Gnosis Prediction Market Contracts Documentation

Gnosis

Jul 22, 2019

Contents:

Collection of smart contracts for the Gnosis prediction market platform (https://www.gnosis.pm). To interact with those contracts have a look at (https://github.com/gnosis/pm-js).

CHAPTER 1

Install

Install requirements with npm:

```
npm install @gnosis.pm/pm-contracts
```

CHAPTER 2

Testing and Linting

Run all tests (requires Node version >=7 for *async/await*, and will automatically run TestRPC in the background):

```
npm test
```

Run all tests matching a regexp pattern by setting the *TEST_GREP* environment variable:

```
TEST_GREP='short selling' npm test
```

Lint the JS:

```
npm run lint
```

CHAPTER 3

Compile and Deploy

These commands apply to the RPC provider running on port 8545. You may want to have TestRPC running in the background. They are really wrappers around the [corresponding Truffle commands](http://truffleframework.com/docs/advanced/commands).

Compile all contracts to obtain ABI and bytecode:

```
npm run compile
```

Migrate all contracts required for the basic framework onto network associated with RPC provider:

```
npm run migrate
```

CHAPTER 4

Network Artifacts

Show the deployed addresses of all contracts on all networks:

```
npm run networks
```

Command line options for *truffle* can be passed down through NPM by preceding the options list with –.

For example, to clean network artifacts:

```
npm run networks -- --clean
```

Network artifacts from running migrations will contain addresses of deployed contracts on the Kovan and Rinkeby testnets.

Take network info from *networks.json* and inject it into contract build artifacts. This is done prepublish as well:

```
npm run injectnetinfo
```

Extract all network information into *networks.json*:

Be aware that this will clobber *networks.json*, so be careful with this command:

```
npm run extractnetinfo
```

CHAPTER 5

Gas Measurements

Log gas measurements into *build/gas-stats.json*:

```
npm run measuregasstats
```

CHAPTER 6

Documentation

There is a copy version hosted online at https://gnosis.github.io/pm-contracts/

Build docs with doxity:

```
scripts/makedocs.sh
```

CHAPTER 7

Security and Liability

All contracts are WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

CHAPTER 8

License

All smart contracts are released under LGPL v.3.

CHAPTER 9

Contributors

- Stefan George (Georgi87)
- Martin Koeppelmann (koeppelmann)
- Alan Lu (cag)
- Roland Kofler (rolandkofler)
- Collin Chin (collinc97)
- Christopher Gewecke (cgewecke)

9.1 CampaignFactory

contract **CampaignFactory**

Title Campaign factory contract - Allows to create campaign contracts

Author Stefan George - <stefan@gnosis.pm>

event **CampaignCreation** (*address indexed creator*, *Campaign campaign*, *Event eventContract*, *StandardMarketFactory marketFactory*, *MarketMaker marketMaker*, *uint24 fee*, *uint funding*, *uint deadline*)

Campaign *public* **campaignMasterCopy**

constructor (*Campaign _campaignMasterCopy*) *public*

function **createCampaign** (*Event eventContract*, *StandardMarketFactory marketFactory*, *MarketMaker marketMaker*, *uint24 fee*, *uint funding*, *uint deadline*) *public*

Creates a new campaign contract

Parameters

- **eventContract** – Event contract

- **marketFactory** – Market factory contract
- **marketMaker** – Market maker contract
- **fee** – Market fee
- **funding** – Initial funding for market
- **deadline** – Campaign deadline

Return Market contract

9.2 Campaign

contract **Campaign** is Proxied, CampaignData

Title Campaign contract - Allows to crowdfund a market

Author Stefan George - <stefan@gnosis.pm>

function **fund** (*uint amount*) *public*
Allows to contribute to required market funding

Parameters

- **amount** – Amount of collateral tokens

function **refund** () *public*
Withdraws refund amount

Return Refund amount

function **createMarket** () *public*
Allows to create market after successful funding

Return Market address

function **closeMarket** () *public*
Allows to withdraw fees from market contract to campaign contract

Return Fee amount

function **withdrawFees** () *public*
Allows to withdraw fees from campaign contract to contributor

Return Fee amount

9.3 CategoricalEvent

contract **CategoricalEvent** is Proxied, Event

Title Categorical event contract - Categorical events resolve to an outcome from a set of outcomes

Author Stefan George - <stefan@gnosis.pm>

function **redeemWinnings** () *public*
Exchanges sender's winning outcome tokens for collateral tokens

Return Sender's winnings

function **getEventHash** () *public*
Calculates and returns event hash

Return Event hash

9.4 CentralizedOracleFactory

contract **CentralizedOracleFactory**

Title Centralized oracle factory contract - Allows to create centralized oracle contracts

Author Stefan George - <stefan@gnosis.pm>

event **CentralizedOracleCreation** (*address indexed creator*, *CentralizedOracle centralizedOracle*, *bytes ipfsHash*)

CentralizedOracle *public* **centralizedOracleMasterCopy**

constructor (*CentralizedOracle _centralizedOracleMasterCopy*) *public*

function **createCentralizedOracle** (*bytes memory ipfsHash*) *public*

Creates a new centralized oracle contract

Parameters

- **ipfsHash** – Hash identifying off chain event description

Return Oracle contract

9.5 CentralizedOracle

contract **CentralizedOracle** is Proxied, Oracle, CentralizedOracleData

Title Centralized oracle contract - Allows the contract owner to set an outcome

Author Stefan George - <stefan@gnosis.pm>

function **replaceOwner** (*address newOwner*) *public*

Replaces owner

Parameters

- **newOwner** – New owner

function **setOutcome** (*int _outcome*) *public*

Sets event outcome

Parameters

- **_outcome** – Event outcome

function **isOutcomeSet** () *public*

Returns if winning outcome is set

Return Is outcome set?

function **getOutcome** () *public*

Returns outcome

Return Outcome

9.6 DifficultyOracleFactory

contract **DifficultyOracleFactory**

Title Difficulty oracle factory contract - Allows to create difficulty oracle contracts

Author Stefan George - <stefan@gnosis.pm>

event **DifficultyOracleCreation** (*address indexed creator*, *DifficultyOracle difficultyOracle*, *uint blockNumber*)

function **createDifficultyOracle** (*uint blockNumber*) *public*

Creates a new difficulty oracle contract

Parameters

- **blockNumber** – Target block number

Return Oracle contract

9.7 DifficultyOracle

contract **DifficultyOracle** is Oracle

Title Difficulty oracle contract - Oracle to resolve difficulty events at given block

Author Stefan George - <stefan@gnosis.pm>

event **OutcomeAssignment** (*uint difficulty*)

uint *public* **blockNumber**

uint *public* **difficulty**

constructor (*uint _blockNumber*) *public*

Contract constructor validates and sets target block number

Parameters

- **_blockNumber** – Target block number

function **setOutcome** () *public*

Sets difficulty as winning outcome for specified block

function **isOutcomeSet** () *public*

Returns if difficulty is set

Return Is outcome set?

function **getOutcome** () *public*

Returns difficulty

Return Outcome

9.8 EventFactory

contract **EventFactory**

Title Event factory contract - Allows creation of categorical and scalar events

Author Stefan George - <stefan@gnosis.pm>

event **CategoricalEventCreation**(*address indexed creator*, *CategoricalEvent categoricalEvent*, *ERC20 collateralToken*, *Oracle oracle*, *uint8 outcomeCount*)

event **ScalarEventCreation**(*address indexed creator*, *ScalarEvent scalarEvent*, *ERC20 collateralToken*, *Oracle oracle*, *int lowerBound*, *int upperBound*)

mapping (bytes32 => CategoricalEvent) *public* **categoricalEvents**

mapping (bytes32 => ScalarEvent) *public* **scalarEvents**

CategoricalEvent *public* **categoricalEventMasterCopy**

ScalarEvent *public* **scalarEventMasterCopy**

OutcomeToken *public* **outcomeTokenMasterCopy**

constructor(*CategoricalEvent _categoricalEventMasterCopy*, *ScalarEvent _scalarEventMasterCopy*, *OutcomeToken _outcomeTokenMasterCopy*) *public*

function **createCategoricalEvent**(*ERC20 collateralToken*, *Oracle oracle*, *uint8 outcomeCount*) *public*

Creates a new categorical event and adds it to the event mapping

Parameters

- **collateralToken** – Tokens used as collateral in exchange for outcome tokens
- **oracle** – Oracle contract used to resolve the event
- **outcomeCount** – Number of event outcomes

Return Event contract

function **createScalarEvent**(*ERC20 collateralToken*, *Oracle oracle*, *int lowerBound*, *int upperBound*) *public*

Creates a new scalar event and adds it to the event mapping

Parameters

- **collateralToken** – Tokens used as collateral in exchange for outcome tokens
- **oracle** – Oracle contract used to resolve the event
- **lowerBound** – Lower bound for event outcome
- **upperBound** – Lower bound for event outcome

Return Event contract

9.9 Event

contract **Event** is EventData

Title Event contract - Provide basic functionality required by different event types

Author Stefan George - <stefan@gnosis.pm>

function **buyAllOutcomes**(*uint collateralTokenCount*) *public*

Buys equal number of tokens of all outcomes, exchanging collateral tokens and sets of outcome tokens 1:1

Parameters

- **collateralTokenCount** – Number of collateral tokens

function **sellAllOutcomes**(*uint outcomeTokenCount*) *public*
Sells equal number of tokens of all outcomes, exchanging collateral tokens and sets of outcome tokens 1:1

Parameters

- **outcomeTokenCount** – Number of outcome tokens

function **setOutcome**() *public*
Sets winning event outcome

function **getOutcomeCount**() *public*
Returns outcome count

Return Outcome count

function **getOutcomeTokens**() *public*
Returns outcome tokens array

Return Outcome tokens

function **getOutcomeTokenDistribution**(*address owner*) *public*
Returns the amount of outcome tokens held by owner

Return Outcome token distribution

function **getEventHash**() *public*
Calculates and returns event hash

Return Event hash

function **redeemWinnings**() *public*
Exchanges sender's winning outcome tokens for collateral tokens

Return Sender's winnings

9.10 FutarchyOracleFactory

contract **FutarchyOracleFactory**

Title Futarchy oracle factory contract - Allows to create Futarchy oracle contracts

Author Stefan George - <stefan@gnosis.pm>

event **FutarchyOracleCreation**(*address indexed creator*, *FutarchyOracle futarchyOracle*, *ERC20 collateralToken*, *Oracle oracle*, *uint8 outcomeCount*, *int lowerBound*, *int upperBound*, *MarketMaker marketMaker*, *uint24 fee*, *uint tradingPeriod*, *uint startDate*)

EventFactory **eventFactory**

StandardMarketWithPriceLoggerFactory **marketFactory**

FutarchyOracle *public* **futarchyOracleMasterCopy**

constructor(*FutarchyOracle _futarchyOracleMasterCopy*, *EventFactory _eventFactory*, *StandardMarketWithPriceLoggerFactory _marketFactory*) *public*
Constructor sets event factory contract

Parameters

- **_eventFactory** – Event factory contract
- **_marketFactory** – Market factory contract

function **createFutarchyOracle**(*ERC20 collateralToken*, *Oracle oracle*, *uint8 outcomeCount*, *intpublic lowerBound*, *int upperBound*, *MarketMaker marketMaker*, *uint24 fee*, *uint tradingPeriod*, *uint startDate*)

Creates a new Futarchy oracle contract

Parameters

- **collateralToken** – Tokens used as collateral in exchange for outcome tokens
- **oracle** – Oracle contract used to resolve the event
- **outcomeCount** – Number of event outcomes
- **lowerBound** – Lower bound for event outcome
- **upperBound** – Lower bound for event outcome
- **marketMaker** – Market maker contract
- **fee** – Market fee
- **tradingPeriod** – Trading period before decision can be determined
- **startDate** – Start date for price logging

Return Oracle contract

9.11 FutarchyOracle

contract **FutarchyOracle** is Proxied, Oracle, FutarchyOracleData

Title Futarchy oracle contract - Allows to create an oracle based on market behaviour

Author Stefan George - <stefan@gnosis.pm>

function **fund**(*uint funding*) *public*

Funds all markets with equal amount of funding

Parameters

- **funding** – Amount of funding

function **close**() *public*

Closes market for winning outcome and redeems winnings and sends all collateral tokens to creator

function **setOutcome**() *public*

Allows to set the oracle outcome based on the market with largest long position

function **isOutcomeSet**() *public*

Returns if winning outcome is set

Return Is outcome set?

function **getOutcome**() *public*

Returns winning outcome

Return Outcome

9.12 LMSRMarketMaker

contract **LMSRMarketMaker** is MarketMaker

Title LMSR market maker contract - Calculates share prices based on share distribution and initial funding

Author Alan Lu - <alan.lu@gnosis.pm>

uint constant **ONE**

int constant **EXP_LIMIT**

function **calcNetCost** (*Market market*, *int[] memory outcomeTokenAmounts*) *public*

Calculates the net cost for executing a given trade.

Parameters

- **market** – Market contract
- **outcomeTokenAmounts** – Amounts of outcome tokens to buy from the market. If an amount is negative, represents an amount to sell to the market.

Return Net cost of trade. If positive, represents amount of collateral which would be paid to the market for the trade. If negative, represents amount of collateral which would be received from the market for the trade.

function **calcCost** (*Market market*, *uint8 outcomeTokenIndex*, *uint outcomeTokenCount*) *public*

Returns cost to buy given number of outcome tokens

Parameters

- **market** – Market contract
- **outcomeTokenIndex** – Index of outcome to buy
- **outcomeTokenCount** – Number of outcome tokens to buy

Return Cost

function **calcProfit** (*Market market*, *uint8 outcomeTokenIndex*, *uint outcomeTokenCount*) *public*

Returns profit for selling given number of outcome tokens

Parameters

- **market** – Market contract
- **outcomeTokenIndex** – Index of outcome to sell
- **outcomeTokenCount** – Number of outcome tokens to sell

Return Profit

function **calcMarginalPrice** (*Market market*, *uint8 outcomeTokenIndex*) *public*

Returns marginal price of an outcome

Parameters

- **market** – Market contract
- **outcomeTokenIndex** – Index of outcome to determine marginal price of

Return Marginal price of an outcome as a fixed point number

function **calcCostLevel** (*int logN*, *int[] memory netOutcomeTokensSold*, *uint funding*,*private Fixed192x64Math.EstimationMode estimationMode*)

Calculates the result of the LMSR cost function which is used to derive prices from the market state

Parameters

- **logN** – Logarithm of the number of outcomes
- **netOutcomeTokensSold** – Net outcome tokens sold by market

- **funding** – Initial funding for market

Return Cost level

function **sumExpOffset**(*int logN*, *int[] memory netOutcomeTokensSold*, *uint funding*, *uint8 outcomeIn-private dex*, *Fixed192x64Math.EstimationMode estimationMode*)

Calculates sum(exp(q/b - offset) for q in quantities), where offset is set so that the sum fits in 248-256 bits

Parameters

- **logN** – Logarithm of the number of outcomes
- **netOutcomeTokensSold** – Net outcome tokens sold by market
- **funding** – Initial funding for market
- **outcomeIndex** – Index of exponential term to extract (for use by marginal price function)

Return A result structure composed of the sum, the offset used, and the summand associated with the supplied index

function **getNetOutcomeTokensSold**(*Market market*) *private*

Gets net outcome tokens sold by market. Since all sets of outcome tokens are backed by corresponding collateral tokens, the net quantity of a token sold by the market is the number of collateral tokens (which is the same as the number of outcome tokens the market created) subtracted by the quantity of that token held by the market.

Parameters

- **market** – Market contract

Return Net outcome tokens sold by market

9.13 MajorityOracleFactory

contract **MajorityOracleFactory**

Title Majority oracle factory contract - Allows to create majority oracle contracts

Author Stefan George - <stefan@gnosis.pm>

event **MajorityOracleCreation**(*address indexed creator*, *MajorityOracle majorityOracle*, *Oracle[] oracles*)

MajorityOracle *public* **majorityOracleMasterCopy**

constructor(*MajorityOracle _majorityOracleMasterCopy*) *public*

function **createMajorityOracle**(*Oracle[] memory oracles*) *public*

Creates a new majority oracle contract

Parameters

- **oracles** – List of oracles taking part in the majority vote

Return Oracle contract

9.14 MajorityOracle

contract **MajorityOracle** is Proxied, Oracle, MajorityOracleData

Title Majority oracle contract - Allows to resolve an event based on multiple oracles with majority vote

Author Stefan George - <stefan@gnosis.pm>

function **getStatusAndOutcome** ( ) *public*

Allows to registers oracles for a majority vote

Return Is outcome set?

Return Outcome

function **isOutcomeSet** ( ) *public*

Returns if winning outcome is set

Return Is outcome set?

function **getOutcome** ( ) *public*

Returns winning outcome

Return Outcome

9.15 MarketMaker

contract **MarketMaker**

Title Abstract market maker contract - Functions to be implemented by market maker contracts

function **calcCost** (*Market market*, *uint8 outcomeTokenIndex*, *uint outcomeTokenCount*) *public*

function **calcProfit** (*Market market*, *uint8 outcomeTokenIndex*, *uint outcomeTokenCount*) *public*

function **calcNetCost** (*Market market*, *int[] memory outcomeTokenAmounts*) *public*

function **calcMarginalPrice** (*Market market*, *uint8 outcomeTokenIndex*) *public*

9.16 Market

contract **Market** is MarketData

Title Abstract market contract - Functions to be implemented by market contracts

function **fund** (*uint _funding*) *public*

function **close** ( ) *public*

function **withdrawFees** ( ) *public*

function **buy** (*uint8 outcomeTokenIndex*, *uint outcomeTokenCount*, *uint maxCost*) *public*

function **sell** (*uint8 outcomeTokenIndex*, *uint outcomeTokenCount*, *uint minProfit*) *public*

function **shortSell** (*uint8 outcomeTokenIndex*, *uint outcomeTokenCount*, *uint minProfit*) *public*

function **trade** (*int[] memory outcomeTokenAmounts*, *int costLimit*) *public*

function **calcMarketFee** (*uint outcomeTokenCost*) *public*

9.17 Oracle

contract **Oracle**

Title Abstract oracle contract - Functions to be implemented by oracles

function **isOutcomeSet** () *public*

function **getOutcome** () *public*

9.18 OutcomeToken

contract **OutcomeToken** is Proxied, ERC20

Title Outcome token contract - Issuing and revoking outcome tokens

Author Stefan George - <stefan@gnosis.pm>

event **Issuance** (*address indexed owner*, *uint amount*)

event **Revocation** (*address indexed owner*, *uint amount*)

address *public* **eventContract**

modifier **isEventContract** ()

function **issue** (*address _for*, *uint outcomeTokenCount*) *public*

Events contract issues new tokens for address. Returns success

Parameters

- **_for** – Address of receiver
- **outcomeTokenCount** – Number of tokens to issue

function **revoke** (*address _for*, *uint outcomeTokenCount*) *public*

Events contract revokes tokens for address. Returns success

Parameters

- **_for** – Address of token holder

- **outcomeTokenCount** – Number of tokens to revoke

9.19 ScalarEvent

contract **ScalarEvent** is Proxied, Event, ScalarEventData

Title Scalar event contract - Scalar events resolve to a number within a range

Author Stefan George - <stefan@gnosis.pm>

function **redeemWinnings** ( ) *public*

Exchanges sender's winning outcome tokens for collateral tokens

Return Sender's winnings

function **getEventHash** ( ) *public*

Calculates and returns event hash

Return Event hash

9.20 SignedMessageOracleFactory

contract **SignedMessageOracleFactory**

Title Signed message oracle factory contract - Allows to create signed message oracle contracts

Author Stefan George - <stefan@gnosis.pm>

event **SignedMessageOracleCreation** (*address indexed creator*, *SignedMessageOracle signedMessageOracle*, *address oracle*)

SignedMessageOracle *public* **signedMessageOracleMasterCopy**

constructor (*SignedMessageOracle _signedMessageOracleMasterCopy*) *public*

function **createSignedMessageOracle** (*bytes32 descriptionHash*, *uint8 v*, *bytes32 r*, *bytes32 s*) *public*

Creates a new signed message oracle contract

Parameters

- **descriptionHash** – Hash identifying off chain event description
- **v** – Signature parameter
- **r** – Signature parameter
- **s** – Signature parameter

Return Oracle contract

9.21 SignedMessageOracle

contract **SignedMessageOracle** is Proxied, Oracle, SignedMessageOracleData

Title Signed message oracle contract - Allows to set an outcome with a signed message

Author Stefan George - <stefan@gnosis.pm>

function **replaceSigner** (*address newSigner*, *uint _nonce*, *uint8 v*, *bytes32 r*, *bytes32 s*) *public*
Replaces signer

Parameters

- **newSigner** – New signer
- **_nonce** – Unique nonce to prevent replay attacks
- **v** – Signature parameter
- **r** – Signature parameter
- **s** – Signature parameter

function **setOutcome** (*int _outcome*, *uint8 v*, *bytes32 r*, *bytes32 s*) *public*
Sets outcome based on signed message

Parameters

- **_outcome** – Signed event outcome
- **v** – Signature parameter
- **r** – Signature parameter
- **s** – Signature parameter

function **isOutcomeSet** () *public*
Returns if winning outcome

Return Is outcome set?

function **getOutcome** () *public*
Returns winning outcome

Return Outcome

9.22 StandardMarketFactory

contract **StandardMarketFactory**

Title Market factory contract - Allows to create market contracts

Author Stefan George - <stefan@gnosis.pm>

event **StandardMarketCreation** (*address indexed creator*, *Market market*, *Event eventContract*, *MarketMaker marketMaker*, *uint24 fee*)

StandardMarket *public* **standardMarketMasterCopy**

constructor (*StandardMarket _standardMarketMasterCopy*) *public*

function **createMarket** (*Event eventContract*, *MarketMaker marketMaker*, *uint24 fee*) *public*
Creates a new market contract

Parameters

- **eventContract** – Event contract
- **marketMaker** – Market maker contract
- **fee** – Market fee

Return Market contract

9.23 StandardMarket

contract **StandardMarket** is Proxied, Market, StandardMarketData

Title Standard market contract - Backed implementation of standard markets

Author Stefan George - <stefan@gnosis.pm>

modifier **isCreator**()

modifier **atStage**(*Stages _stage*)

function **fund**(*uint _funding*) *public*
Allows to fund the market with collateral tokens converting them into outcome tokens

Parameters

- **_funding** – Funding amount

function **close**() *public*
Allows market creator to close the markets by transferring all remaining outcome tokens to the creator

function **withdrawFees**() *public*
Allows market creator to withdraw fees generated by trades

Return Fee amount

function **buy**(*uint8 outcomeTokenIndex*, *uint outcomeTokenCount*, *uint maxCost*) *public*
Allows to buy outcome tokens from market maker

Parameters

- **outcomeTokenIndex** – Index of the outcome token to buy
- **outcomeTokenCount** – Amount of outcome tokens to buy
- **maxCost** – The maximum cost in collateral tokens to pay for outcome tokens

Return Cost in collateral tokens

function **sell**(*uint8 outcomeTokenIndex*, *uint outcomeTokenCount*, *uint minProfit*) *public*
Allows to sell outcome tokens to market maker

Parameters

- **outcomeTokenIndex** – Index of the outcome token to sell
- **outcomeTokenCount** – Amount of outcome tokens to sell
- **minProfit** – The minimum profit in collateral tokens to earn for outcome tokens

Return Profit in collateral tokens

function **shortSell**(*uint8 outcomeTokenIndex*, *uint outcomeTokenCount*, *uint minProfit*) *public*
Buys all outcomes, then sells all shares of selected outcome which were bought, keeping shares of all other outcome tokens.

Parameters

- **outcomeTokenIndex** – Index of the outcome token to short sell
- **outcomeTokenCount** – Amount of outcome tokens to short sell
- **minProfit** – The minimum profit in collateral tokens to earn for short sold outcome tokens

Return Cost to short sell outcome in collateral tokens

function **trade** (*int[] memory outcomeTokenAmounts*, *int collateralLimit*) *public*

Allows to trade outcome tokens and collateral with the market maker

Parameters

- **outcomeTokenAmounts** – Amounts of each outcome token to buy or sell. If positive, will buy this amount of outcome token from the market. If negative, will sell this amount back to the market instead.
- **collateralLimit** – If positive, this is the limit for the amount of collateral tokens which will be sent to the market to conduct the trade. If negative, this is the minimum amount of collateral tokens which will be received from the market for the trade. If zero, there is no limit.

Return If positive, the amount of collateral sent to the market. If negative, the amount of collateral received from the market. If zero, no collateral was sent or received.

function **tradeImpl** (*uint8 outcomeCount*, *int[] memory outcomeTokenAmounts*, *int collateralLimit*) *private*

function **calcMarketFee** (*uint outcomeTokenCost*) *public*

Calculates fee to be paid to market maker

Parameters

- **outcomeTokenCost** – Cost for buying outcome tokens

Return Fee for trade

9.24 StandardMarketWithPriceLoggerFactory

contract **StandardMarketWithPriceLoggerFactory**

Title Market factory contract - Allows to create market contracts

Author Stefan George - <stefan@gnosis.pm>

event **StandardMarketWithPriceLoggerCreation** (*address indexed creator*, *Market market*, *Event eventContract*, *MarketMaker marketMaker*, *uint24 fee*, *uint startDate*)

StandardMarketWithPriceLogger *public* **standardMarketWithPriceLoggerMasterCopy**

constructor (*StandardMarketWithPriceLogger _standardMarketWithPriceLoggerMasterCopy*) *public*

function **createMarket** (*Event eventContract*, *MarketMaker marketMaker*, *uint24 fee*, *uint startDate*) *public*

Creates a new market contract

Parameters

- **eventContract** – Event contract
- **marketMaker** – Market maker contract
- **fee** – Market fee
- **startDate** – Start date for price logging

Return Market contract

9.25 StandardMarketWithPriceLogger

contract **StandardMarketWithPriceLogger** is StandardMarket, StandardMarketWithPriceLoggerData

function **buy** (*uint8 outcomeTokenIndex*, *uint outcomeTokenCount*, *uint maxCost*) *public*
Allows to buy outcome tokens from market maker

Parameters

- **outcomeTokenIndex** – Index of the outcome token to buy
- **outcomeTokenCount** – Amount of outcome tokens to buy
- **maxCost** – The maximum cost in collateral tokens to pay for outcome tokens

Return Cost in collateral tokens

function **sell** (*uint8 outcomeTokenIndex*, *uint outcomeTokenCount*, *uint minProfit*) *public*
Allows to sell outcome tokens to market maker

Parameters

- **outcomeTokenIndex** – Index of the outcome token to sell
- **outcomeTokenCount** – Amount of outcome tokens to sell
- **minProfit** – The minimum profit in collateral tokens to earn for outcome tokens

Return Profit in collateral tokens

function **shortSell** (*uint8 outcomeTokenIndex*, *uint outcomeTokenCount*, *uint minProfit*) *public*
Buys all outcomes, then sells all shares of selected outcome which were bought, keeping shares of all other outcome tokens.

Parameters

- **outcomeTokenIndex** – Index of the outcome token to short sell
- **outcomeTokenCount** – Amount of outcome tokens to short sell
- **minProfit** – The minimum profit in collateral tokens to earn for short sold outcome tokens

Return Cost to short sell outcome in collateral tokens

function **trade** (*int[] memory outcomeTokenAmounts*, *int collateralLimit*) *public*
Allows to trade outcome tokens with market maker

Parameters

- **outcomeTokenAmounts** – Amounts of outcome tokens to trade
- **collateralLimit** – The maximum cost or minimum profit in collateral tokens

Return Cost/profit in collateral tokens

function **close** () *public*
Allows market creator to close the markets by transferring all remaining outcome tokens to the creator

function **getAvgPrice** () *public*
Calculates average price for long tokens based on price integral

Return Average price for long tokens over time

function **logPriceBefore** () *private*
Adds price integral since the last trade to the total price integral

function **logPriceAfter**() *private*
Updates last trade timestamp and price

9.26 UltimateOracleFactory

contract **UltimateOracleFactory**

Title Ultimate oracle factory contract - Allows to create ultimate oracle contracts

Author Stefan George - <stefan@gnosis.pm>

event **UltimateOracleCreation**(*address indexed creator*, *UltimateOracle ultimateOracle*, *Oracle oracle*, *ERC20 collateralToken*, *uint8 spreadMultiplier*, *uint challengePeriod*, *uint challengeAmount*, *uint frontRunnerPeriod*)

UltimateOracle *public* **ultimateOracleMasterCopy**

constructor(*UltimateOracle _ultimateOracleMasterCopy*) *public*

function **createUltimateOracle**(*Oracle oracle*, *ERC20 collateralToken*, *uint8 spreadMultiplier*,*public*
uint challengePeriod, *uint challengeAmount*, *uint frontRunnerPeriod*)
Creates a new ultimate Oracle contract

Parameters

- **oracle** – Oracle address
- **collateralToken** – Collateral token address
- **spreadMultiplier** – Defines the spread as a multiple of the money bet on other outcomes
- **challengePeriod** – Time to challenge oracle outcome
- **challengeAmount** – Amount to challenge the outcome
- **frontRunnerPeriod** – Time to overbid the front-runner

Return Oracle contract

9.27 UltimateOracle

contract **UltimateOracle** is Proxied, Oracle, UltimateOracleData

Title Ultimate oracle contract - Allows to swap oracle result for ultimate oracle result

Author Stefan George - <stefan@gnosis.pm>

function **setForwardedOutcome**() *public*
Allows to set oracle outcome

function **challengeOutcome**(*int _outcome*) *public*
Allows to challenge the oracle outcome

Parameters

- **_outcome** – Outcome to bid on

function **voteForOutcome**(*int _outcome*, *uint amount*) *public*
Allows to challenge the oracle outcome

Parameters

- **_outcome** – Outcome to bid on
- **amount** – Amount to bid

function **withdraw**() *public*
Withdraws winnings for user

Return Winnings

function **isChallengePeriodOver**() *public*
Checks if time to challenge the outcome is over

Return Is challenge period over?

function **isFrontRunnerPeriodOver**() *public*
Checks if time to overbid the front runner is over

Return Is front runner period over?

function **isChallenged**() *public*
Checks if outcome was challenged

Return Is challenged?

function **isOutcomeSet**() *public*
Returns if winning outcome is set

Return Is outcome set?

function **getOutcome**() *public*
Returns winning outcome

Return Outcome

CHAPTER 10

Indices and tables

- genindex

Index

F

L

M

O

S

U