

---

# **Gnosis Prediction Market Contracts Documentation**

**Gnosis**

**Jan 10, 2020**



---

## Contents:

---

<b>1</b>	<b>Install</b>	<b>3</b>
<b>2</b>	<b>Testing and Linting</b>	<b>5</b>
<b>3</b>	<b>Compile and Deploy</b>	<b>7</b>
<b>4</b>	<b>Network Artifacts</b>	<b>9</b>
<b>5</b>	<b>Gas Measurements</b>	<b>11</b>
<b>6</b>	<b>Documentation</b>	<b>13</b>
<b>7</b>	<b>Security and Liability</b>	<b>15</b>
<b>8</b>	<b>License</b>	<b>17</b>
<b>9</b>	<b>Contributors</b>	<b>19</b>
9.1	CampaignFactory . . . . .	19
9.2	Campaign . . . . .	20
9.3	CategoricalEvent . . . . .	20
9.4	CentralizedOracleFactory . . . . .	21
9.5	CentralizedOracle . . . . .	21
9.6	DifficultyOracleFactory . . . . .	22
9.7	DifficultyOracle . . . . .	22
9.8	EventFactory . . . . .	22
9.9	Event . . . . .	23
9.10	FutarchyOracleFactory . . . . .	24
9.11	FutarchyOracle . . . . .	25
9.12	LMSRMarketMaker . . . . .	25
9.13	MajorityOracleFactory . . . . .	27
9.14	MajorityOracle . . . . .	28
9.15	MarketMaker . . . . .	28
9.16	Market . . . . .	28
9.17	Oracle . . . . .	29
9.18	OutcomeToken . . . . .	29
9.19	ScalarEvent . . . . .	30
9.20	SignedMessageOracleFactory . . . . .	30
9.21	SignedMessageOracle . . . . .	30

9.22	StandardMarketFactory . . . . .	31
9.23	StandardMarket . . . . .	32
9.24	StandardMarketWithPriceLoggerFactory . . . . .	33
9.25	StandardMarketWithPriceLogger . . . . .	34
9.26	UltimateOracleFactory . . . . .	35
9.27	UltimateOracle . . . . .	35
<b>10</b>	<b>Indices and tables</b>	<b>37</b>
	<b>Index</b>	<b>39</b>

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  $\geq 7$  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.



- 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](mailto: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, Market-public Maker marketMaker, uint24 fee, uint funding, uint deadline*)

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](mailto: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](mailto:stefan@gnosis.pm)>

*function* **redeemWinning** () *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](mailto: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](mailto: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, Standard-public MarketWithPriceLoggerFactory \_marketFactory*)  
Constructor sets event factory contract

**Parameters**

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

*function* **createFutarchyOracle** (*ERC20 collateralToken, Oracle oracle, uint8 outcomeCount, int public 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](mailto: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](mailto: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  $\sum(\exp(q/b - \text{offset})$  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](mailto: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](mailto: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](mailto: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** ()

Updates last trade timestamp and price

*private*

## 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** ()

Allows to set oracle outcome

*public*

*function* **challengeOutcome** (*int \_outcome*)

Allows to challenge the oracle outcome

*public*

### Parameters

- **\_outcome** – Outcome to bid on

<i>function</i> <b>voteForOutcome</b> ( <i>int</i> _outcome, <i>uint</i> amount)	<i>public</i>
Allows to challenge the oracle outcome	
<b>Parameters</b>	
• <b>_outcome</b> – Outcome to bid on	
• <b>amount</b> – Amount to bid	
<i>function</i> <b>withdraw</b> ()	<i>public</i>
Withdraws winnings for user	
<b>Return</b> Winnings	
<i>function</i> <b>isChallengePeriodOver</b> ()	<i>public</i>
Checks if time to challenge the outcome is over	
<b>Return</b> Is challenge period over?	
<i>function</i> <b>isFrontRunnerPeriodOver</b> ()	<i>public</i>
Checks if time to overbid the front runner is over	
<b>Return</b> Is front runner period over?	
<i>function</i> <b>isChallenged</b> ()	<i>public</i>
Checks if outcome was challenged	
<b>Return</b> Is challenged?	
<i>function</i> <b>isOutcomeSet</b> ()	<i>public</i>
Returns if winning outcome is set	
<b>Return</b> Is outcome set?	
<i>function</i> <b>getOutcome</b> ()	<i>public</i>
Returns winning outcome	
<b>Return</b> Outcome	



## CHAPTER 10

---

### Indices and tables

---

- `genindex`



## C

Campaign (*contract*), 20  
 Campaign.closeMarket (*function*), 20  
 Campaign.createMarket (*function*), 20  
 Campaign.fund (*function*), 20  
 Campaign.refund (*function*), 20  
 Campaign.withdrawFees (*function*), 20  
 CampaignFactory (*contract*), 19  
 CampaignFactory.CampaignCreation (*event*), 19  
 CampaignFactory.campaignMasterCopy (*statevar*), 19  
 CampaignFactory.constructor (*constructor*), 19  
 CampaignFactory.createCampaign (*function*), 19  
 CategoricalEvent (*contract*), 20  
 CategoricalEvent.getEventHash (*function*), 20  
 CategoricalEvent.redeemWinnings (*function*), 20  
 CentralizedOracle (*contract*), 21  
 CentralizedOracle.getOutcome (*function*), 21  
 CentralizedOracle.isOutcomeSet (*function*), 21  
 CentralizedOracle.replaceOwner (*function*), 21  
 CentralizedOracle.setOutcome (*function*), 21  
 CentralizedOracleFactory (*contract*), 21  
 CentralizedOracleFactory.CentralizedOracleCreation (*event*), 21  
 CentralizedOracleFactory.centralizedOracleMasterCopy (*statevar*), 21  
 CentralizedOracleFactory.constructor (*constructor*), 21  
 CentralizedOracleFactory.createCentralizedOracle (*function*), 21

## D

DifficultyOracle (*contract*), 22

DifficultyOracle.blockNumber (*statevar*), 22  
 DifficultyOracle.constructor (*constructor*), 22  
 DifficultyOracle.difficulty (*statevar*), 22  
 DifficultyOracle.getOutcome (*function*), 22  
 DifficultyOracle.isOutcomeSet (*function*), 22  
 DifficultyOracle.OutcomeAssignment (*event*), 22  
 DifficultyOracle.setOutcome (*function*), 22  
 DifficultyOracleFactory (*contract*), 22  
 DifficultyOracleFactory.createDifficultyOracle (*function*), 22  
 DifficultyOracleFactory.DifficultyOracleCreation (*event*), 22

## E

Event (*contract*), 23  
 Event.buyAllOutcomes (*function*), 23  
 Event.getEventHash (*function*), 24  
 Event.getOutcomeCount (*function*), 24  
 Event.getOutcomeTokenDistribution (*function*), 24  
 Event.getOutcomeTokens (*function*), 24  
 Event.redeemWinnings (*function*), 24  
 Event.sellAllOutcomes (*function*), 23  
 Event.setOutcome (*function*), 24  
 EventFactory (*contract*), 22  
 EventFactory.CategoricalEventCreation (*event*), 23  
 EventFactory.categoricalEventMasterCopy (*statevar*), 23  
 EventFactory.categoricalEvents (*statevar*), 23  
 EventFactory.constructor (*constructor*), 23  
 EventFactory.createCategoricalEvent (*function*), 23  
 EventFactory.createScalarEvent (*function*), 23  
 EventFactory.outcomeTokenMasterCopy (*statevar*), 23

EventFactory.ScalarEventCreation (*event*),  
23  
EventFactory.scalarEventMasterCopy (*statevar*), 23  
EventFactory.scalarEvents (*statevar*), 23

## F

FutarchyOracle (*contract*), 25  
FutarchyOracle.close (*function*), 25  
FutarchyOracle.fund (*function*), 25  
FutarchyOracle.getOutcome (*function*), 25  
FutarchyOracle.isOutcomeSet (*function*), 25  
FutarchyOracle.setOutcome (*function*), 25  
FutarchyOracleFactory (*contract*), 24  
FutarchyOracleFactory.constructor (*constructor*), 24  
FutarchyOracleFactory.createFutarchyOracle (*function*), 24  
FutarchyOracleFactory.eventFactory (*statevar*), 24  
FutarchyOracleFactory.FutarchyOracleCreation (*event*), 24  
FutarchyOracleFactory.futarchyOracleMasterCopy (*statevar*), 24  
FutarchyOracleFactory.marketFactory (*statevar*), 24

## L

LMSRMarketMaker (*contract*), 25  
LMSRMarketMaker.calcCost (*function*), 26  
LMSRMarketMaker.calcCostLevel (*function*), 26  
LMSRMarketMaker.calcMarginalPrice (*function*), 26  
LMSRMarketMaker.calcNetCost (*function*), 26  
LMSRMarketMaker.calcProfit (*function*), 26  
LMSRMarketMaker.EXP\_LIMIT (*statevar*), 26  
LMSRMarketMaker.getNetOutcomeTokensSold (*function*), 27  
LMSRMarketMaker.ONE (*statevar*), 26  
LMSRMarketMaker.sumExpOffset (*function*), 27

## M

MajorityOracle (*contract*), 28  
MajorityOracle.getOutcome (*function*), 28  
MajorityOracle.getStatusAndOutcome (*function*), 28  
MajorityOracle.isOutcomeSet (*function*), 28  
MajorityOracleFactory (*contract*), 27  
MajorityOracleFactory.constructor (*constructor*), 27  
MajorityOracleFactory.createMajorityOracle (*function*), 27  
MajorityOracleFactory.MajorityOracleCreation (*event*), 27

MajorityOracleFactory.majorityOracleMasterCopy (*statevar*), 27  
Market (*contract*), 28  
Market.buy (*function*), 28  
Market.calcMarketFee (*function*), 29  
Market.close (*function*), 28  
Market.fund (*function*), 28  
Market.sell (*function*), 29  
Market.shortSell (*function*), 29  
Market.trade (*function*), 29  
Market.withdrawFees (*function*), 28  
MarketMaker (*contract*), 28  
MarketMaker.calcCost (*function*), 28  
MarketMaker.calcMarginalPrice (*function*), 28  
MarketMaker.calcNetCost (*function*), 28  
MarketMaker.calcProfit (*function*), 28

## O

Oracle (*contract*), 29  
Oracle.getOutcome (*function*), 29  
Oracle.isOutcomeSet (*function*), 29  
OutcomeToken (*contract*), 29  
OutcomeToken.eventContract (*statevar*), 29  
OutcomeToken.isEventContract (*modifier*), 29  
OutcomeToken.Issuance (*event*), 29  
OutcomeToken.issue (*function*), 29  
OutcomeToken.Revocation (*event*), 29  
OutcomeToken.revoke (*function*), 29

## S

ScalarEvent (*contract*), 30  
ScalarEvent.getEventHash (*function*), 30  
ScalarEvent.redeemWinnings (*function*), 30  
SignedMessageOracle (*contract*), 30  
SignedMessageOracle.getOutcome (*function*), 31  
SignedMessageOracle.isOutcomeSet (*function*), 31  
SignedMessageOracle.replaceSigner (*function*), 30  
SignedMessageOracle.setOutcome (*function*), 31  
SignedMessageOracleFactory (*contract*), 30  
SignedMessageOracleFactory.constructor (*constructor*), 30  
SignedMessageOracleFactory.createSignedMessageOracle (*function*), 30  
SignedMessageOracleFactory.SignedMessageOracleCreation (*event*), 30  
SignedMessageOracleFactory.signedMessageOracleMasterCopy (*statevar*), 30  
StandardMarket (*contract*), 32  
StandardMarket.atStage (*modifier*), 32  
StandardMarket.buy (*function*), 32

StandardMarket.calcMarketFee (function), 33  
 StandardMarket.close (function), 32  
 StandardMarket.fund (function), 32  
 StandardMarket.isCreator (modifier), 32  
 StandardMarket.sell (function), 32  
 StandardMarket.shortSell (function), 32  
 StandardMarket.trade (function), 33  
 StandardMarket.tradeImpl (function), 33  
 StandardMarket.withdrawFees (function), 32  
 StandardMarketFactory (contract), 31  
 StandardMarketFactory.constructor (constructor), 31  
 StandardMarketFactory.createMarket (function), 31  
 StandardMarketFactory.StandardMarketCreation (event), 31  
 StandardMarketFactory.standardMarketMasterCopy (statevar), 31  
 StandardMarketWithPriceLogger (contract), 34  
 StandardMarketWithPriceLogger.buy (function), 34  
 StandardMarketWithPriceLogger.close (function), 34  
 StandardMarketWithPriceLogger.getAvgPrice (function), 34  
 StandardMarketWithPriceLogger.logPriceAfter (function), 34  
 StandardMarketWithPriceLogger.logPriceBefore (function), 34  
 StandardMarketWithPriceLogger.sell (function), 34  
 StandardMarketWithPriceLogger.shortSell (function), 34  
 StandardMarketWithPriceLogger.trade (function), 34  
 StandardMarketWithPriceLoggerFactory (contract), 33  
 StandardMarketWithPriceLoggerFactory.constructor (constructor), 33  
 StandardMarketWithPriceLoggerFactory.createMarket (function), 33  
 StandardMarketWithPriceLoggerFactory.StandardMarketWithPriceLoggerCreation (event), 33  
 StandardMarketWithPriceLoggerFactory.standardMarketWithPriceLoggerMasterCopy (statevar), 33  
 UltimateOracle.isChallengePeriodOver (function), 36  
 UltimateOracle.isFrontRunnerPeriodOver (function), 36  
 UltimateOracle.isOutcomeSet (function), 36  
 UltimateOracle.setForwardedOutcome (function), 35  
 UltimateOracle.voteForOutcome (function), 35  
 UltimateOracle.withdraw (function), 36  
 UltimateOracleFactory (contract), 35  
 UltimateOracleFactory.constructor (constructor), 35  
 UltimateOracleFactory.createUltimateOracle (function), 35  
 UltimateOracleFactory.UltimateOracleCreation (event), 35  
 UltimateOracleFactory.ultimateOracleMasterCopy (statevar), 35

## U

UltimateOracle (contract), 35  
 UltimateOracle.challengeOutcome (function), 35  
 UltimateOracle.getOutcome (function), 36  
 UltimateOracle.isChallenged (function), 36