - the player has received the comp points within a Comp Points Booster campaign.
`created_at`	string	Date and time when the comp point transaction has been conducted in the ISO 8601 format (YYYY-MM-DDThh:mm:ssZ). **Note** that this field never changes. For example, in the case of data re-push, the value of this field remains intact and stores its initial value.

`version`	number	Unix representation of `created_at`. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Comp point balances

Each comp point type has a separate balance record associated with it. For example, if a transaction contains chargeable comp points, it reaches the balance for the chargeable comp points. Correspondingly, if a transaction contains persistent comp points — it reaches the balance for the persistent comp points. The balances are independent. Examine the following examples:

Example: Player received chargeable comp points

```
{
  "id": 123,
  "client_name": "Best-casino",
  "user_id": 444,
  "account_type": "chargeable",
  "points_delta": 2.0,
  "balance": 5.0,
  "target_type": "BonusIssue"
  "created_at": "2020-02-02T00:00:00Z",
  "version": 1578009600,
  "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
  "api_version": "0.3.0",
}
```

Example: Admin user allocated persistent comp points for player

```
{
  "id": 542,
  "client_name": "Best-casino",
  "user_id": 444,
  "account_type": "persistent",
  "points_delta": 7.0,
  "balance": 15.0,
  "target_type": "AdminUser"
  "created_at": "2020-02-02T00:00:00Z",
  "version": 1578009600,
  "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
  "api_version": "0.3.0",
}
```

User

Subscribe to the `es.{casino_name}.user` Kafka topic to consume events associated with the player. It can be:

- Player's registration in the casino
- Email confirmation
- Any changes in the player's personal information
- Player's duplicate status updates
- Other actions

Explore the message structure below to build a solid understanding of what player's data are written to and read from Kafka:

Example: Player registered in casino

```
{
        "user_id": 333,
        "client_name": "Best-casino",
        "operation": "create",
        "email": "john.doe@example.com",
        "first_name": "John",
        "last_name": "Doe",
        "full_name": "John Doe",
        "nickname": "Johhnys",
        "active_phone_status": "not_present",
        "user_access_limits": [{"type": "coolingoff", "created_at": "2022-
09-09T15:19:46Z"}],
         "phones": [{
                  "id":557245,
                  "phone_number":"+375297740176",
                  "updated_at":"2023-05-29T10:33:09Z",
                  "verified_at":"2023-05-29T10:33:09Z"
        }],
        "current_sign_in_ip": "111.11.11.11",
        "current_sign_in_at": "2020-01-03T00:00:00Z",
        "confirmed_at": "2020-01-03T00:00:00Z",
        "created_at": "2020-01-03T00:00:00Z",
        "state": "suspended",
        "tags": ["TT"],
         "btag": "",
        "btag_net_refer": null,
          "qtag": "",
        "stag_affiliate": "128569",
        "s_tag_visit": null,
        "subid": "subid",
        "current_sign_in_country": "DE",
        "currencies": ["EUR", "BTC"],
        "visible_currencies": ["EUR", "BTC"],
        "duplicate": false,
        "address": "Sovetskaya 33",
         "address2":"",
        "auto_issuing_bonuses": true,
        "gender": "m",
        "country": "BY",
        "city": "Minsk",
        "postal_code": "100524",
        "deposit_payment_systems": ["devcode:skrill"],
        "cashout_payment_systems": [],
        "receive_promos": true,
        "receive_sms_promos": true,
        "date_of_birth": "1990-09-01",
        "affiliate_email": "affiliate@example.com",
        "affiliate_profile_existence": false,
    "egames_status": "REGISTRED",
    "language": "ru",
    "disposable_email": true,
    "social_networks": ["facebook"],
    "device_types": ["mobile"],
    "version": 1578009600,
    "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
    "api_version": "0.2.0",
        "action": "cooling_off_created",
        "personal_id_number": "",
           "utm_campaign": null,
          "utm_content": null,
          "utm_medium": null,
          "utm_source": "",
          "utm_term": null,
         "ga_id": null
}
```

Field definitions

Field	Type	Description
`user_id`	number	Unique identifier for the player. This field serves as a partition key.
`client_name`	string	Name of the casino in the Casino Platform.
`operation`	string	Name of the operation with the player. The possible values are: • `create` – indicates that a new player has been created. • `update` – indicates that particular updates in the player's data have occured. See the details in the Operation update section below.
`email`	string	Player's email address.
`phones`	array[object]	Array of objects where each object contains information about the player's phone number, including deactivated ones. If the player doesn't have a phone number, the field is an empty array.
`phones.id`	number	Unique identifier for the phone number.
`phones.phone_number`	string	Player's phone number with a preceding plus sign (+). The phone number includes the country code.
`phones.verified_at`	string|null	Date and time when the phone number was verified, in the UTC format. The field is `null` if the phone number isn't verified.
`phones.updated_at`	string|null	Date and time when the phone number was updated, in the UTC format. This field is updated in two cases: • Phone number is added to player's profile. • Player's phone number is verified.
`first_name`	string	Player's first name.
`last_name`	string	Player's last name.
`full_name`	string	Player's full name.
`nickname`	string	Player's nickname at the casino.
`address2`	string	Second line of the player's address. In general, this string includes components like the apartment, room, floor, building, or department numbers.
`postal_code`	string	Player's postal code.
`city`	string	Player's city.
`state`	string	The state of the player's account. The possible values are: • `normal`: The player is neither blocked nor suspended, and don't has any restrictions. • `suspended`: The player is temporarily suspended. They still can withdraw money. • `disabled`: The player is blocked and can't log in the casino. To get detailed information on players' blocks, refer to the **Terminate player profile** page.
`tags`	array [string]	Tags assigned to the player. For example, the tags can be used to filter or group players.

`btag`	string	If the field is not an empty string, shows that the player has come from the Netrefer affiliate system. Otherwise, the field is an empty string.
`qtag`	string	If the field is not an empty string, shows that the player has come from the Quintessence affiliate system. Otherwise, the field is an empty string.
`stag_affiliate`	string	If the field is not an empty string, shows the unique identifier for the affiliate in the Affilka affiliate system. Otherwise, the field is an empty string.
`visible_currencies`	array [string]	An array with currencies of the player's active accounts. For example, a player can disable a particular account, if doesn't want to bet with its currency.
`duplicate`	boolean	Shows whether the player is a duplicate. A player is considered to be a duplicate if has multiple casino profiles.
`address`	string	The player's address.
`auto_issuing_bonuses`	boolean	Shows whether the auto issuing bonuses is enabled for the player.
`gender`	string	The player's gender.
`country`	string	The player's country. Two-letter ISO country code.
`deposit_payment_systems`	array [string]	An array of payment systems the player has ever used to deposit money. Each element is a string with the system name and integration type separated by a colon.
`cashout_payment_systems`	array [string]	An array of payment systems the player has ever used to withdraw money. Each element is a string with the system name and integration type separated by a colon.
`receive_promos`	boolean	Shows whether the player is subscribed to the promo emails.
`receive_sms_promos`	boolean	Shows whether the player is subscribed to the sms promo campaigns.
`date_of_birth`	string	The player's date of birth.
`affiliate_email`	string	The affiliate's email.
`affiliate_profile_existence`	boolean	Shows whether the player is an affiliate themselves.
`egames_status`	string	The player's Egames status. Concerns the casinos under Belgian license whose players have to be verified by Belgian Gaming Commission. The possible values are: • `REGISTERED` • `VALIDATED` • `MINOR_ERROR` • `MAJOR_ERROR` • `EPIS_ALERT` • `INACTIVE` Read more about Egames statuses on the **Filters** page in the **Egames status** filter condition.
`language`	string	The player's language in use. Two-letter ISO code, in lowercase.

`disposable_email`	boolean	Shows whether the player's email address is disposable. Disposable email addresses are temporal and destruct themselves after a certain time elapses.
`social_networks`	array [string]	An array of the player's social networks ever used to sign in the casino.
`device_types`	array [string]	An array of devices the player ever used to sign in the casino.
`version`	number	Unix representation of `created_at`. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).
`action`	string	The field is sent only with `operation: "update"`. For example, when a player registers, the associated message has `operation: "create"` and doesn't contain the `action` field. Event Streaming sends another message right after the first one, with `operation: "update"` and `action: "registered"`. Shows what triggered a message push to the topic. Possible values: • `disposable_checked`: Player's email is checked on being disposable. Updated fields: `disposable_email`. • `account_visibility_updated`: Player's account became hidden or visible. Updated fields: `visible_currencies`. • `account_created`: New player's account is added. Updated fields: `currencies` and `visible_currencies`. • `payment_created`: Player requested a deposit or cashout. Updated fields: `deposit_payment_systems` or `cashout_payment_systems`. • `payment_manual_created`: Admin user made a manual payment. Updated fields: `deposit_payment_systems` or `cashout_payment_systems`. • `payment_canceled`: Player's payment was canceled. Updated fields: `deposit_payment_systems` or `cashout_payment_systems`. • `payment_deposit_created`: Payment system processed a deposit successfully. Updated fields: `deposit_payment_systems`. • `payment_cashout_created`: Payment system processed a cashout successfully. Updated fields: `cashout_payment_systems` field is updated. • `user_limit_updated`: Player's limits are updated. Updated fields: `user_access_limits`. • `self_exclusion_created`: Player set the 'Self-exclusion' limit. Updated fields: `user_access_limits`. • `cooling_off_created`: The 'Cooling-off' limit is imposed on the player. Updated fields: `user_access_limits`. • `registered`: Player registered in the casino. Updated fields: `current_sign_in_country` and `current_sign_in_ip`. • `signup_confirmed`: Player confirmed their email. Updated fields: `confirmed_at`. • `social_network_created`: Player removed a social network. Updated fields: `social_networks`. • `social_network_updated`: Player added a new social network. Updated fields: `social_networks`. • `egames_status_changed`: Player's egames status is updated. Updated fields: `egames_status`. • `affiliate_profile_created`: Player is marked as an affiliate. Updated fields: `affiliate_profile_existence`. • `session_created`: Player logged in to the casino. Updated fields: `current_sign_in_country`, `current_sign_in_ip`, `device_types`, and `current_sign_in_at`. • `verified_phone_created`: Player provided a phone number. Updated fields: `active_phone_status` and `phones`. • `verified_phone_verified`: Player verified their phone number. Updated fields: `active_phone_status` and `phones`. • `resource_repusher`: The User historical data is exported via the Export topic.

`personal_id_number`	string	Player's personal ID number. Passport or ID card number. Filled in case of special requirements of the licensee country.
`user_access_limits`	array [object]	Array of objects where each object contains information about player's access limits, including the limit type and creation date in the UTC format. The array can have up to two limit types: `coolingoff` and `selfexclusion`. To get detailed information on the limits and their types, refer to the **Set limits** page.
`created_at`	string	Date and time when the player was registered in the casino. Shown in the UTC format.
`utm_source`	string	Source that is sending traffic to the casino website. The source can be a social network, search engine, newsletter name, and more.
`ga_id`	string|null	Unique identifier in the UUID format that is associated with the player's browser or device via which they opened the casino website. The identifier is returned only if the Google Analytics feature is available at the casino.
`utm_medium`	string|null	Type of traffic the player originated from.
`utm_campaign`	string|null	A name of a campaign. This could be the product name, a contest name, a code to identify a specific sale or promotion, an influencer ID or a tagline.
`utm_content`	string|null	Name of different ads within the same campaign.
`utm_term`	string|null	Paid search keywords or key phrases.
`s_tag_visit`	string|null	Unique identification code connected with an affiliate player who came into the casino through an 'Affilka' affiliate platform link. If `null`, the player didn't use an affiliate link to come to the casino.
`btag_net_refer`	string|null	Unique identification code connected with an affiliate player who came into the casino through a 'Netrefer' affiliate platform link. If `null`, the player didn't use an affiliate link to come to the casino.

Operation update

When a new player is created, we produce a message that contains all the fields described above. If an event is associated with some updates in the player's data, and not with creation, the message will contain only those fields that have been updated + the following static fields:

- `user_id`
- `client_name`
- `operation`
- `version`
- `msg_id`
- `api_version`

Let's assume that the player has requested the 'Self-exclusion' limit to take a break. When an admin user accepts the request, we'll publish the following message to Kafka:

Example: Limit is imposed on player

```
{
        "client_name": "Best-casino",
        "operation": "update",
        "user_access_limits": [{"type": "selfexclusion", "created_at":
"2022-09-09T15:19:46Z"}], # Limit imposed
    "user_id": 15,
    "version": 1640682537,
    "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
    "api_version": "0.1.0",
        "action": "cooling_off_created"
}
```

When the limit expires, we'll publish another message of the following structure:

Example: Limit expired

```
{
        "client_name": "Best-casino",
    "operation": "update",
    "user_access_limits": [], # Array is empty – the player has no active
limits
    "user_id": 15,
    "version": 1640682537,
    "msg_id": "T3DDEIFAAK6F90MRSN4QVSR1ZR",
    "api_version": "0.1.0",
        "action": "user_limit_updated"
}
```

Payment

Subscribe to the `es.{casino_name}.payment` Kafka topic to consume events associated with the player's payments. A payment can be:

- Deposit
- Cashout
- Refund
- Reversal
- Chargeback
- Affiliate payment

Note, that payment assumes the player's balance updates and triggers a corresponding `BalanceTransaction` event. Thus when a payment occurs, we send two messages – one into the `es.{casino_name}.payment` topic and the second to the `es.{casino_name}.balance_transaction`.

Player made a deposit

```
{
    "id": 189656,
    "client_name": "Best-casino",
    "operation": "update",
    "user_id": 407574,
        "account_id": "11231",
    "action": "deposit",
        "ftd": "true",
        "ftd_attempt": false,
        "business_event": "deposit_succeed"
    "currency": "EUR",
    "amount": 4400,
    "amount_cents": 4400,
    "status": "success",
    "user_country": "FI",
    "payment_system_data": {
        "payment_system_id": 24,
        "aggregator": "trustly",
        "child_system": "base",
        "integration_type": null,
        "payment_method": null,
        "psp_service": "Skrill",
        "psp_brand": "Skrill",
        "payment_code": 21312343,
        "psp_status_code": null,
        "psp_status_message": null
    },
    "bonus_code": "PROMO10",
    "commission_amount_cents": 0,
        "created_at": "2023-06-05T14:16:58Z",
        "updated_at": "2023-06-05T14:16:58Z",
        "balance_transaction_created_at": "2022-04-20T15:52:24Z",
    "manual": false,
        # The example shows a successful payment. In real situation, value
of the external_error field would be null.
        # Here, the value is just for the example purpose
        "external_error": "status_code: ERR_DECLINED_BANK_REFUSAL,
psp_status_code: 1, psp_status_message: AUTH 3DAUTH ERROR Issuer Bank
Error",
    "processing": false,
    "recalled": false,
    "crypto_data": {},
    "version": 1650469944,
    "msg_id": "01G13TSFFYN7X5G4P1BC2WGVYB",
    "api_version": "1.2.0"
}
```

Field definitions

Field	Type	Description
`id`	number	Unique identifier for the payment.
`client_name`	string	Name of the casino in the Casino Platform.

`operation`	string	Operation type for the payment. Possible values: • `create`: The payment is created (just created, not finished yet). • `update`: The payment status is updated. The `operation` field of the following payment types always has the `update` value: • Crypto payment • Manual payment • Reversal • Refund • Chargeback The message structure is always the same regardles of the `operation`'s value.
`user_id`	number	Unique identifier for the player. This field serves as a partition key.
`account_id`	number	Unique identifier for the player's account subjected to the payment.
`action`	string	Type of the payment. The possible types are: • `deposit`: The player deposited money. • `cashout`: The player made a cashout. • `chargeback`: The player received a deposit chargeback. • `reversal`: The payment system reversed the player's cashout. • `affiliate_payment`: The player's affiliate received the money. • `refund`: The casino operator refunded money to the player.
`currency`	string	Currency of the payment operation. Three-letter ISO currency code.
`amount_cents`	number	Payment amount in the smallest currency unit. One currency unit consists of a fixed number of its smallest units. For most fiat currencies, 1000 smallest units make 1 currency unit. For example, 1000 smallest eur units make 10 euro (1000 euro cents). For cryptocurrencies, the logic is slightly different. Each cryptocurrency has a different number of the smallest units to make 1 whole unit. In the list below, you'll see the supported cryptocurrencies, their smallest units, and how many of the smallest units are needed to compose 1 whole unit. • 1 Bitcoin (BTC) = 100,000,000 of Satoshi. • 1 Bitcoin Cash (BCH) = 100,000,000 of Satoshi. • 1 Litecoin (LTC) = 100,000,000 of Litoshi. • 1 Peercoin (PPC) = 100,000,000 of its smallest units. • 1 Dogecoin (DOG) = 100,000,000 of Koinu. • 1 Ethereum (ETH) = 1,000,000,000 of Gwei. • 1 Tether (USDT) = 100,000,000 of its smallest unit. • 1 Ripple (XRP) = 1,000,000 of Drop. • 1 TRON (TRX) = 1,000,000 of SUN. • 1 Binance (BNB) = 100,000,000 of Jager. • 1 AVCcoin (AVC) = 100,000,000 of its smallest unit. • 1 Basic Attention Token (BAT) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Bitcoin Satoshi's Vision (BSV) = 100,000,000 of its smallest unit. • 1 Dai (DAI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Polkadot (DOT) = 100,000,000 of Planck. • 1 Enjin (ENJ) = 1,000,000,000,000,000,000 of its smallest unit. • 1 EOS = 100,000,000 of its smallest unit. • 1 Chainlink (LINK) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Decentraland (MANA) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Turtle (TRTL) = 100,000,000 of its smallest unit. • 1 Uniswap token (UNI) = 1,000,000,000,000,000,000 of its smallest unit. • 1 Vietnemese Stable Coin Pegged to the Dong (VNDC) = 100,000,000 of its smallest unit. • 1 V Systems (VSYS) = 100,000,000 of its smallest unit. • 1 Stellar (XLM) = 10,000,000 of Stroop. • 1 Monero (XMR) = 1,000,000,000,000 of Piconero. • 1 Netbox (NBX) = 100,000,000 of its smallest unit.
`manual`	boolean	If `true`, the payment is manually conducetd by the casino operator. Otherwise, `false`.

`processing`	boolean	Shows whether the payment is still being processed. Relevant for non-instant operations, for example, with the fiat currencies.
`recalled`	boolean	If `true`, the payment has been recalled. Otherwise, `false`. The player can cancel their cashout request. A cashout cancellation can only be requested while on the 'Pending' or 'Ready to Process' statuses.
`crypto_data`	object	The field is only relevant for payment via the CoinsPaid and UTORG payment system. In case of a crypto payment, this field contains additional information about the payment. If the payment is made with a fiat currency, this object is empty (`crypto_data: {}`). Expand an example of a crypto payment below:

Example of crypto payment

```
{
   "id": 189656,
   "client_name": "Best-casino",
   "operation": "update",
   "user_id": 407574,
   "action": "deposit",
   "currency": "BTC",
   "amount": 4400,
   "amount_cents": 4400,
   "status": "success",
   "user_country": "FI",
   "payment_system_data": {
      "payment_system_id": 24,
      "aggregator": "coinspaid",
      "child_system": "base",
      "integration_type": null,
      "payment_method": null
   },
   "created_at": "2022-04-20T15:52:01Z",
   "balance_transaction_created_at": "2022-04-20T15:
52:24Z",
   "manual": false,
   "processing": false,
   "recalled": false,
   "crypto_data": {
      "wallet_currency": "BTC",
      "crypto_tx_id": "158ad874628fab3cf6b0362ccbf0"
   },
   "version": 1650469944,
   "msg_id": "01G13TSFFYN7X5G4P1BC2WGVYB",
   "api_version": "0.3.0"
}
```

`crypto_data.wallet_currency`	string	Currency of the payment in three-letter ISO currency code.
`crypto_data.crypto_tx_id`	string	Unique identifier for the crypto payment. For example, `"158ad874628fab3cf64ad29a40"`.
`crypto_data.tx_address`	stirng	Unique identifier for the crypto wallet to which the player made a cashout. `null`, if the payment isn't a cashout
`crypto_data.original_currency`	string	Original currency in case of a crypto-to-fiat payment.

`status`		Status of the payment. The possible statuses are: • `success` – indicates that the payment has been made successfully. • `pending` – indicates that the payment is pending. It's the case for standard bank transfer operations that imply additional processings. Note, that while payment is pending, a `BalanceTransaction` event isn't fired as there is no any balance updates yet. • `failure` – indicates that the payment has failed. **Note**, that for a crypto transaction there's either `success` or `failure` status.
`balance_transaction_created_at`	string|null	Date and time when the payment has been successfully finished and the player's balance has been updated in UTC (YYYY-MM-DDThh:mm:ssZ). While the payment's `status` is `pending` this field is `null`.
`user_country`	string|null	The player's country. Two-letter ISO country code, in uppercase. Can be `null`.
`created_at`	string	Date and time when the payment has been made. Shown in the UTC format. **Note** that this field never changes. For example, in the case of data re-push, the value of this field remains intact and stores its initial value.
`updated_at`	string	Date and time when the payment has been updated. Shown in the UTC format.
`payment_system_data`	object	Contains information on the payment system used for this payment.
`payment_system_data.payment_system_id`	number|null	Unique identifier for the payment system. `null`, if the payment system unique identifier is undefined.
`payment_system_data.aggregator`	string	Name of the payment aggregator.
`payment_system_data.child_system`	string	Name of the aggregator's child payment system that is processing the transaction.
`payment_system_data.integration_type`	string|null	Integration to which the payment method refers to. Can be `null`.
`payment_system_data.payment_method`	string|null	Name of the payment method. Can be `null`.
`payment_system_data.psp_service`	string	Optional payment service used when a payment aggregator has several underlying services.
`payment_system_data.psp_brand`	string	Name of the payment provider.
`payment_system_data.psp_status_code`	string	HTTP status code of the payment.

payment_system.data.psp_status_message	string	User-friendly message describing the payment status.
payment_code	string	Unique identifier for the payment on the payment system's side.
bonus_code	string	Bonus code the player entered via a coupon. Applicable for the deposit bonuses. If player doesn't use the coupon, the bonus_code field is null.
commission_amount_cents	number	Amount of the fee that the payment system charges in the smallest currency unit.
version	number	Date and time when the payment has been created or updated. Measured in seconds since the Unix epoch.
msg_id	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same id, while msg_id will always be unique, so you can use the latter to differentiate between the duplicates.
api_version	string	API version in the semantic versioning format (major.minor.patch).
business_event	string	Payment type and its status separated by an underscore (_). Possible values: • cashout_canceled: Payment system canceled a cashout. • cashout_created: Player requested a cashout. • cashout_failed: Admin user canceled a cashout. • manual_cashout_succeed: Admin user made a manual cashout. • cashout_succeed: Player cashed out money successfully. • chargeback_canceled: Payment system canceled a chargeback request. • chargeback_created: Player requested a chargeback. • deposit_created: Player made a deposit, and a payment system is processing it. • deposit_failed: Payment system canceled a deposit. • manual_deposit_succeed: Admin user made a manual deposit. • deposit_succeed: Player deposited money successfully. • refund_canceled: Payment system canceled a refund request. • refund_created: Casino operator requested a refund. • reversal_canceled: Payment system canceled a reversal. • reversal_created: Payment system created a reversal of a cashout. • event_streamed: Payment topic is exported.
ftd_attempt	boolean	Shows whether the payment is the player's first deposit attempt. • If true, the payment is the player's first deposit attempt. • If false, the player already tried to deposit before the current deposit. The statuses of the previous deposits are not considered for the ftd_attempt field. If you want to check whether there're any **successful** deposits before the current one, see the ftd field. If the payment is not a deposit or is a manual deposit made by an admin user, the field is null.

`ftd`	boolean	If the payment is a deposit, the field shows the following:
		
		• If `true`, the player doesn't have previous deposits with the `status` field set to `success`.
		• If `false`, the player has at least one previous deposit with the `status` field set to `success`.
		
		1. A player makes the first deposit. The `status` of the deposit is `pending`. The `ftd` field is `true` because there're no previous successful deposits. The `ftd_attempt` field is `true` because the current deposit is the very first player's deposit.
		
		`# Fields for the first deposit`
		`{`
		`        "status": "pending",`
		`        "ftd_attempt": true,`
		`        "ftd": true`
		`}`
		
		2. The player makes the second deposit. The `status` of the deposit is `success`. The `ftd` field is `true` because there're no previous **successful** deposits (the first deposit is in pending status). The `ftd_attempt` field is `false` because player already tried to deposit.
		
		`# Fields for the second deposit`
		`{`
		`        "status": "success",`
		`        "ftd_attempt": false,`
		`        "ftd": true`
		`}`
		
		3. The player makes the third deposit. The `status` of the deposit is `failure`. The `ftd` field is `false` because a previous deposit is successful. The `ftd_attempt` field is `false` because player already made two deposits.
		
		`# Fields for the third deposit`
		`{`
		`        "status": "failure",`
		`         "ftd_attempt": false,`
		`        "ftd": false`
		`}`
		
		When admin user confirms the player's very first deposit, the deposit status is changed to `success`.
		
		`# Fields for the first deposit`
		`{`
		`        "status": "success",`
		`        "ftd_attempt": true,`
		`        "ftd": true`
		`}`
		
		The `ftd` and `ftd_attempt` fields of a payment always keep the initial value regardless of the payment status changes.
		
		For all the following deposits, both `ftd` and `ftd_attempt` will be set to `false` because there're already at least two successful deposits.
		
		If the payment is not a deposit or is a manual deposit made by an admin user, the field is `null`.

`external_error`	string|null	Information about the error if the payment fails on the payment system's side.

Sportsbook bets

Subscribe to the `es.{casino_name}.sportsbook_bet` Kafka topic to consume updates of a Sportsbook bet status. Explore the message structure and field definitions below:

Example: Player placed bet in Sportsbook

```
{
   "version":1646295148,
   "api_version":"0.3.0"
   "client_name":"Best-bets",
   "quick":false,
   "bet_id":"111a2233-4444-555b-6c6c-7788dd99ee1b",
   "bet_type":"single",
   "total_odds":2300,
   "odds_precision":3,
   "stake_value":4760,
   "stake_precision":2,
   "stake_currency":"EUR",
   "stake_eur_value":4760,
   "stake_eur_precision":2,
   "bet_result":"win",
   "potential_win_amount_value":10948,
   "potential_win_amount_eur_value":10948,
   "win_amount_value":10948,
   "win_amount_eur_value":10948,
   "win_amount_without_cashout_value":0,
   "win_amount_without_cashout_eur_value":0,
   "cashed_out":false,
   "accepted_at":"2022-03-03T07:03:50.123456Z",
   "settled_at":"2022-03-03T08:13:08.123456Z",
   "margin":107,
   "margin_precision":2,
   "system":"",
   "events_count":1,
   "odds_type":"european",
   "bonus": {},
   "customer": {
      "id":744740,
      "player_id":3321,
      "login":"john.doe@exmaple.com",
      "gender":"male",
      "date_of_birth":"2003-06-14T00:00:00Z"
      "country":"BY",
      "registered_at":"2021-10-13T03:41:32.123456Z",
      "ip":"111.222.333.44",
      "device":"desktop",
      "verified":false,
   },
   "selections": [
      {
         "producer":"live",
         "event_id":486016275,
         "sport_id":3,
         "sport_name":"Basketball",
         "category_id":352380,
         "category_name":"International",
         "tournament_id":484376585,
         "tournament_name":"FIBA World Cup Women",
         "market_id": 196,
         "market_name":"Handicap (incl. overtime)",
         "outcome_id":500,
         "outcome_name":"{$competitor2} ({-hcp})",
         "played_outcome_id":501,
         "played_outcome_name":"Mali (w) (+16.5)",
         "team_home_name":"Serbia (w)",
         "team_home_country_code":"SRB",
         "team_home_country_name":"Serbia",
         "team_away_name":"Mali (w)",
         "team_away_country_code":"MLI",
         "team_away_country_name":"Mali",
         "game_period":"3rd period",
         "score_away":43,
         "score_home":55,
         "odds":2300
      }
   ],
        "msg_id": "01G13TSFFYN7X5G4P1BC2WGVYB"
}
```

Field definitions

Field	Type	Description
`version`	string	Unix representation of the snapshot. Measured in seconds since the Unix epoch.
`client_name`	string	Owner of the Sportsbook project.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).
`quick`	boolean	Shows whether the bet is quick. Quick bets let skip the stake specifying and accepting.
`bet_id`	string	Unique identifier for the Sportsbook bet in the UUID format.
`bet_type`	string	Bet type. Possible values are: • `single` — bet in which a player is betting on a single outcome of an event. • `combo` — bet in which a player is betting on two games and more. • `system` — advanced version of the `combo` bet where not all selections have to be successful for bets to be winning ones.
`total_odds`	number	Total odd of the bet. To avoid floating-point numbers, the total odds are multiplied by 1000. The multiplier is defined in the `odds_precision` field. For example, instead of 1.65 odds, you will see 1650 (1.65 × 1000).
`odds_precision`	number	Shows the power of 10. $10^{\texttt{odds\_precision}}$ — number of the smallest currency unit making up a whole currency unit. For example, if `odds_precision` is 3, the odds will be multiplied by 1000 (10^3 = 1000).
`stake_value`	number	Stake amount in the smallest currency unit.
`stake_precision`	number	Shows the power of 10. $10^{\texttt{stake\_precision}}$ — number of the smallest currency unit making up a whole currency unit. For example, if `stake_precision` is 2, the odds will be multiplied by 100 (10^2 = 100). Depending on the stake currency `stake_precision` can vary. For example, for USD it's 2, while for BTC is 8.
`stake_currency`	string	Currency of the stake in three-letter ISO currency code.
`stake_eur_value`	number	Amount of the stake converted to EUR in the smallest currency unit.
`stake_eur_precision`	number	Always 2 meaning that 1 eur = 10^2 euro cents.
`bet_result`	string	Bet result. Possible values: • `no_results` — bet wasn't settled yet. • `win` — player won the bet. • `lose` — player lost the bet. • `refund` — bet amount was refunded to the player. • `cancel` — bet was canceled. Bet amount returned to the player.

`potential_win_amount_value`	number	Amount of money the player can potentially win. Calculated as *stake_value* × *total_odds*. Measured in the smallest currency unit.
`potential_win_amount_eur_value`	number	Amount of money the player can potentially win converted to EUR. Calculated as *stake_eur_value* × *total_odds*. Measured in the smallest currency unit.
`win_amount_value`	number	Amount of money the player won in the smallest currency unit. If the bet isn't settled yet, this field is `0`.
`win_amount_eur_value`	number	Amount of money the player won converted to EUR. If the bet isn't settled yet, this field is `0`. Measured in the smallest currency unit.
`win_amount_without_cashout_value`	number	Amount of money the player wins if they don't cashout during the match (an early cashout). This field is `0` unless the bet is settled.
`win_amount_without_cashout_eur_value`	number	Amount of money the player wins if they don't cashout during the match (an early cashout). Converted to EUR.
`cashed_out`	boolean	Shows whether the player made an early cashout before the bet had been settled.
`accepted_at`	string	Date and time when the bet was accepted by the Sportsbook in the UTC format. Static field.
`settled_at`	string	Date and time when the bet was settled in the UTC format. The field value can change in case of a rollback.
`margin`	number	The margin is the difference between the odds set by the operator, and the true probability of the outcome occurring. Operator's profit. To get actual percentage divide `margin` by $10^{\texttt{margin\_precision}}$. For example, if `margin` is 110 and `margin_precision` is 2, the actual percentage is 10.
`margin_precision`	number	Always 2 meaning that to get actual margin percentage you need to divide `margin` by 10^2.
`system`	string	Shows 2 digits separated by an underscore (_). The first digit shows the number of outcomes required for a win. The second digit shows the total number of outcomes in the system. The field is empty unless `bet_type: system`.
`events_count`	number	Number of the unique events in the bet.
`odds_type`	string	Odds type. Possible values are: • `european` • `british` • `hongkong` • `american` • `indonesian` • `malaysian` You can read about every type on the **Sportsbook FAQ** page.
`bonus`	object	Contains information on the bonus used for the bet. If the player doesn't use a bonus, this field is an empty object: `{}`.
`bonus.id`	string	Unique identifier for the bonus in the UUID format.

`bonus.type`	string	Bonus type. Possible values: • `hunting` • `freebet_no_risk` • `freebet_all_win` • `freebet_only_win` • `comboboost` • `lootbox` You can read about every type on the **Sportsbook FAQ** page.
`bonus.odds`	number	Total bonus odds of the bet. The field is empty unless `bonus.type: "comboboost"`.
`bonus.amount_value`	number	Amount of the bonus money the player won. Depending on the bonus type this field is calculated differently. If the player used the Comboboost, this field shows amount of money the player win with this bonus. • If the player used the Freebet All Win bonus, this field shows the sum of the stake and win. • If the player used the Freebet Only Win bonus, this field shows the winning amount, excluding the stake value. • If the player used the Freebet No Risk bonus and won, this field shows the sum of the stake and win. • If the player used the Freebet No Risk bonus and lost, this field shows the stake amount only. The values are measured in the smallest currency unit.
`bonus.amount_eur_value`	number	Amount of the bonus money converted to EUR. Measured in the smallest currency unit.
`bonus.potential_amount_value`	number	Maximum amount of money the player can win using the bonus. Measured in the smallest currency unit.
`bonus.potential_amount_eur_value`	number	Maximum amount of money the player can win using the bonus converted to EUR. Measured in euro cents.
`bonus.freebet_nominal_amount_value`	number	Amount of a freebet bonus in the smallest currency unit.
`bonus.freebet_nominal_amount_eur_value`	number	Amount of a freebet bonus converted to EUR. Measured in euro cents.
`customer`	object	Contains information on the player.
`customer.id`	number	Unique identifier for player used in the Sportsbook.
`customer.player_id`	number	Unique identifier for the player used in the Casino Platform.
`customer.login`	string	Player's email address.
`customer.gender`	string	Player's gender. Possible values: • `male` • `female` • `unknown`
`customer.date_of_birth`	string	Player's date of birth in the UTC format. The time is always `00:00:00`.
`customer.country`	string	Player's country in two-letter ISO country code. For example, `"IT"`.

`customer.registered_at`	string	Date and time when the player registered in Sportsbook in the UTC format.
`customer.ip`	string	IP of the player when they placed the bet.
`customer.device`	string	Device of the player when they placed the bet. Possible values: • `desktop` • `tablet` • `mobile` • `unknown` (all other devices)
`customer.verified`	string	Shows whether the player is verified.
`selections`	array	Lists outcomes on which the player can place a bet.
`selections.producer`	string	Selection producer. Possible values: • `live` — selections for the ongoing events. • `prematch` — selections for the planned events. • `premium_cricket` — selections for cricket events. Retrieved from Betradar.
`selections.event_id`	number	Unique identifier for the event. Event is a planned and organized occasion. Also known as game or match.
`selections.sport_id`	number	Unique identifier for the sport.
`selections.sport_name`	string	Sport name. For example, `"Tennis"`.
`selections.category_id`	number	Unique identifier for the sport category.
`selections.category_name`	string	Category name. For example, for soccer it can be `"Argentina"` and for tennis — `"Davis Cup"`.
`selections.tournament_id`	string	Unique identifier of the tournament.
`selections.tournament_name`	string	Tournament title. For example, for soccer it can be `"Premier League"` and for tennis — `"Billie Jean King Cup"`.
`selections.market_id`	number	Unique identifier for the market. Market is a specific type or category of bets with odds related to each.
`selections.market_name`	string	Market name.
`selections.outcome_id`	number	Unique identifier for the outcome the player selected.
`selections.outcome_name`	string	Name of the outcome the player selected.
`selections.played_outcome_id`	number	Unique identifier for the actual outcome.
`selections.played_outcome_name`	string	Name of the actual outcome. The field is empty until the bet is settled.
`selections.team_home_name`	string	Name of the home team.
`selections.team_home_country_code`	string	Three-letter ISO country code of the home team.

`selections.team_home_country_name`	string	Country of the home team.
`selections.team_away_name`	string	Name of the away team.
`selections.team_away_country_code`	string	Three-letter ISO country code of the away team.
`selections.team_away_country_name`	string	Country of the away team.
`selections.game_period`	string	Period of the event at the time of the bet. If the event hasn't started yet, the field is empty.
`selections.score_away`	number	Score of the away team.
`selections.score_home`	number	Score of the home team.
`selections.odds`	number	Selection odds.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). The duplicates have a unique `msg_id` , so you can use it to differentiate between the duplicates.

Group

Subscribe to the `es.{casino_name}.group` Kafka topic to consume updates of a user group. For example, to see which group is created or updated. Explore the message structure and field definitions below:

Example: Group updated

```
{
        "id": 15,
        "client_name": "Best-casino",
        "operation": "update",
        "name": "Mega VIP",
        "enabled": true,
        "version": 1578009600,
        "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
        "api_version": "1.0.0"
}
```

Field definitions

Field	Type	Description
`id`	number	Unique identifier for the group.
`client_name`	string	Name of the casino in the Casino Platform.
`operation`	string	Name of the operation with the group. The possible values are: • `create`: New user groups has been created. • `update`: Existing user groups has been updated. The attributes that trigger an update are `name` and `enabled`. The message structure is always the same regardless of the `opertion`'s value.
`name`	string	Name of the group.

`enabled`	boolean	Shows whether the group is enabled or not.
`version`	number	Date and time when the group was created or updated. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

User group

Group feed is stored in Kafka's `es.{casino_name}.user_group` topic. When a player joins or leaves a user group, a message is produced to this topic. Explore the message structure and field definitions below:

Example: Player's group updated

```
{
        "client_name": "Best-casino",
        "user_id": 4574,
        "group_ids": [21, 444],
        "version": 1578009600,
        "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
        "api_version": "1.0.0"
}
```

Field definitions

Field	Type	Description
`client_name`	string	Name of the casino in the Casino Platform.
`user_id`	number	Unique identifier for the player. This field serves as a partition key.
`group_ids`	array [number]	Lists identifier for the user groups to which the player belongs.
`version`	number	Date and time when the player joined or left the group. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Game

Game feed is stored in Kafka's `es.{casino_name}.game` topic. When a new game is imported to the casino from the Game Aggregator or allowed countries of an existing game are updated, a message is produced to this topic. Explore the message structure and field definitions below:

Example: New game is added to casino

```
{
        "id": 14,
        "operation": "update",
        "provider": "storygaming",
        "category": "slots",
        "feature_group": "basic",
        "released_at": "2022-12-06",
        "recalled_at": null,
        "variation": "ProfessorClanksCombinator"
        "identifier": "storygaming:ProfessorClanksCombinator",
        "title": "Professor Clank's Combinator",
        "producer": "reelplay",
        "allowed_countries": ["US", "PL"],
        "client_name": "Best-casino",
        "version": 1578009600,
        "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
        "api_version": "1.0.0"
}
```

Field definitions

Field	Type	Description
`client_name`	string	Name of the casino in the Casino Platform.
`id`	number	Unique numeric identifier for the game on the casino side.
`operation`	string	Name of the operation with the game. The possible values are: • `create`: Game is imported to the casino. • `update`: Game's allowed countries have been updated. The message structure is always the same regardless of the `operation`'s value.
`identifier`	string	Unique identifier for the game. Includes the game provider and variation.
`title`	string	Game title.
`provider`	string	Game provider.
`released_at`	string	Date when the game is released in the format of YYYY-MM-DD. If `null`, the game is under set-up. Players can't launch it yet.
`recalled_at`	string	Date when the game is recalled in the format of YYYY-MM-DD. If a game is recalled, players can't launch it. If `null`, the game is enabled and is available for players.
`category`	string	Game category.
`live`	boolean	If `true`, the game is live; otherwise `false`. Live games are streamed in real time. Real dealers interact with the players.
`feature_group`	string	Name of the game fee group. Fee group defines the amount of money a casino operator pays as a fee to a game provider
`producer`	string	Game producer. The game producer developed the game while the game provider distributes it.

variation	string	Game variation. It implies changes in a game that modifies the player's user experience through new features and opportunities. Also, the game variation goes after the forward slash in the `identifier` field.
allowed_countries	array [string]	Lists countries where the game is allowed to play. Each country is in two-letter ISO country code.
version	number	Date and time when the game was created or updated. Measured in seconds since the Unix epoch.
msg_id	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
api_version	string	API version in the semantic versioning format (major.minor.patch).

Game category

Game category feed is stored in Kafka's `es.{casino_name}.game_category` topic. Any updates associated with a game category are produced to this topic. Explore the message structure and field definitions below:

Example: 'slots' category is updated

```
{
        "id": 15,
        "operation": "update",
        "identifier": "slots",
        "games": ["softswiss:PlatinumLightning", "softswiss:CherryFiesta"],
        "disabled": false,
        "client_name": "Best-casino",
        "version": 1578009600,
        "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
        "api_version": "1.0.0"
}
```

Field definitions

Field	Type	Description
client_name	string	Name of the casino in the Casino Platform.
id	number	Unique numeric identifier for the game category.
operation	string	Name of the operation with the game. The possible values are: • `create`: Game category has been created in the Casino Platform. • `update`: Game category has been updated. The message structure is always the same regardless of the `operation`'s value.
identifier	string	Unique identifier for the game's category.
games	array [string]	Lists identifier for the games belonging to the game category. Each game includes the game provider and variation. Game variation implies changes in a game that modifies the player's user experience by means of new features and opportunities.
disabled	boolean	Shows whether the game category is enabled or not. Disabled game categories aren't shown on the casino site, so players can't launch them.

`version`	number	Date and time when the game category was created or updated. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Export

The `es.{casino_name}.export` topic produces historical data occurred in your casino before you integrated Event Streaming. All events, regardless of their topic, are written on this channel.

The data are produced upon request. So, to consume these data:

1. Tell your account manager which data and for which period you want.
   The account manager forwards your request to the dev team, and they launch the data flow.
2. Subsribe to the `es.{casino_name}.export` channel and read the messages.

The message structure contains a static `topic` field and other fields depending on the event. The `topic` field is a string with the following possible values:

- `balance_transaction`
- `bonus_issue`
- `comp_point_transaction`
- `freespin_issue`
- `game`
- `game_category`
- `group`
- `payment`
- `user`
- `user_group`

A message of a historical event doesn't include the `operation` field.

The Export topic produces one message per event entity. An entity is a balance transaction, a game, a bonus, a free spin, and others. At the moment of export, you have the latest state of the requested entity.

Important

The Export topic doesn't include intermediate states of an entity. For example, status changes of a bonus aren't exported. Only the latest state of the bonus data is exported.

Export examples

See examples of exported data for each topic:

User

```
{
        "action": "user_export",
        "active_phone_status": "unverified",
        "address": "55555 Deonna Werr",
        "affiliate_email": "",
        "affiliate_profile_existence": false,
        "api_version": "1.2.0",
        "auto_issuing_bonuses": true,
        "btag": "",
        "cashout_payment_systems": [],
        "client_name": "Best-casino",
        "confirmed_at": null,
        "country": "FI",
        "created_at": "2022-09-28T14:18:48Z",
        "currencies": ["EUR"],
        "current_sign_in_at": "2022-09-28T14:18:49Z",
        "current_sign_in_country": "DE",
        "current_sign_in_ip": "162.55.92.109",
        "date_of_birth": "1986-02-10",
        "deposit_payment_systems": [],
        "device_types": ["desktop"],
        "disposable_email": false,
        "duplicate": false,
        "egames_status": "",
        "email": "alex.enigma@example.com",
        "gender": "m",
        "language": "en",
        "msg_id": "01GR4J1CW3KH53BNKS9TGAJHE9",
        "personal_id_number": null,
        "qtag": "",
        "receive_promos": false,
        "receive_sms_promos": false,
        "social_networks": [],
        "stag_affiliate": "",
        "state": "normal",
        "subid": "",
        "tags": ["VIP", "golden_status"],
        "topic": "user",
        "user_access_limits": [],
        "user_id": 579848,
        "version": 1674961869,
        "visible_currencies": ["EUR"]
}
```

Balance transaction (bonus issue)

```
{
        "id": 11078266,
        "client_name": "Best-casino",
        "user_id": 663706,
        "account_id": 700276,
        "reference_id": 730700,
        "reference_type": "BonusIssue",
        "action": "issued_bonus",
        "currency": "EUR",
        "amount_cents": 10000,
        "bonus_amount_cents": 0,
        "balance": 10000,
        "balance_before": null,
        "created_at": "2023-01-31T17:41:36Z",
        "game_info": {},
        "bonus": {},
        "version": 1675186896,
        "msg_id": "01GR635WTN3VQ5FZ4VFWB7WFMP",
        "api_version": "1.2.0",
        "topic": "balance_transaction"
}
```

Balance transaction (game)

```
{
        "id": 11078216,
        "client_name": "Best-casino",
        "user_id": 663681,
        "account_id": 700247,
        "reference_id": 7460968,
        "reference_type": "Game",
        "action": "rollback_win",
        "currency": "BTC",
        "amount_cents": -4555059,
        "bonus_amount_cents": 0,
        "balance": 979939,
        "balance_before": 20135,
        "created_at": "2023-01-31T17:38:36Z",
        "game_info": {
                "game_table_id": 39512,
                "name": "softswiss:AztecMagic",
                "categories": ["slots"],
                "tx_number": 6
        },
        "bonus": {},
        "version": 1675186716,
        "msg_id": "01GR635WMFYPWW66093K7WCQQ4",
        "api_version": "1.2.0",
        "topic": "balance_transaction"
}
```

Payment

```
{
        "id": 338201,
        "client_name": "Best-casino",
        "user_id": 663991,
        "action": "deposit",
        "currency": "EUR",
        "amount_cents": 4200,
        "status": "success",
        "user_country": "FI",
        "payment_code": "16911810475",
        "bonus_code": null,
        "commission_amount_cents": 0,
        "payment_system_data": {
                "payment_system_id": 23,
                "aggregator": "pay_n_play",
                "child_system": "base",
                "integration_type": null,
                "payment_method": null,
                "psp_service": null,
                "psp_status_code": null,
                "psp_status_message": null,
                "psp_brand": ""
        },
        "created_at": "2023-01-31T18:24:55Z",
        "balance_transaction_created_at": "2023-01-31T18:24:55Z",
        "manual": false,
        "processing": false,
        "recalled": false,
        "crypto_data": { "tx_address": "address_101" },
        "version": 1675189495,
        "msg_id": "01GR62XRB1D39EYJ2JF5EWSKXQ",
        "api_version": "1.2.0",
        "topic": "payment"
}
```

Group

```
{
        "client_name": "Best-casino",
        "id": 8266,
        "name": "VIP_players",
        "enabled": true,
        "version": 1674956169,
        "msg_id": "01GR620FEPEBGA78WAVC6HV54K",
        "api_version": "1.2.0",
        "topic": "group"
}
```

User group

```
{
        "client_name": "Best-casino",
        "user_id": 664404,
        "group_ids": [
                7212, 7219, 7226, 7233, 7240, 7253
        ],
        "version": 1675196409,
        "msg_id": "01GR6307NBMYH1JSQ0FDVBXFRC",
        "api_version": "1.2.0",
        "topic": "user_group"
}
```

Game category

```
{
        "id": 2176,
        "client_name": "Best-casino",
        "identifier": "poker",
        "games": [
                "softswiss:TexasHoldem",
                "softswiss:CaribbeanPoker",
                "softswiss:CasinoHoldem",
                "softswiss:LetItRide",
                "softswiss:OasisPoker",
                "softswiss:TreyPoker"
        ],
        "disabled": true,
        "version": 1675186983,
        "msg_id": "01GR62JJXZ76RS5SFZ08708B0K",
        "api_version": "1.2.0",
        "topic": "game_category"
}
```

Game

```
{
        "id": 8758,
        "client_name": "Best-casino",
        "identifier": "1x2gaming:Shostak",
        "title": "Shostak",
        "producer": "1x2gaming",
        "allowed_countries": [
                "AD",
                "AE",
                "AG",
                "AI",
                "AL",
                "AM",
                "AO",
                "AQ",
                "AR",
                "AS",
                "AT",
                "AW",
                "AX",
                "AZ",
                "BA"
        ],
        "provider": "1x2gaming",
        "variation": "Shostak",
        "released_at": null,
        "recalled_at": null,
        "category": "slots",
        "live": false,
        "feature_group": "basic",
        "version": 1674129512,
        "msg_id": "01GR62Q49KNKZSRQFDP8DJN17A",
        "api_version": "1.2.0",
        "topic": "game"
}
```

Bonus issue

```
{
        "id": 730231,
        "client_name": "Best-casino",
        "user_id": 663359,
        "stage": "handle_bets",
        "title": "Welcome bonus EUR",
        "currency": "EUR",
        "created_at": "2023-01-31T03:18:31Z",
        "updated_at": "2023-01-31T03:18:31Z",
        "activatable_until": null,
        "valid_until": "2023-02-28T03:18:31Z",
        "amount_cents": 10000,
        "amount_wager_requirement_cents": 10000,
        "amount_wager_cents": 0,
        "strategy": "registration",
        "payment_id": null,
        "freespin_id": null,
        "account_id": 699921,
        "version": 1675135111,
        "msg_id": "01GR62SFFR2W6VYWNM5EQHT6SE",
        "api_version": "1.2.0",
        "topic": "bonus_issue"
}
```

Free spin issue

```
{
        "id": 34970,
        "client_name": "Best-casino",
        "user_id": 662493,
        "stage": "issued",
        "title": "Registration Free Spins with bonus code",
        "currency": "EUR",
        "created_at": "2023-01-30T03:04:44Z",
        "activatable_until": "2023-02-06T03:04:44Z",
        "valid_until": null,
        "freespins_total": 20,
        "freespins_performed": null,
        "win_amount_cents": 0,
        "strategy": "registration",
        "provider": "softswiss",
        "games": [
                "softswiss:AztecMagic",
                "softswiss:AztecMagicDeluxe",
                "softswiss:BobsCoffeeShop",
                "softswiss:BookOfPyramids",
                "softswiss:BraveViking"
        ],
        "version": 1675047884,
        "msg_id": "01GR62W2W3CJ5T07TCY5752S21",
        "api_version": "1.2.0",
        "topic": "freespin_issue"
}
```

Comp points transaction

```
{
        "id": 506571,
        "client_name": "Best-casino",
        "user_id": 653414,
        "account_type": "persistent",
        "target_type": "AdminUser",
        "points_delta": 669,
        "balance": 669,
        "created_at": "2023-01-20T10:39:28Z",
        "version": 1674211168,
        "msg_id": "01GR63348DPC1TJ8CCHZRN90MC",
        "api_version": "1.2.0",
        "topic": "comp_point_transaction"
}
```

Dossier

Subscribe to the `es.{casino_name}.dossier` Kafka topic to consume the following predicted criteria:

- Lifetime value
- Churn rate
- Active days
- Deposit probability

Lifetime value
Player's lifetime value (LTV) is a predicted amount of revenue the player can generate during their presence in the casino. Event Streaming starts sending players' LTV on the 6th day after their registration. The prediction is sent daily. If a player placed their latest bet more than 30 days ago, Event Streaming stops observing the player and sending their data to the topic until the next bet.

You can use these predictions to arrange promo campaigns and offer players custom bonuses based on their loyalty.

Example: Player's LTV

```
{

   "client_name": "Best-casino",
   "user_id": 123,
   "criteria": "ltv_class"
   "value": "1-500€"
   "version": 1578009600,
   "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
   "api_version": "1.4.0"

}
```

Churn rate

The Churn rate criteria shows the probability with which the player may leave the casino within the next 30 days. Event Streaming daily sends information on players who made at least 1 deposit within the latest 30 days.

Knowing the players' churn rates, you can arrange custom promo campaigns to engage the low-activity players and reward loyal ones.

Example: Player's churn rate

```
{
   "client_name": "Best-casino",
   "user_id": 123,
   "criteria": "churn",
   "value": 0.7,
   "version": 1578009600,
   "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
   "api_version": "1.4.0"
}
```

Active days

The Active days criteria shows a predicted number of days the player will stay active within the next 30 days. Players are considered active if they place bets. For example, if a player placed at least one bet within a day, they're considered active for this day. By default, Event Streaming starts sending the prediction on the 2nd day after their registration. The starting point is configurable, so you can ask your account manager to set a custom value.

The prediction is sent daily. If a player placed their latest bet more than 30 days ago, Event Streaming stops observing the player and sending their data to the topic until the next bet.

Example: Player's churn rate

```
{
   "client_name": "Best-casino",
   "user_id": 123,
   "criteria": "active_days",
   "value": 7,
   "version": 1578009600,
   "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
   "api_version": "1.4.0"
}
```

Deposit probability

The Deposit criteria shows the probability with which the player may make at least one deposit within the next 30 days. Event Streaming starts sending the prediction on the 2nd day after their registration. The prediction is sent daily. If a player placed their latest bet more than 30 days ago, Event Streaming stops observing the player and sending their data to the topic until the next bet.

Example: Player's churn rate

```
{
   "client_name": "Best-casino",
   "user_id": 123,
   "criteria": "deposit",
   "value": 0.5,
   "version": 1578009600,
   "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
   "api_version": "1.4.0"
}
```

Field definitions

Field	Type	Description
`client_name`	string	Name of the casino in the Casino Platform
`user_id`	number	Unique identifier for the player. This field serves as a partition key.
`criteria`	string	Dossier's message scope. Possible values: • `churn`: The `value` field shows the probability of player's leave from the casino. • `ltv_class`: The `value` field shows the player's lifetime value for the casino. • `active_days`: The `value` field shows a predicted number of days the player will stay active within next 30 days. • `deposit`: The value field shows the probability of player's deposit within the next 30 days.
`value`	number|string	Predicted value. Depending on the `criteria` field the values range and its meaning varies. **When `criteria` is `churn`** Probability of player's leaving the casino where `0` is the lowest chance and `1` is the highest. **When `criteria` is `ltv_class`** Predicted amount of the player's LTV. Calculated as the difference between successful deposits and cashouts. Possible values: • `Negative LTV`: The player is likely to produce negative LTV. Sometimes, negative revenue is an indicator that the player performs fraudulent actions like bonus abuse. • `0€` • `1-500€` • `500-2500€` • `>2500€` All currencies are converted to EUR based on exchange rates. **When `criteria` is `active_days`** Predicted number of days the player will stay active within the next 30 days. The range is from 0 upto 30 days. A player is considered active if they place bets. **When `criteria` is `deposit`** Probability with which the player may deposit within the next 30 days where `0.0` is the lowest chance and `1.0` is the highest.
`version`	number	Shows when Event Streaming sent the message. Measured in seconds since the Unix epoch.

`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

User session

Subscribe to the `es.{casino_name}.user_session` Kafka topic to consume information on the user session. Event Streaming sends messages to this channel, when the player logs in to the casino or logs out.

Example: User session closed

```
{
        "client_name": "Best-casino",
        "id": 312,
        "user_id": 554433,
        "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36",
        "device_type": "desktop",
        "os": "macOS",
        "country": "AU",
        "ip": "116.202.58.241",
        "closed_at": "2022-09-09T15:19:46Z",
        "created_at": "2022-09-09T15:19:46Z",
        "updated_at": "2022-09-09T15:19:46Z",
         "version": 123123123,
        "msg_id": "MGKSORJFGLSJDKSN23LS",
        "api_version": "1.2.0"
}
```

Field definitions

Field	Type	Description
`client_name`	string	Name of the casino in the Casino Platform.
`id`	number	Unique identifier for the user session.
`user_id`	number	Unique identifier for the player. This field serves as a partition key.
`user_agent`	string	Browser from which the player logged in to the casino.
`created_at`	string	Date and time when the player logged in to the casino and the player's session started, in UTC.
`updated_at`	string	Date and time with when the player's session updated, in UTC. This field is updated only twice: • When the session started. • When the session ended.
`closed_at`	string	Date and time when the user session ended, in UTC. `null`, if the session isn't closed yet.
`version`	number	Unix representation of `created_at`. Measured in seconds since the Unix epoch.

`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).
`device_type`	string	Type of the device from which the player signed in to the casino. Possible values: • `desktop` • `tablet` • `mobile` If an empty string, the Casino Platform couldn't identify the player's device.
`country`	string	ISO code of the country from which the player signed in to the casino. If `null`, the Casino Platform couldn't identify the player's country.
`os`	string	Name of the operation system of the player's device that they used to sign in to the casino. If an empty string, the Casino Platform couldn't identify the player's operating system.
`ip`	string	Player's IP address.

Account

Subscribe to the `es.{casino_name}.account` Kafka topic to consume changes of the player's accounts. Event Streaming sends messages to this channel when the player creates a new account.

Example: Player created EUR account

```
{
  "id": 11,
  "created_at": "2020-01-01T19:00:00",
  "currency": "EUR",
  "user_id": 55,
  "operation": "create",
  "client_name": "Best-casino",
  "version": 123123123,
  "msg_id": "01ARZ3NDEKTSV4RRFFQ69ASDMI",
  "api_version": "1.4.0"
}
```

Fields definition

Field	Type	Description
`client_name`	string	Name of the casino in the Casino Platform
`id`	number	Unique identifier of the player's account.
`user_id`	number	Unique identifier for the player. This field serves as a partition key.
`currency`	string	ISO currency code of the newly created account.
`created_at`	string	Date and time when the player created the account with `currency`.
`operation`	string	Name of the operation with the account. Possible values: • `create`: The player created a new account in `currency`.

`version`	number	Shows when Event Streaming sent the message. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id` , while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Account limits

Subscribe to the `es.{casino_name}.account_limit` Kafka topic to consume events associated with the limits set on the player's accounts. Event Streaming sends messages to the topic in the following cases:

- A new limit is set.
- An existing limit is updated.
- An existing limit is canceled.

The limits can be set for the deposit, wager, and loss transactions.

Existing limit is updated

```
{
  "id": 17830,
  "account_id": 197097,
  "user_limit_id": 235678,
  "amount_cents": 5000000,
  "current_period": "2023-06-15",
  "current_value_amount_cents": -294062,
  "client_name": "Best-casino",
  "operation": "update",
  "msg_id": "01H2ZF0QERMKV5X3DQ5VTB1GJG"
}
```

Fields definitions

Field	Type	Description
`client_name`	string	Name of the casino in the Casino Platform.
`operation`	string	Name of the operation with the account limit. Possible values: • `create`: A new limit is imposed on the player's account. • `update`: An existing limit is updated or disabled.
`account_id`	number	Unique identifier for the player's account where a limit is created, updated, or disabled.
`user_limit_id`	number	Unique identifier for the limit.
`amount_cents`	number	Limit amount in the smallest currency unit.
`current_period`	string	Date and time when the limit expires.
`current_value_amount_cents`	number	Amount of money the player has already spent within the limit.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker timeouts). The `msg_id` is always unique.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Tournament

Before connecting to the topic, ask your technical account manager to enable the `export_enabled` and `player_export_enabled` settings. The link is available to the SOFTSWISS employees only.

Subscribe to the `es.{casino_name}.tournament` topic to consume events associated with the casino's tournaments. Event Streaming produces messages to this topic when a new tournament is created or an existing tournament is updated.

Example: Tournament is updated

```
{
        "casino_name": "Best-casino",
        "operation": "update",
        "id": 9359,
        "title": "Best tournament",
        "enabled": false,
        "user_confirmation_required": false,
        "frontend_identifier": "best_tournament",
        "strategy": "bet",
        "currency_settings": [
                {
                        "currency": "AUD",
                        "min_bet_cents": 0,
                        "max_bet_cents": null,
                        "active": false
                }
        ],
        "game_category_identity": "slots",
        "start_at": "2023-02-16T03:00:00Z",
        "end_at": "2023-02-16T11:00:00Z",
        "finished_at": "2023-02-16T11:00:26Z",
        "updated_at": "2023-02-16T11:00:26Z",
        "currency": "USD",
        "start_money_budget_cents": 1000,
        "max_money_budget_cents": 10000000,
        "bet_percent": 10.0,
        "start_chargeable_micropoints_budget": 1000000,
        "max_chargeable_micropoints_budget": 0,
        "start_persistent_micropoints_budget": 0,
        "max_persistent_micropoints_budget": 0,
        "chargeable_comp_points_percent": 1.0,
        "persistent_comp_points_percent": 0.0,
        "spins_step": null,
        "award_automatically": true,
        "extra": {
                "bet_points": 0,
                "points_rules": []
        },
        "force_recalculate_before_finish": false,
        "games_taken_limit": null,
        "group_ids": [57, 56],
        "group_tournament": false,
        "masking_rate": 1.0,
        "one_point_per_multiplier": false,
        "only_real_bets": false,
        "recurring_shift_period": "hours",
        "recurring_shift_period_count": 4,
        "version": 1676545226,
        "msg_id": "01GSCY3ZD069QFW23KC7YC0BA4",
        "api_version": "1.4.0"
}
```

Field definitions

Field	Type	Description
`casino_name`	string	Name of the casino in the Casino Platform.

`operation`	string	Operation type for the tournament. Possible values: • create: A new tournament is created. • update: An existing tournament is updated.
`id`	number	Unique identifier for the tournament.
`title`	string	Tournament title.
`enabled`	boolean	Tournament status. • If `true`, the tournament is enabled. • If `false`, the tournament is disabled.
`user_confirmation_required`	boolean	Shows whether players are required to confirm their participation in the tournament. • If `true`, the players must confirm their participation via the **POST** `/api/tournaments/{id}/confirm` endpoint. • If `false`, manual confirmation isn't required. Players become participants once they place a bet in a tournament game.
`frontend_identifier`	string	Unique frontend identifier for the tournament. This identifier is used on the casino site as a URL slug. As a rule, it's similar to the tournament title.
`strategy`	string	Strategy that is used to select the tournament winners. possible values: • `bet`: The winner is the player with the biggest sum of bets. • `win`: The winner is the player with the highest total of wins. • `rate`: The winner is the player with the highest ratio of win amount to bet amount. The spins_step field is taken into account when calculating the ratio. • `spin`: The winner is the player who placed the most bets, regardless of bet size. • `points`: The winner is the player who accumulated the most comp points during the tournament. Go to the Help Center to learn more about each strategy.
`currency_settings`	array [object]	An array of objects where each object contains information about the currency supported in the tournament.
`currency_settings.currency`	string	Currency supported in the tournament, in ISO code.
`currency_settings.min_bet_cents`	number	Minimum amount of currency available for a bet in the tournament.
`currency_settings.max_bet_cents`	number	Maximum amount of currency available for a bet in the tournament.
`currency_settings.active`	boolean	Shows whether the currency is active. If `true`, player's can use the currency for bets in the tournament. Otherwise, `false`.
`game_category_identity`	string	Game category that include the games available in the tournament.
`start_at`	string	Start date and time of the tournament in the UTC format.
`end_at`	string	End date and time of the tournament in the UTC format.
`finished_at`	string	Finish date and time of the tournament in the UTC format. A tournament is finished when players get rewards.
`updated_at`	string	Date and time of the latest tournament update.
`currency`	string	Tournament main currency in ISO code. It's also the currency for prize calculations. For example, a player places bets in USD, and the tournament currency is EUR. The player gets the USD prize at the EUR exchange rate.

`start_money_budget_cents`	number	Money a winner gets in addition to the main prize. Shown in the smallest currency unit.
`max_money_budget_cents`	number	The maximum amount of money a winner can get in addition to the main prize. Shown in the smallest currency unit.
`bet_percent`	number	Percentage of each bet placed that is added to the budget. Shown as a float number. For example, `start_money_budget_cents` is set to `5000`. `bet_percent` is set to `10.0`. The player makes a bet of `100`. The money budget increases by `10`.
`start_chargeable_micropoints_budget`	number	Amount of chargeable micro Comp Points a winner gets in addition to the main prize. `1000000` micro Comp Points = `1` Comp Point.
`max_chargeable_micropoints_budget`	number	Maximum amount of chargeable micro Comp Points a winner can get in addition to the main prize. `1000000` micro Comp Points = `1` Comp Point.
`start_persistent_micropoints_budget`	number	Amount of persistent micro Comp Points a winner gets in addition to the main prize. `1000000` micro Comp Points = `1` Comp Point.
`max_persistent_micropoints_budget`	number	Maximum amount of persistent micro Comp Points a winner can get in addition to the main prize. `1000000` micro Comp Points = `1` Comp Point.
`chargeable_comp_points_percent`	number	Percentage of each chargeable Comp Points earned that are added to the budget. For example, `start_chargeable_micropoints_budget` is set to `1000000`. `chargeable_comp_points_percent` is set to `1.0` (1%). The player places a bet and earns `5000000` chargeable micro comp points. The budget increases by `50000` chargeable micro comp points.
`persistent_comp_points_percent`	number	Percentage of each status or persistent comp points earned that are added to the budget. For example, `start_persistent_micropoints_budget` is set to `1000000`. `persistent_comp_points_percent` is set to `1.0` (1%). The player places a bet and earns `5000000` persistent micro comp points. The budget increases by `50000` persistent micro comp points.
`spins_step`	number|null	The number of spins a player has to make for the bet to be saved in the tournament. Applicabble for the tournaments with the `strategy` field set to `rate`. The field is `null` for other strategies.
`award_automatically`	boolean	Shows whether the awards are given to the winners automatically. • If `true`, winners will be automatically awarded when the tournament ends. • If `false`, winners are manually awarded by the admin users.
`extra`	object	Contains information about rules for earning tournament points. Relevant for the tournaments with the `strategy` field set to `points`.
`extra.bet_points`	number	Number of tournament points a player gets for every bet placed.
`extra.points_rules`	array [object]	Contains information about custom points rules.
`extra.points_rules.multiplier`	string	Shows in how many times the win should exceed the initial bet to receive the points specified in the `extra.points_rules.points` field.

extra.points_rules.points	string	Number of points a player gets if the number in the `extra.points_rules.multiplier` field is reached.
force_recalculate_before_finish	boolean	Shows whether the prize recalculation will be forced before the tournament finish. Admin users can force a disabled tournament to finish.
games_taken_limit	number	Number of spins a player can make within the given tournament, while it's active. Applicable for the tournaments with the `strategy` field is set to `spin`. The field is `null` for other strategies.
group_ids	array [number]	An array of numbers where each number is a unique identifier for the player's group who can participate in the tournament.
group_tournament	boolean	Shows whether it's a group tournament. • If `true`, it's a group tournament. Teams of players compete against each other. • If `false`, it's not a group tournament. Players compete against one another.
masking_rate	number[float]	Shows a multiplier by which the players' statistics are recalculated and sent to the frontend side. This recalculation lets the casino hide actual values from the players making it difficult for them to figure out the calculation formula for winners. The following player's statistics are recalculated: • Bets number • Bets amount • Wins number • Wins amount • Games taken or spins made **Example** Given that `masking_rate` is `1.7`. A player places a single bet of 3500 eurocents and wins 1000 eurocents. The values sent to the frontend side are the following: • Bets number: 1.7 (1 × 1.7) • Bets amount: 5950 (3500 × 1.7) • Wins number: 1.7 (1 × 1.7) • Wins amount: 1700 (1000 × 1.7) • Games taken or spins made: 1.7 (1 × 1.7)
one_point_per_multiplier	boolean	If `true`, a participant gets a number of points equal to the number of multipliers after each bet. The multiplier and the number of points for 1 bet are calculated by the following formula: *win/bet*. For example, a player bets $10 and wins $50. 50/10=5 points earned after one bet. The points are summed up when the tournament ends.
only_real_bets	boolean	Shows whether the participants can use the FUN currency for bets. • If `true`, participants can only use real money. • If `false`, articiopants can also use the FUN currency.
recurring_shift_period	string	Tournament recurring period. A new tournament starts, after the winners of the previous tournament receive their award. Possible values: • `hours` • `days` • `weeks` • `months`
recurring_shift_period_count	number	Number of recurring periods defined in `recurring_shift_period`.

version	number	Date and time when the tournament has been created. Measured in seconds since the Unix epoch.
msg_id	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same id, while msg_id will always be unique, so you can use the latter to differentiate between the duplicates.
api_version	string	API version in the semantic versioning format (major.minor.patch).

Tournaments: Player update

Before connecting to the topic, ask your technical account manager to enable the export_enabled and player_export_enabled settings. The link is available to the SOFTSWISS employees only.

Subscribe to the es.{casino_name}.tournament_player topic to consume tournament events associated with the player updates. Event Streaming writes messages to this topic in the following cases:

- A new player is added to the tournament by confirming the participation or joining a user group.
- A player places a bet in the tournament.
- A player changes their position on the leaderboard.

Player placed a bet

```
{
  "casino_name": "Best-casino",
  "user_id": 674513,
  "tournament_id": 19404,
  "tournament_team_id": null,
  "user_confirmed": true,
  "rate": 0.0,
  "games_taken": 1,
  "bet_mcents": 100000,
  "win_mcents": 0,
  "points": 0,
  "created_at": "2023-02-16T10:45:15Z",
  "updated_at": "2023-02-16T10:45:15Z",
  "winner": false,
  "operation": "update",
  "version": 1676544315,
  "msg_id": "01GSCX83XZ5KK3SCQT2JYXQXBP",
  "api_version": "1.4.0"
}
```

Field definitions

Field	Type	Description
casino_name	string	Name of the casino in the Casino Platform.
operation	string	Title of the operation in the tournament event. Always update.
user_id	number	Unique identifier for the player.
tournament_id	number	Unique identifier for the tournament.
tournament_team_id	number|null	Unique identifier for the team if it's a team tournament. Otherwise, null.
user_confirmed	boolean	Shows whether the player confirmed their participation in the tournament.

`rate`	number[float]	Player's win-to-bet ratio.
`games_taken`	number	The number of games or spins the player played.
`bet_mcents`	number	Total amount of money the player bet in the tournament, in the smallest currency unit.
`win_mcents`	number	Total amount of money the player won in the tournament, in the smallest currency unit.
`points`	number	Total number of points the player received within the tournament.
`created_at`	string	Date and time when the tournament player was created.
`updated_at`	string	Date and time when the tournament player was updated latest time.
`winner`	boolean	Shows whether the player is a winner. • If `true`, the player is a winner and will get the prize. • If `false`, the player isn't a winner and won't get the prize.
`version`	number	Date and time when the tournament has been created. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id`, while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Tournaments: Team updates

Before connecting to the topic, ask your technical account manager to enable the `export_enabled` and `player_export_enabled` settings. The link is available to the SOFTSWISS employees only.

Subscribe to the `es.{casino_name}.tournament_team` topic to consume tournament events associated with the team updates. writes messages to this topic in the following cases:

- A new team is created.
- An existing team's bet, win, or points amount is updated.
- Admin user removed an existing team from the tournament.

New team is added to tournament

```
{
  "casino_name":"Best-casino",
  "operation":"create",
  "id":119,
  "tournament_id":700,
  "title":"Mega team",
  "rate":0.0,
  "games_taken":0,
  "bets_cents":0,
  "wins_cents":0,
  "currency":"EUR",
  "points":0,
  "chargeable_comp_points":"0.0",
  "persistent_comp_points":"0.0",
  "created_at":"2023-02-07T10:55:00Z",
  "updated_at":"2023-02-07T10:55:00Z",
  "version":1675767300,
  "msg_id":"01GRNR7H6489HH0YNMZSG7WS2Y",
  "api_version": "1.4.0"
}
```

Field definitions

Property	Type	Description
`casino_name`	string	Name of the casino in the Casino Platform.
`operation`	string	Operation type for the tournament. Possible values: • `create`: A new team is added to the tournament. • `update`: Information about an existing team is updated. • `delete`: An admin user removed an existing team from the tournament.
`id`	number	Unique identifier for the tournament team.
`tournament_id`	number	Unique identifier for the tournament.
`title`	string	Team title.
`rate`	number[float]	Team's total ratio of win amount to bet amount. Calculated out of all spin steps.
`games_taken`	number	Total number of games or spins the team members played.
`bet_mcents`	number	Total amount of money the team bet in the tournament, in the smallest currency unit.
`win_mcents`	number	Total amount of money the team won in the tournament, in the smallest currency unit.
`points`	number	Total number of points the team received within the tournament.
`currency`	string	Tournaments currency in ISO format.
`chargeable_comp_points`	string	Total number of chargeable or redeemable Comp Points earned by the team. Shown as a decimal number.
`persistent_comp_points`	string	Total number of status or persistent Comp Points earned by the team. Shown as a decimal number.
`created_at`	string	Date and time when the team was created in the UTC format.
`updated_at`	string	Date and time when the teams was updated in the UTC format. The field's value is changed in the following cases: • When the team is created. • When the team's information is updated. • When the team is removed from the tournament.
`version`	number	Date and time when the tournament has been created. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id`, while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Tournaments: Player's win

Before connecting to the topic, ask your technical account manager to enable the `export_enabled` and `player_export_enabled` settings. The link is available to the SOFTSWISS employees only.

Subscribe to the `es.{casino_name}.tournament_player_win` topic to consume tournament events associated with the players' wins. Event Streaming writes messages to this topic when a player becomes a winner in a tournament.

Player's team won

```
{
  "casino_name": "Best-caisno",
  "user_id": 972,
  "tournament_id": 1210,
  "tournament_team_id": 495,
  "winner": true,
  "version": 1680844154,
  "msg_id": "01GXDKKCG24X6YVN6QG0V7GGFT",
  "api_version": "1.4.0"
}
```

Field definitions

Property	Type	Description
`casino_name`	string	Name of the casino in the Casino Platform.
`user_id`	number	Unique identifier for the player.
`tournament_id`	number	Unique identifier for the tournament.
`tournament_team_id`	number	Unique identifier for the tournament team. The filed is `null` if it isn't a team tournament.
`winner`	boolean	Shows whether the player is a winner. • If `true`, the player is a winner and gets a prize. • If `false`, the player isn't a winner and doesn't get a prize.
`version`	number	Date and time when the message was sent. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id`, while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Tournaments: Player confirmation

Before connecting to the topic, ask your technical account manager to enable the `export_enabled` and `player_export_enabled` settings. The link is available to the SOFTSWISS employees only.

Subscribe to the `es.{casino_name}.tournament_player_confirm` topic to consume information about the players who confirmed or rejected their participation in the tournament.

Player confirmed their participation

```
{
  "casino_name": "Best-casino",
  "user_id": 704765,
  "tournament_id": 22589,
  "tournament_team_id": null,
  "user_confirmed": true,
  "version": 1680746827,
  "msg_id": "01GXA52M3VGRS6X9FAKT5BC6YM",
  "api_version": "1.4.0"
}
```

Field definitions

Property	Type	Description
`casino_name`	string	Name of the casino in the Casino Platform.
`user_id`	number	Unique identifier for the player.
`tournament_id`	number	Unique identifier for the tournament.
`tournament_team_id`	number|null	Unique identifier for the team. The field is \`null\` if it's not a team tournament.
`user_confirmed`	boolean	Shows whether the player confirmed their particiation in the tournament. • If `true`, the player confirmed their participation in the tournament. • If `false`, the player rejected their participation in the tournament.
`version`	number	Date and time when the message was sent. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id`, while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.
`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Tournaments: Prize

Before connecting to the topic, ask your technical account manager to enable the `export_enabled` and `player_export_enabled` settings. The link is available to the SOFTSWISS employees only.

Subscribe to the `es.{casino_name}.prize` topic to consume information about tournament prizes.

Player received a prize

```
{
  "casino_name": "Best-casino",
  "operation": "update",
  "id": 106919,
  "reference_type": "Tournament",
  "reference_id": 1250,
  "user_id": null,
  "place": 22,
  "stuff": "",
  "money_budget_percent": "0.0",
  "money_award_cents": 0,
  "money_total_cents": 0,
  "currency": "EUR",
  "wager_multiplier": 20,
  "chargeable_comp_points_percent": "0.0",
  "chargeable_comp_points": "0.0",
  "chargeable_comp_points_total": "0.0",
  "persistent_comp_points_percent": "0.0",
  "persistent_comp_points": "0.0",
  "persistent_comp_points_total": "0.0",
  "freespins_count": 10,
  "bonus_group_key": "award5_fs",
  "issue_history_id": null,
  "user_notified": false,
  "created_at": "2023-05-04T08:39:23Z",
  "updated_at": "2023-05-04T08:39:23Z",
  "version": 1683189563,
  "msg_id": "01H06M7585M2R9TX2Q1152Z6TM",
  "api_version": "1.4.0"
}
```

Field definitions

Property	Type	Description
`casino_name`	string	Name of the casino in the Casino Platform.
`operation`	string	Operation type for the tournament. Possible values: • `create`: A new prize is added to the prize list. • `update`: An existing prize is updated. • `delete`: An existing prize is removed from the prize list.
`id`	number	Unique identifier for the prize.
`reference_type`	string	Type of the event a winner gets the prize in. Always `Tournament`.
`reference_id`	number	Unique identifier for the tournament.
`user_id`	number	Unique identifier for the player who received the prize. The field is `null` if there's no winner of the prize.
`place`	number	Place a player must take to get the prize.
`stuff`	string	Name of a physical prize a winner gets.
`money_budget_percent`	string	Percentage of the budget money the winner of the prize gets in addition to `money_award_cents`. The percentage money can't exceed tournament max budget. Check the `max_money_budget_cents` field in the `es.{casino_name}.tournament` topic.

`money_award_cents`	number	Fixed amount of money the winner of the prize gets.
`money_total_cents`	number	Total amount of money the winner of the prize gets. Calculated by the following formula: *`money_award_cents` + (tournament budget / `100` × `money_budget_percent`)*
`currency`	string	Currency in which the winner gets the prize. It's also the main tournament currency. Shown in ISO code.
`wager_multiplier`	number	Wagering requirement the winner must satisfy before cashing out the winning amount.
`chargeable_comp_points_percent`	string	Percentage of the chargeable Comp points budget the winner of the prize gets in addition to `chargeable_comp_points`. The percentage Comp points can't exceed tournament max budget for the chargeable Comp points. Check the `max_chargeable_micropoints_budget` field in the `es.{casino_name}.tournament` topic.
`chargeable_comp_points`	string	Fixed number of chargeable Comp points the winner of the prize gets. Shown as a decimal number.
`chargeable_comp_points_total`	string	Total number of chargeable comp poitns the winner of the prize gets. Calculated by the following formula: *`chargeable_comp_points` + (tournament starting budget for chargeable comp points / `100` × `chargeable_comp_points_percent`)*
`persistent_comp_points_percent`	string	Percentage of the persistent comp points budget the winner of the prize gets in addition to `persistent_comp_points`. The percentage comp points can't exceed tournament max budget for the status comp points. Check the `max_persistent_micropoints_budget` field in the `es.{casino_name}.tournament` topic.
`persistent_comp_points`	string	Fixed number of status comp points the winner of the prize gets. Shown as a decimal number.
`persistent_comp_points_total`	string	Total number of status comp poitns the winner of the prize gets. Calculated by the following formula: *`persistent_comp_points` + (tournament starting budget for status comp points / `100` × `persistent_comp_points_percent`)*
`freespins_count`	number	Number of free spins the winner of the prize gets.
`bonus_group_key`	string	Bonus identifier in Bonus DSL. The winner of the prize gets the bonus.
`issue_history_id`	number	Unique identifier for the bonus issue. The field is `null` if the prize doesn't contain bonuses or there's no winner for the prize.
`user_notified`	boolean	Shows whether the player is norified of the prize. Players can be notified of the prizes via **POST** `/api/prizes/{id}/notify_user`. If `true`, the player is notified. Otherwise, `false`.
`created_at`	string	Date and time when the prize is created, in the UTC format.
`updated_at`	string	Date and time when the prize is updated, in the UTC format.
`version`	number	Date and time when the message was sent. Measured in seconds since the Unix epoch.
`msg_id`	string	Unique alphanumeric identifier for the Kafka message. The stream can potentially produce duplicated messages (for example, if the broker times out). Both duplicates will have the same `id`, while `msg_id` will always be unique, so you can use the latter to differentiate between the duplicates.

`api_version`	string	API version in the semantic versioning format (major.minor.patch).

Please rate this page: