Juster v1 Core

Stepan Naumov
ztepler@gmail.com

Alexander Rakunov
ar@baking-bad.org

Abstract

This paper covers Juster v1 core protocol architecture, including common event rules, actors' roles, funds distribution formulas, price defining mechanisms, oracle interaction, event steps and states, and emergency event quitting conditions. It also includes different possible implementations and planned token functions and economy.

1. Introduction

Juster is an on-chain smart contract platform allowing users to take part in an automated betting market by creating events[1], providing liquidity to them, and making bets. Juster v1 is the first version of the platform that allows to create and make bets on crypto pairs price changes in a certain time interval ("XTZUSD price will rise by 10% or more in the following week" or "the ETHUSD price will decrease in the next hour"), based on the oracle data feed. This is an original solution that came from the #10 Challenges at the Tezos Decentralized Finance Hackathon (February 26-28, 2021), organized by Tezos Ukraine, mentored by Baking Bad and Madfish teams, and supported by TQ Tezos, Tezos Commons, and Tezos Foundation.

[1] Event - Juster contract data structure, consisting of a bunch of parameters and associated with some off-chain event through an on-chain oracle's data feed.

Juster v1 protocol is useful for traders, hedgers, bakers and tez holders in various cases, for example:

- **Case 1.** XTZ price has reached a new high in a short time and someone who runs a Tezos public baker decides to cash some part of his tez holdings; however, their funds are mostly blocked in security deposits and it could take a relatively long time to unblock them. By using Juster v1 protocol, the baker can make a sufficient hedging bet on XTZ price fall putting down a comparatively small amount of funds;
- **Case 2.** Some tez holders used to earn ordinary staking rewards, delegating their funds to public bakers. Using Juster v1 protocol, the holder can get additional yield returns by getting fees for continuously providing liquidity for events.

It's important to note that the use of that kind of option-like instruments[2] gives a significant leverage effect that leads to much higher potential returns and a higher degree of risk to users' capital, even in the case of low market volatility. This feature may be a great advantage versus ordinary DEXs' instruments; however, it must be used with special caution and is not suitable for newbies on financial markets and crypto or users with an overactive betting behavior.

2. Basic principles

Every event has two possible outcomes:

- the event ended as described (positive): S;
- the event ended contrary to the description (negative): $\overline{S}$.

Every outcome is provided by the underlying liquidity pool which represents the source of payouts for users who bet on the winning pool. Both sides' liquidity pools are formed symmetrically by liquidity providers.

The liquidity pools protocol is based on the proven "***constant product formula***" [2], used in a vast bunch of well-known DEXs such as Uniswap and QuipuSwap, and which was re-invented for a betting protocol implementation.

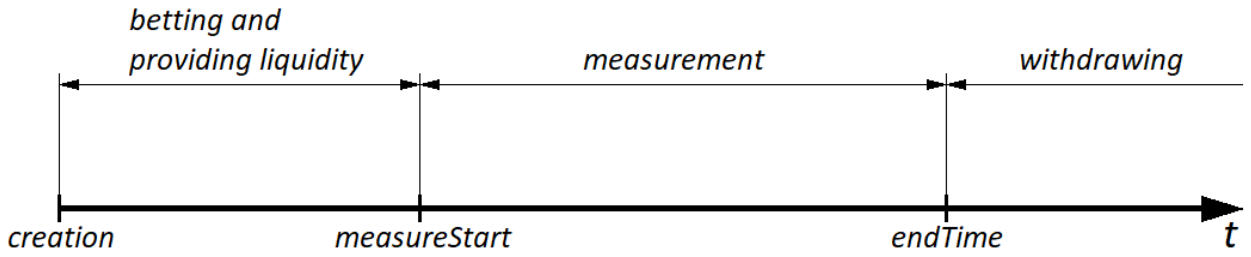
[2] An ordinary option is a contract between two parties giving the taker (buyer) the right, but not the obligation, to buy or sell a security at a predetermined price on or before a predetermined date. To acquire this right the taker pays a premium to the writer (seller) of the contract [1].

3. Event lifetime

Any network member can create a new event. In the creation transaction, they must provide all the variable values which describe the event (these variables will be specified below).

Successful events go through 4 stages:

- creation;
- liquidity adding and betting period which ends with $measureStart$ timepoint;
- measurement period which lasts for $measurePeriod$ and leads the event to the expiration at the $endTime$ timestamp;
- withdrawal period when the event winners and liquidity providers get their payouts.



4. Creating events

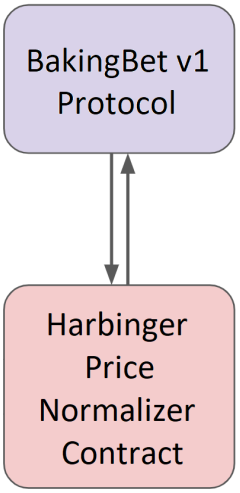
Every event has its' own parameters that are defined by the creator, these are:

- $currencyPair$ – exchange pairs from the list of oracle pairs;
- $targetDynamics$ – the value that determines the needed price move for S to occur. For example, $targetDynamics$ = *1100000* means that S will occur if the price of $currencyPair$ will rise by at least 10% during the $measurePeriod$;
- $betsCloseTime$ – time since new bets would not be accepted;
- $measurePeriod$ – number of seconds from $measureStartTime$ before anyone can call $close()$ to finish the event;

- $liquidityPercent$ – percent that is charged from winners' profits for liquidity use.

5. Contract interaction

Juster v1 platform has one main contract, that holds all events data and funds. To determine the outcome, the main contract interacts with the ***Harbinger Normalizer Oracle*** [3] contract.

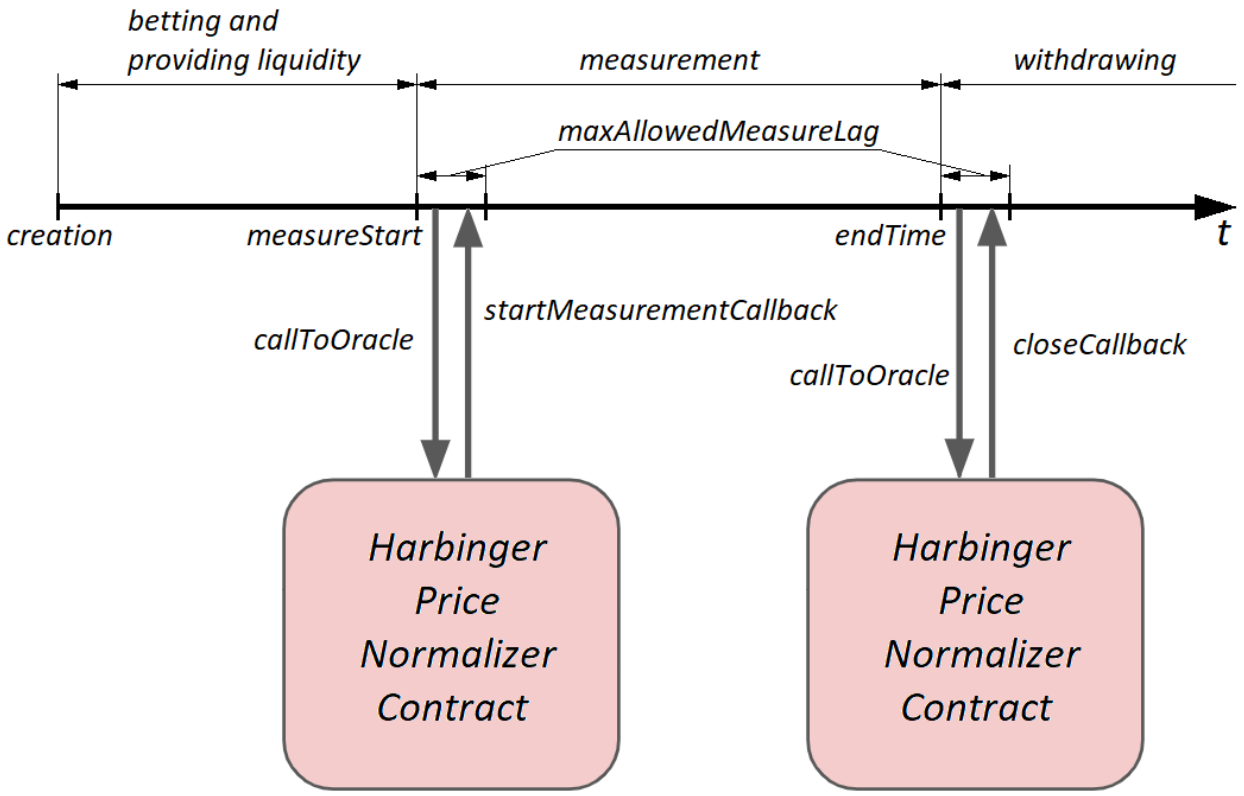


During the event lifetime, prices are retrieved twice during

$[betsCloseTime, betsCloseTime + maxAllowedMeasureLag)$ and

$[betsCloseTime + measurePeriod,$

$betsCloseTime + measurePeriod + maxAllowedMeasureLag)$ periods:



Prices retrieved from the oracle are used in the $close()$ function to define the outcome of the event.

6. Event parameters

In Juster v1, a bunch of service parameters are defined by the main contract manager and are common for all events. Here are the main ones:

- $measureStartFee$ and $expirationFee$ – fees that the event creator must pay to ensure that someone would make $startMeasurement()$ and $close()$ calls. Callers of these functions would get these fees as rewards for their work;
- $rewardCallFee$ – fees that are taken from winning participants in case they don't withdraw in time and someone else makes it for them;
- $oracleAddress$ – address of the oracle used to retrieve prices;
- $minMeasurePeriod$ and $maxMeasurePeriod$ – *minimum and maximum possible duration of the event*;
- $minLiquidityPercent$and $maxLiquidityPercent$ – minimum and maximum percent charged from winners' profits for liquidity use;

- $maxAllowedMeasureLag$ – maximum time window during which the $startMeasument()$ and $close()$ functions should be called to complete the event successfully.

7. Betting

Every created event has two outcomes S and $\overline{S}$, which any user can bet on. Betting on the S outcome adds $betFor$ tez to that outcome liquidity pool and blocks $winDelta$ tez in the opposite outcome liquidity pool, keeping the constant product[3]:

$$poolFor_0 \cdot poolAgainst_0 = poolFor_1 \cdot poolAgainst_1, \quad (1)$$

where:

$poolFor_0$ – liquidity pool size for S outcome before the user's bet;

$poolAgainst_0$ – liquidity pool size for $\overline{S}$outcome before the user's bet;

$poolFor_1$ – liquidity pool size for S outcome after the user's bet;

$poolAgainst_1$ – liquidity pool size for $\overline{S}$outcome after the user's bet.

So:

$$poolFor_1 = poolFor_0 + betFor, \quad (2)$$

$$poolAgainst_1 = \frac{poolFor_0 \cdot poolAgainst_0}{poolFor_0 + betFor}, \quad (3)$$

$$winDelta = poolAgainst_0 - poolAgainst_1 = betFor \cdot \frac{poolAgainst_0}{poolFor_0 + betFor}, \quad (4)$$

where:

$betFor$ – the amount of tez user pays to make a bet;

$winDelta$ – actual net user's winnings in case of Soutcome.

So, when the event is successfully closed, the user gets

[3] Note: fees are not taken into account to simplify the explanation, and are described in a separate paragraph below.

$$payout = betFor + winDelta = betFor\left(1 + \frac{poolAgainst_0}{poolFor_0 + betFor}\right) \quad (5)$$

tez in total if S occurs or 0 tez if $\bar{S}$outcome occurs otherwise.

8. Providing liquidity

Providing liquidity protocol is quite similar to the well-known AMM models [4]. Any user can provide liquidity to the event during the providing and betting stage. The first user that provides liquidity to an empty event gets liquidity shares equals to:

$$newShares = sharePrecision, \quad (6)$$

where $sharePrecision$ – is a predefined value (currently 100 000 000).

Provided liquidity is shared between S and $\bar{S}$ pools in the $rate$ proportion specified by the provider.

When a new liquidity provider adds some liquidity to an existing event with some liquidity inside, they get a number of liquidity shares calculated using the equation:

$$newShares = amount \cdot \frac{totalLiquidityShares}{max(poolFor, poolAgainst)}, \quad (7)$$

where:

$amount$ – the amount of new liquidity added in tez;

$totalLiquidityShares$ – total number of liquidity shares before the adding operation;

$poolFor$ – the amount of tez in the S pool before the adding operation;

$poolAgainst$ – the amount of tez in the $\bar{S}$ pool before the adding operation.

In this case, provided liquidity is shared between S and $\bar{S}$ pools in the current $rate$ proportion. It's important to note that as only one of two outcomes can occur, the provider only needs to put funds in the biggest pool at the moment of adding new liquidity. Thus, they deposit the maximum possible amount that they can lose. The liquidity to the smallest pool is provided virtually to clear the calculations.

After the event is closed, liquidity providers share all the liquidity from the losing pool in proportion to their shares in that pool plus take back the amount they

added to the winning pool minus liquidity (the $loan$) that was virtually provided to the smallest pool.

9. Withdrawing

As mentioned above, when the event is successfully closed with the S outcome, for example, a participant that made a winning bet (of index i) on S gets a payout that equals to:

$$payout_i = betFor_i + winDelta_i = betFor_i \left(1 + \frac{poolAgainst_{i-1}}{poolFor_{i-1} + betFor_i}\right) \tag{8}$$

while participants who bet on $\overline{S}$get 0.

In that case, a liquidity provider that has some $liquidityShares_j$ gets a payout calculated using the equations:

$$loan = min(providedLiquidityFor, providedLiquidityAgainst), \tag{9}$$

$$payoutAgainst = liquidityShares_j \frac{poolAgainst}{totalLiquidityShares} - loan, \tag{10}$$

$$payout_j = providedLiquidityFor + payoutAgainst, \tag{11}$$

where the $loan$ is the amount that was not actually provided to the smallest pool but is used to make calculations more intuitive.

10. Fees

Taking into account that in Juster v1 participants pay liquidity bonus to the providers, the formulas (2), (3) and (4) above take the form of:

$$winDelta = betFor \cdot \frac{poolAgainst_0}{poolFor_0 + betFor} (1 - fee(t)) \tag{12}$$

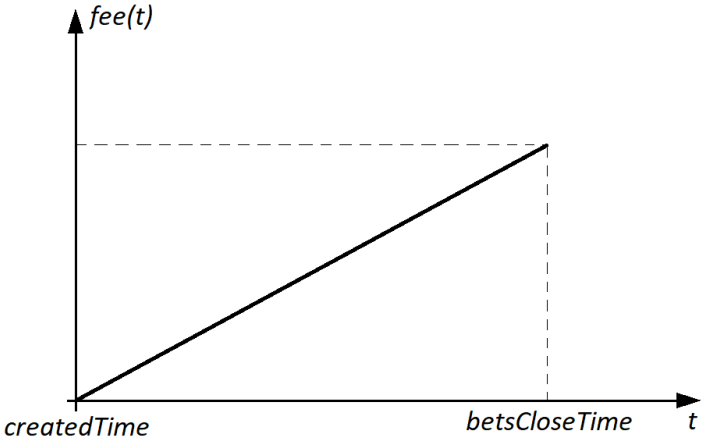
$$poolFor_1 = poolFor_0 + betFor \quad (13)$$

$$poolAgainst_1 = poolAgainst_0 - \frac{betFor \cdot poolAgainst_0}{poolFor_0 + betFor}(1 - fee(t)) \quad (14)$$

$fee(t)$ – is the fee for using liquidity that is increasing linearly during the *measurePeriod*:

$$fee(t) = liquidityPercent \cdot \frac{currentTime - createdTime}{betsCloseTime - createdTime} \quad (13)$$

As such, betting at the start of the event should be cheaper for participants, and $fee(t)$ is rising linearly in the period $[createdTime, betsCloseTime]$:



11. Quitting

For example, in some cases, when $startMeasurement()$ and $close()$ calls haven't been made in $maxAllowedMeasureLag$ period, the event is considered to fall into the Force Majeure state. Only withdrawal calls are possible in this state, and every participant and provider take their funds back.

Proposed JUSTER token economy and protocol improvements

1. New event types

In general, Juster Core protocol can be implemented for any market where events can be defined to have two possible outcomes S and $\overline{S}$. These can not only be financial markets but any sport, political events, etc. The only thing required is a reliable on-chain data feed on Tezos. Considering the explosive development of all kinds of oracle systems such as Chainlink [5], various new event types are supposed to be implemented in the near future.

To become more flexible and suitable for new types of events Juster contracts would be divided into two components:

- The first component would implement constant product pools with all betting, providing liquidity and withdrawal logic. It would rely entirely on the events from the second component;
- the second component would include all event lifecycle: creation, oracle interaction, and providing info about event results (either S or $\overline{S}$). A new component of this type can be created for new event logic.

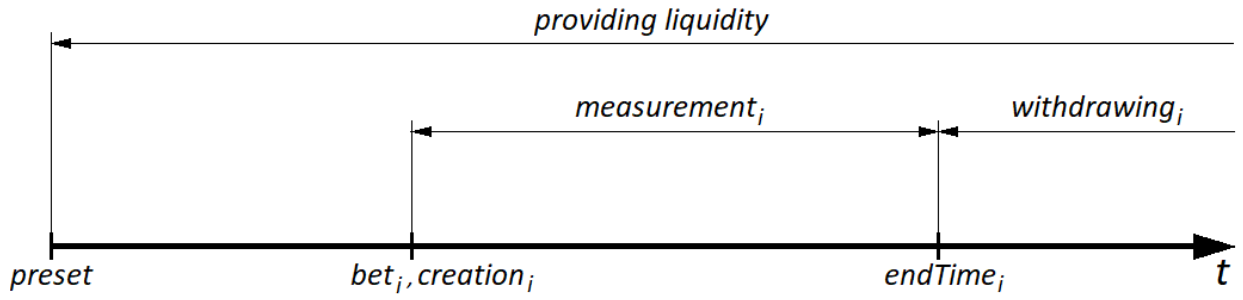
2. Binary options mode

Juster v1 protocol can be cheaply converted to ordinary binary options instruments. To do that, the liquidity provider and the participant have to compete against each other - the liquidity provider is trying to define the correct odds rate for the S and $\overline{S}$outcomes, actually making a kind of a bet on its correctness, while the participant is betting on the outcome, which is supposed to be underestimated by the provider.

The process looks pretty the same as in the case of the ordinary protocol implementation. But this time, it has the following 5 stages:

- preset, which is similar to creation, but has no $betsCloseTime$ parameter specified;
- liquidity adding, when the provider adds liquidity to the S and $\overline{S}$pools in some proportion, defining the rate;
- a bet, made by any user, creates a sub-event i and starts its measurement period;
- measurement period which lasts for $measurePeriod_i$ and leads the sub-event to the expiration at the $endTime_i$ timestamp;
- the withdrawal period of the sub-event when the participant and liquidity provider get their payouts.

It's important to note that every bet creates a new sub-event and changes the S and $\overline{S}$pools. Thus, every further bet is made with another rate. Also, new liquidity can be added at any time to the pools at the current rate. The whole process is illustrated below:



All other rules and calculations remain the same.

3. On-chain governance

For the first implementation, service variables would be controlled by the manager. That would allow the development team to check the service architecture's stability and respond to possible potential problems as quickly as possible. Once all the hypotheses are checked, it is supposed to let the community control all service variables via holding the JUSTER DAO token.

4. A community-managed treasury

To support the platform and distribute profits between JUSTER DAO token holders, liquidity provider profits are reduced by some fee percent that will be in control of the DAO. This fee is calculated when providers withdraw their liquidity after the event is closed and is aggregated on the smart contract. The DAO would be able to withdraw these rewards and vote to distribute these profits between token holders or use this fund to define and grant future development of new features and linked services and other purposes.

5. Liquidity providing market and shared liquidity pool

Staking rewards in Tezos is a crucial incentive that defines both additional restrictions and new opportunities for defi protocol development. In that context, new protocols that allow generating higher yield returns can become new hubs of network activity and create a new level of on- and off-chain community interaction.

Inspired by the core Tezos protocol evolutionary path, Juster protocol's roadmap suggests using some ideas and development scenarios of the Tezos network. One of these scenarios is the competitive public bakers market resulting from the need for bakers to independently distribute baking rewards, define fees, and contend to attract delegators' funds.

In Juster v1 protocol, anyone can create a new event, but being the first provider to propose the odds rate for the S and $\bar{S}$outcomes require some off-chain knowledge, which seems to be complicated enough to keep the number of initial providers relatively small. However, once the initial rate is defined, any user could easily add some funds to the liquidity pools in the same proportion just following some reliable initial providers' accounts.

Considering that any cheating initial provider can deliberately propose the wrong rate to make a bet on the underestimated outcome subsequently, this setup could lead to a new reputation market appear. That market could have a representation that is quite similar to the bakers rating services interface [7].

Also, this reputation could be tokenized in perspective, turning into tokens that allow using funds of a shared liquidity pool to create and fill events with liquidity. Profits from that activity should be distributed between event providers and users who made deposits to the pool (pool shareholders). From the point of

view of the pool shareholders, it would present another way to increase passive income.

This shared liquidity pool is supposed to be delegated to some public baker, and all shareholders would get baking rewards. When the size of the pool expands, it can be then divided into several pools to delegate funds more efficiently.

6. Tradable FA2 tokens for long-term events

For events with long $measurePeriod$ value (weeks and months), it is possible to create some liquid FA2 tokens representing potential payouts for bets made on S and $\overline{S}$ outcomes. After the event is closed, every winning token can be exchanged for 1 tez. Losing tokens can be burned for fun.

So, liquidity providers, participants and users could exchange and trade these tokens even after the $betsCloseTime$ passed. For that purpose, some kind of a simple DEX with limited functionality and truncated price curve can be built as an on-top protocol service.

7. Tezos tokens pools

Along with the Tezos tokens capitalization growth, it would become rational to use tokens to make bets and provide liquidity to the events' pools. First of all, it could be synthetic tokens like tzBTC and some stablecoins like kUSD. Also, using the JUSTER native token would provide the protocol with new potential use cases and features.

References

[1] https://www.asx.com.au/documents/resources/UnderstandingOptions.pdf

[2] Hayden Adams. 2018. url: https://hackmd.io/@HaydenAdams/HJ9jLsfTz

[3] https://github.com/tacoinfra/harbinger#readme

[4] https://blog.gnosis.pm/building-a-decentralized-exchange-in-ethereum-eea4e7452d6e

[5] https://medium.com/the-cryptonomic-aperiodical/bringing-secure-chainlink-oracles-to-tezos-8b7b917e5c18

[6] https://pintail.medium.com/uniswap-a-good-deal-for-liquidity-providers-104c0b6816f2

[7] https://baking-bad.org/docs/tezos-baker-metrics

Disclaimer