

dydx Process Quality Review

Score 92%

This is a [dydx](#) Process Quality Audit started on 20 May 2020 and completed on 18 June 2020. This was one of the three written while developing the process. That is why it took a month. It was performed using the Process Audit process (version 0.2) then was updated to V0.4 on 27 July 2020 and then 0.6 in 29 December 2020. The process is documented [here](#). The audit was performed by ShinkaRex of [Caliburn Consulting](#). Check out our [Telegram](#).

The final score of the audit is 92%, a great score. The breakdown of the scoring is in [Scoring Appendix](#).

Summary of the Process

Very simply, the review looks for the following declarations from the developer's site. With these declarations, it is reasonable to trust the smart contracts.

- **Here are my smart contracts on the blockchain**
- **Here is the documentation that explains what my smart contracts do**
- **Here are the tests I ran to verify my smart contract**
- **Here are the audit(s) performed on my code by third party experts**

Disclaimer

This report is for informational purposes only and does not constitute investment advice of any kind, nor does it constitute an offer to provide investment advisory or other services. Nothing in this report shall be considered a solicitation or offer to buy or sell any security, future, option or other financial instrument or to offer or provide any investment advice or service to any person in any jurisdiction. Nothing contained in this report constitutes investment advice or offers any opinion with respect to the suitability of any security, and the views expressed in this report should not be taken as advice to buy, sell or hold any security. The information in this report should not be relied upon for the purpose of investing. In preparing the information contained in this report, we have not taken into account the investment needs, objectives and financial circumstances of any particular investor. This information has no regard to the specific

investment objectives, financial situation and particular needs of any specific recipient of this information and investments discussed may not be suitable for all investors.

Any views expressed in this report by us were prepared based upon the information available to us at the time such views were written. Changed or additional information could cause such views to change. All information is subject to possible correction. Information may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.


This completed report is copyright (c) DeFiSafety 2021. Permission is given to copy in whole, retaining this copyright label.

Code and Team

This section looks at the code deployed on the Mainnet that gets audited and its corresponding software repository. The document explaining these questions is [here](#). This audit will answer the questions;

1. Is the deployed code address(s) readily available? (Y/N)
2. Is the code actively being used? (%)
3. Are the Contract(s) Verified/Verifiable? (Y/N)a
4. Does the code match a tagged version in the code hosting platform? (%)
5. Is the software repository healthy? (%)

Are the executing code addresses readily available? (Y/N)

 Answer: Yes

They are available at Address <https://docs.dydx.exchange/#solo-contract-addresses> as indicated in the [Appendix: Deployed Code](#). This Audit only covers the SoloMargin contract (address [0x1E0447b19BB6EcFdAe1e4AE1694b0C3659614e4e](#) created Apr 16, 2019 at 12:24 which is proxy'd to [0x56E7d4520ABFECf10b38368b00723d9BD3c21ee1](#) created on Apr-16-2019 12:23:36 AM +UTC (in other words, almost immediately after the first).

Is the code actively being used? (%)

 Answer: 100%

Activity is well in excess of 10 transactions a day, as indicated in the [Appendix](#).

Percentage Score Guidance

100%	More than 10 transactions a day
70%	More than 10 transactions a week
40%	More than 10 transactions a month
10%	Less than 10 transactions a month
0%	No activity

Is there a public software repository? (Y/N)

 Answer: Yes

GitHub: <https://github.com/dydxprotocol>

Is there a public software repository with the code at a minimum, but normally test and scripts also (Y/N). Even if the repo was created just to hold the files and has just 1 transaction, it gets a Yes. For teams with private repos, this answer is No.

Is there a development history visible? (%)

 Answer: 100%

With 498 commits this is a healthy repo.

This checks if the software repository demonstrates a strong steady history. This is normally demonstrated by commits, branches and releases in a software repository. A healthy history demonstrates a history of more than a month (at a minimum).

Guidance:

100% Any one of 100+ commits, 10+branches

70% Any one of 70+ commits, 7+branches

50% Any one of 50+ commits, 5+branches

30% Any one of 30+ commits, 3+branches

0% Less than 2 branches or less than 10 commits

Is the team public (not anonymous)? (Y/N)

 Answer: Yes

Teams members on the Company webpage.

Location: <https://dydx.exchange/company/>

For a yes in this question the real names of some team members must be public on the website or other documentation. If the team is anonymous and then this question is a No.

Documentation

This section looks at the software documentation. The document explaining these questions is [here](#).

Required questions are;

1. Is there a whitepaper? (Y/N)
2. Are the basic application requirements documented? (Y/N)
3. Do the requirements fully (100%) cover the deployed contracts? (%)
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)
5. Is it possible to trace software requirements to the implementation in code (%)

Is there a whitepaper? (Y/N)

 Answer: Yes

Location: <https://whitepaper.dydx.exchange/>

Are the basic software functions documented? (Y/N)

 Answer: Yes

Location: <https://docs.dydx.exchange/#/protocol>

How to improve this score

Does the software function documentation fully (100%) cover the deployed contracts? (%)


 Answer: 100%

While the basic functions of the code are explained on the website and GitHub, there is no association between these explanations and the code. So it is difficult to determine all the relevant code has requirements.

How to improve this score

This score can improve by adding content to the requirements document such that it comprehensively covers the requirements. For guidance, refer to the [SecurEth System Description Document](#) . Using tools that aid traceability detection will help.

Are there sufficiently detailed comments for all functions within the deployed contract code (%)

 Answer: 43%

Most structures (for instance in Actions.sol) have definitions. But most function definitions have virtually no commenting. The overall level of commenting is quite low and subsequent code maintenance could be challenging. Code examples are in the [Appendix: Example Code](#). As per the [Appendix: Software Lines of Code](#), there is 23% commenting to code.

The Comments to Code (CtC) ratio is the primary metric for this score.

Guidance:

- 100% CtC > 100 Useful comments consistently on all code
- 90-70% CtC > 70 Useful comment on most code
- 60-20% CtC > 20 Some useful commenting
- 0% CtC < 20 No useful commenting

How to improve this score

This score can improve by adding comments to the deployed code such that it comprehensively covers the code. For guidance, refer to the [SecurEth Software Requirements](#).

Is it possible to trace software requirements to the implementation in code (%)

 Answer: 90%

Location: <https://docs.dydx.exchange/#solo-operations>

The solo documentation shows clear traceability by including code snippets with the docs.

Guidance:

- 100% - Clear explicit traceability between code and documentation at a requirement level for all code
- 60% - Clear association between code and documents via non explicit traceability
- 40% - Documentation lists all the functions and describes their functions
- 0% - No connection between documentation and code

How to improve this score

This score can improve by adding content to the requirements document such that it comprehensively covers the requirements. For guidance, refer to the [SecurEth System](#)

[Description Document](#) . Using tools that aid traceability detection will help.

Testing

This section looks at the software testing available. It is explained in this [document](#). This section answers the following questions;

1. Full test suite (Covers all the deployed code) (%)
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)
3. Scripts and instructions to run the tests (Y/N)
4. Packaged with the deployed code (Y/N)
5. Report of the results (%)
6. Formal Verification test done (%)
7. Stress Testing environment (%)

Is there a Full test suite? (%)

 Answer: 100%

There are a significant number and lines of tests. There are contract tests (over 28 source files), action tests and others. Without actually running the tests it is difficult to confirm it is a complete test suite, but it certainly appears so. As per the software lines of code [Appendix: Software Lines of Code](#), there is a 221% test to code ratio.

Code coverage (Covers all the deployed lines of code, or explains misses) (%)

 Answer: 100%

They declare 100% code coverage and the report is available on their GitHub


Location: <https://docs.dydx.exchange/#code-coverage>

There are clear artifacts of unit tests (in **/tests** and **/src**) and scripts for coverage testing. We did not find the output of the coverage tests. At this point it seems to indicate full coverage. However without evidence, we cannot give a score higher than 70%.

How to improve this score

This score can improve by adding tests achieving full code coverage. A clear report and scripts in the software repository will guarantee a high score.

Scripts and instructions to run the tests (Y/N)

 Answer: Yes


In the scripts and readme subdirectory there are scripts to test, coverage, lint and verify.

Packaged with the deployed code (Y/N)

 Answer: Yes

The deployed code was saved as a GitHub release. The tests and scripts were packaged with the release in the repository zip file.

Report of the results (%)

 Answer: 70%

GitHub coveralls report clearly visible.

Guidance:

100% - Detailed test report as described below

70% - GitHub Code coverage report visible

0% - No test report evident

How to improve this score


Add a report with the results. The test scripts should generate the report or elements of it.

Formal Verification test done (%)

 Answer: 0%

No evidence of Formal Validation was found. This is still a rare type of test.

Stress Testing environment (%)

 Answer: 0%

No evidence of an active test network was found for the existing deployed protocol.

Audits

 Answer: 100%

dydx had multiple audits through their development as documented on their site. The OpenZeppelin audit included improvements that were resolved as indicated.

They have one audit from a top level audit organization. The audits is public and they have implemented findings in order to improve their code.

1. Multiple Audits performed before deployment and results public and implemented or not required (100%)
2. Single audit performed before deployment and results public and implemented or not required (90%)

3. Audit(s) performed after deployment and no changes required. Audit report is public. (70%)
4. No audit performed (20%)
5. Audit Performed after deployment, existence is public, report is not public and no improvements deployed (0%)

Appendices

Author Details

The author of this audit is Rex of [Caliburn Consulting](#).

Email : rex@caliburnc.com Twitter : @ShinkaRex

I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

I was coaxed to go to EthDenver 2017 and there I started [SecuEth.org](#) with Bryant and Roman. We created guidelines on good processes for blockchain code development. We got [EthFoundation funding](#) to assist in their development.

Process Quality Audits are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

Career wise I am a business development for an avionics supplier.

Scoring Appendix

PQ Audit Scoring Matrix (v0.6)	Total	dydx	
	Points	Answer	Points
Total	240		220.8
Code and Team			92%
1. Are the executing code addresses readily available? (Y/N)	30	Y	30
2. Is the code actively being used? (%)	10	100%	10
3. Is there a public software repository? (Y/N)	5	Y	5
4. Is there a development history visible? (%)	5	100%	5
Is the team public (not anonymous)? (Y/N)	20	Y	20
Code Documentation			
1. Is there a whitepaper? (Y/N)	5	Y	5
2. Are the basic software functions documented? (Y/N)	10	Y	10
3. Does the software function documentation fully (100%) cover the deployed contracts? (%)	15	100%	15
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	10	43%	4.3
5. Is it possible to trace from software documentation to the implementation in code (%)	5	90%	4.5
Testing			
1. Full test suite (Covers all the deployed code) (%)	20	100%	20
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)	5	100%	5
3. Scripts and instructions to run the tests? (Y/N)	5	Y	5
4. Packaged with the deployed code (Y/N)	5	Y	5
5. Report of the results (%)	10	70%	7
6. Formal Verification test done (%)	5	0%	0
7. Stress Testing environment (%)	5	0%	0
Audits			
Audit done	70	100%	70
Section Scoring			
Code and Team	70	100%	
Documentation	45	86%	
Testing	55	76%	
Audits	70	100%	

Deployed Code Appendix

The screenshot shows a web browser window with the URL `https://docs.dydx.exchange/#interest`. The page features a dark sidebar on the left with the **dydx** logo and a navigation menu. The main content area is light blue and displays a table of fields for the 'Interest' section.

dydx

Search

Introduction

Clients

Account API

Trading API

Markets API

WebSocket API

Solo Protocol

- Accounts
- Solo Markets
- Solo Trading Amounts
- Interest**
- Wej & Par
- Index
- Actions
- Operations
- Solo Amounts
- Solo Logs
- Solo Types
- Solo Web3
- Solo BigNumber
- Solo Standard Actions
- Solo Operations
- Get Standard Actions
- Solo Contract Addresses

Perpetual Protocol

Perpetual Contract Guide

Security

<code>deltaMargin</code>		The change in settlement token (e.g. USDC).
<code>newMargin</code>		The new balance of settlement token (e.g. USDC).
<code>deltaPosition</code>		The change in position token (e.g. PBTC).
<code>newPosition</code>		The amount in position token (e.g. PBTC).
<code>indexValue</code>		The new index value of the account.
<code>indexTimestamp</code>		The timestamp for when the index value was set.
<code>orderNumber</code>		Number used for ordering the balance updates.
<code>isPendingBlock</code>		Whether the specific balance update is pending or not

Unsubscribing

Field Name	JSON type	Description
<code>type</code>	string	Must be set to "unsubscribe"
<code>channel</code>	string	The channel to unsubscribe from
<code>id</code>	string	A market to unsubscribe from on the channel

Response

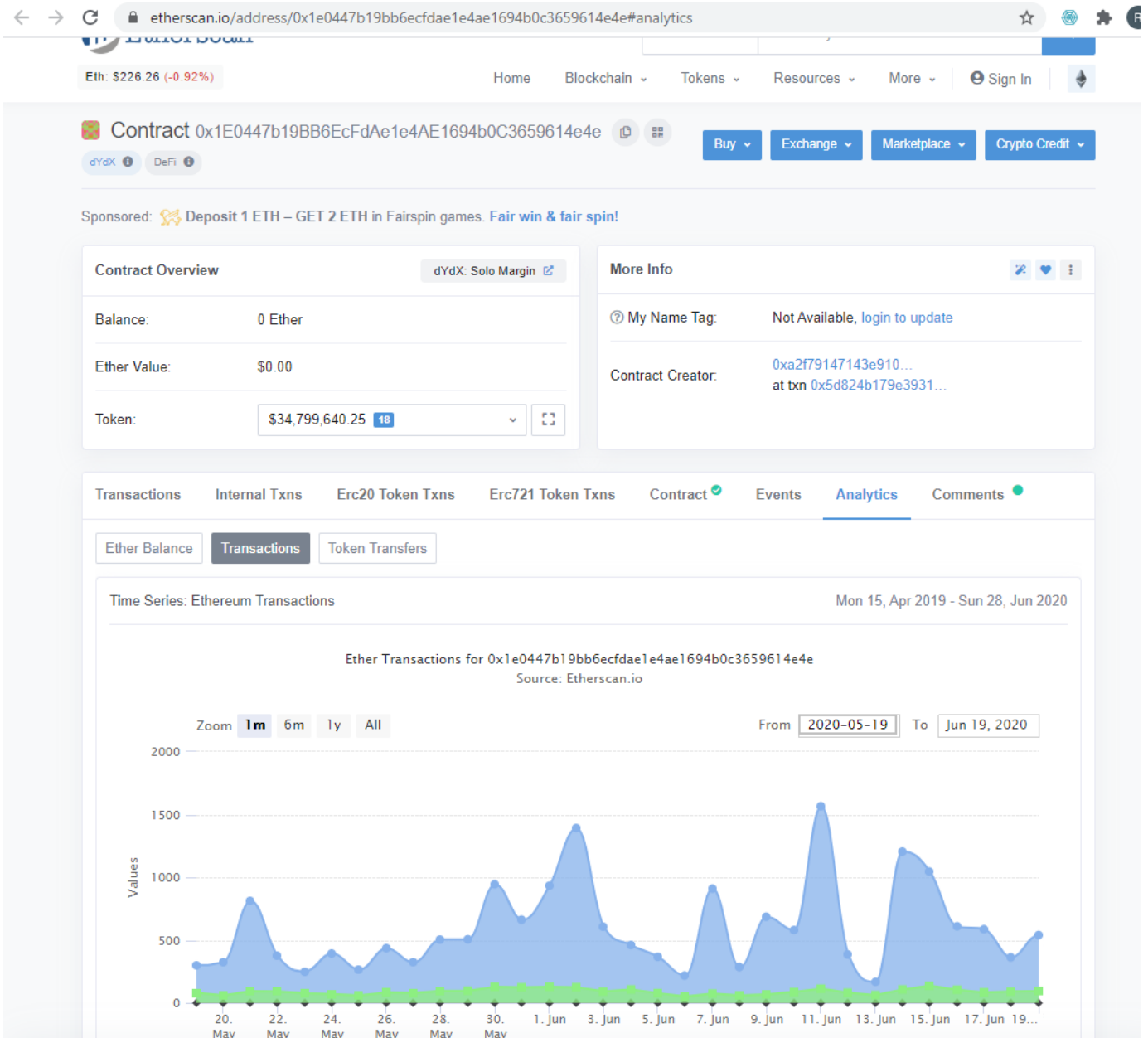
Once unsubscribed, clients will receive a message:

Perpetual Markets

The perpetual markets channel allows clients to receive updates about a particular market.

Subscribing

Code Used Appendix



Example Code Appendix

```

1 /**
2  * @title Actions
3  * @author dYdX
4  *
5  * Library that defines and parses valid Actions
6  */
7 library Actions {
8
9     // ===== Constants =====
10
11     bytes32 constant FILE = "Actions";
12
13     // ===== Enums =====
14
15     enum ActionType {

```

```

16     Deposit, // supply tokens
17     Withdraw, // borrow tokens
18     Transfer, // transfer balance between accounts
19     Buy, // buy an amount of some token (externally)
20     Sell, // sell an amount of some token (externally)
21     Trade, // trade tokens against another account
22     Liquidate, // liquidate an undercollateralized or expiring account
23     Vaporize, // use excess tokens to zero-out a completely negative account
24     Call // send arbitrary data to an address
25 }
26
27 enum AccountLayout {
28     OnePrimary,
29     TwoPrimary,
30     PrimaryAndSecondary
31 }
32
33 enum MarketLayout {
34     ZeroMarkets,
35     OneMarket,
36     TwoMarkets
37 }
38
39 // ===== Structs =====
40
41 /*
42  * Arguments that are passed to Solo in an ordered list as part of a single
43  * Each ActionArgs has an ActionType which specifies which action struct
44  * is parsed into before being processed.
45  */
46 struct ActionArgs {
47     ActionType ActionType;
48     uint256 accountId;
49     Types.AssetAmount amount;
50     uint256 primaryMarketId;
51     uint256 secondaryMarketId;
52     address otherAddress;
53     uint256 otherAccountId;
54     bytes data;
55 }
56
57 // ===== Action Types =====
58
59 /*
60  * Moves tokens from an address to Solo. Can either repay a borrow or provide
61  */
62 struct DepositArgs {
63     Types.AssetAmount amount;
64     Account.Info account;
65     uint256 market;
66     address from;
67 }
68
69 /*
70  * Moves tokens from Solo to another address. Can either borrow tokens or

```

```
71     * previously supplied.
72     */
73     struct WithdrawArgs {
74         Types.AssetAmount amount;
75         Account.Info account;
76         uint256 market;
77         address to;
78     }
79
80     /*
81     * Transfers balance between two accounts. The msg.sender must be an operator.
82     * The amount field applies to accountOne.
83     * This action does not require any token movement since the trade is done on-chain.
84     */
85     struct TransferArgs {
86         Types.AssetAmount amount;
87         Account.Info accountOne;
88         Account.Info accountTwo;
89         uint256 market;
90     }
91
92     /*
93     * Acquires a certain amount of tokens by spending other tokens. Sends the msg.value to
94     * specified exchangeWrapper contract and expects makerMarket tokens in return.
95     * applies to the makerMarket.
96     */
97     struct BuyArgs {
98         Types.AssetAmount amount;
99         Account.Info account;
100        uint256 makerMarket;
101        uint256 takerMarket;
102        address exchangeWrapper;
103        bytes orderData;
104    }
105
106    /*
107    * Spends a certain amount of tokens to acquire other tokens. Sends msg.value to
108    * specified exchangeWrapper and expects makerMarket tokens in return.
109    * to the takerMarket.
110    */
111    struct SellArgs {
112        Types.AssetAmount amount;
113        Account.Info account;
114        uint256 takerMarket;
115        uint256 makerMarket;
116        address exchangeWrapper;
117        bytes orderData;
118    }
119
120    /*
121    * Trades balances between two accounts using any external contract that implements the
122    * AutoTrader interface. The AutoTrader contract must be an operator for the makerMarket
123    * which it is trading on-behalf-of). The amount field applies to the makerMarket.
124    * inputMarket. This proposed change to the makerAccount is passed to the takerAccount.
125    * quote a change for the makerAccount in the outputMarket (or will discontinue trading).
```

```

126     * This action does not require any token movement since the trade is done
127     */
128     struct TradeArgs {
129         Types.AssetAmount amount;
130         Account.Info takerAccount;
131         Account.Info makerAccount;
132         uint256 inputMarket;
133         uint256 outputMarket;
134         address autoTrader;
135         bytes tradeData;
136     }
137
138     /*
139     * Each account must maintain a certain margin-ratio (specified globally
140     * below this margin-ratio, it can be liquidated by any other account.
141     * (arbitrageurs) to repay any borrowed asset (owedMarket) of the liquidating
142     * exchange for any collateral asset (heldMarket) of the liquidAccount.
143     * by the price ratio (given by the oracles) plus a spread (specified globally).
144     * account also sets a flag on the account that the account is being liquidated.
145     * anyone to continue liquidating the account until there are no more borrowed
146     * liquidating account. Liquidators do not have to liquidate the entire account.
147     * can liquidate as much as they choose. The liquidating flag allows liquidators
148     * liquidating the account even if it becomes collateralized through price
149     * price movement.
150     */
151     struct LiquidateArgs {
152         Types.AssetAmount amount;
153         Account.Info solidAccount;
154         Account.Info liquidAccount;
155         uint256 owedMarket;
156         uint256 heldMarket;
157     }
158
159     /*
160     * Similar to liquidate, but vaporAccounts are accounts that have only borrowed
161     * remaining. The arbitrageur pays back the negative asset (owedMarket) of the
162     * exchange for a collateral asset (heldMarket) at a favorable spread. If the
163     * liquidAccount has no collateral assets, the collateral must come from the
164     * liquidating account.
165     */
166     struct VaporizeArgs {
167         Types.AssetAmount amount;
168         Account.Info solidAccount;
169         Account.Info vaporAccount;
170         uint256 owedMarket;
171         uint256 heldMarket;
172     }
173
174     /*
175     * Passes arbitrary bytes of data to an external contract that implements
176     * Does not change any asset amounts. This function may be useful for setting
177     * on layer-two contracts for certain accounts without having to make a
178     * transaction for doing so. Also, the second-layer contracts can ensure
179     * from an operator of the particular account.
180     */
181     struct CallArgs {

```



```
181     Account.Info account;
182     address callee;
183     bytes data;
184 }
185
186 // ===== Helper Functions =====
187
188 function getMarketLayout(
189     ActionType actionType
190 )
191     internal
192     pure
193     returns (MarketLayout)
194 {
195     if (
196         actionType == Actions.ActionType.Deposit
197         || actionType == Actions.ActionType.Withdraw
198         || actionType == Actions.ActionType.Transfer
199     ) {
200         return MarketLayout.OneMarket;
201     }
202     else if (actionType == Actions.ActionType.Call) {
203         return MarketLayout.ZeroMarkets;
204     }
205     return MarketLayout.TwoMarkets;
206 }
207
208 function getAccountLayout(
209     ActionType actionType
210 )
211     internal
212     pure
213     returns (AccountLayout)
214 {
215     if (
216         actionType == Actions.ActionType.Transfer
217         || actionType == Actions.ActionType.Trade
218     ) {
219         return AccountLayout.TwoPrimary;
220     } else if (
221         actionType == Actions.ActionType.Liquidate
222         || actionType == Actions.ActionType.Vaporize
223     ) {
224         return AccountLayout.PrimaryAndSecondary;
225     }
226     return AccountLayout.OnePrimary;
227 }
228
229 // ===== Parsing Functions =====
230
231 function parseDepositArgs(
232     Account.Info[] memory accounts,
233     ActionArgs memory args
234 )
235     internal
```

```
236     pure
237     returns (DepositArgs memory)
238     {
239         assert(args.actionType == ActionType.Deposit);
240         return DepositArgs({
241             amount: args.amount,
242             account: accounts[args.accountId],
243             market: args.primaryMarketId,
244             from: args.otherAddress
245         });
246     }
247
248     function parseWithdrawArgs(
249         Account.Info[] memory accounts,
250         ActionArgs memory args
251     )
252         internal
253         pure
254         returns (WithdrawArgs memory)
255     {
256         assert(args.actionType == ActionType.Withdraw);
257         return WithdrawArgs({
258             amount: args.amount,
259             account: accounts[args.accountId],
260             market: args.primaryMarketId,
261             to: args.otherAddress
262         });
263     }
264
265     function parseTransferArgs(
266         Account.Info[] memory accounts,
267         ActionArgs memory args
268     )
269         internal
270         pure
271         returns (TransferArgs memory)
272     {
273         assert(args.actionType == ActionType.Transfer);
274         return TransferArgs({
275             amount: args.amount,
276             accountOne: accounts[args.accountId],
277             accountTwo: accounts[args.otherAccountId],
278             market: args.primaryMarketId
279         });
280     }
281
282     function parseBuyArgs(
283         Account.Info[] memory accounts,
284         ActionArgs memory args
285     )
286         internal
287         pure
288         returns (BuyArgs memory)
289     {
290         assert(args.actionType == ActionType.Buy);
```

```
291     return BuyArgs({
292         amount: args.amount,
293         account: accounts[args.accountId],
294         makerMarket: args.primaryMarketId,
295         takerMarket: args.secondaryMarketId,
296         exchangeWrapper: args.otherAddress,
297         orderData: args.data
298     });
299 }
300
301 function parseSellArgs(
302     Account.Info[] memory accounts,
303     ActionArgs memory args
304 )
305     internal
306     pure
307     returns (SellArgs memory)
308 {
309     assert(args.actionType == ActionType.Sell);
310     return SellArgs({
311         amount: args.amount,
312         account: accounts[args.accountId],
313         takerMarket: args.primaryMarketId,
314         makerMarket: args.secondaryMarketId,
315         exchangeWrapper: args.otherAddress,
316         orderData: args.data
317     });
318 }
319
320 function parseTradeArgs(
321     Account.Info[] memory accounts,
322     ActionArgs memory args
323 )
324     internal
325     pure
326     returns (TradeArgs memory)
327 {
328     assert(args.actionType == ActionType.Trade);
329     return TradeArgs({
330         amount: args.amount,
331         takerAccount: accounts[args.accountId],
332         makerAccount: accounts[args.otherAccountId],
333         inputMarket: args.primaryMarketId,
334         outputMarket: args.secondaryMarketId,
335         autoTrader: args.otherAddress,
336         tradeData: args.data
337     });
338 }
339
340 function parseLiquidateArgs(
341     Account.Info[] memory accounts,
342     ActionArgs memory args
343 )
344     internal
345     pure
```

```
346     returns (LiquidateArgs memory)
347     {
348     assert(args.actionType == ActionType.Liquidate);
349     return LiquidateArgs({
350         amount: args.amount,
351         solidAccount: accounts[args.accountId],
352         liquidAccount: accounts[args.otherAccountId],
353         owedMarket: args.primaryMarketId,
354         heldMarket: args.secondaryMarketId
355     });
356     }
357
358     function parseVaporizeArgs(
359         Account.Info[] memory accounts,
360         ActionArgs memory args
361     )
362         internal
363         pure
364         returns (VaporizeArgs memory)
365     {
366     assert(args.actionType == ActionType.Vaporize);
367     return VaporizeArgs({
368         amount: args.amount,
369         solidAccount: accounts[args.accountId],
370         vaporAccount: accounts[args.otherAccountId],
371         owedMarket: args.primaryMarketId,
372         heldMarket: args.secondaryMarketId
373     });
374     }
375
376     function parseCallArgs(
377         Account.Info[] memory accounts,
378         ActionArgs memory args
379     )
380         internal
381         pure
382         returns (CallArgs memory)
383     {
384     assert(args.actionType == ActionType.Call);
385     return CallArgs({
386         account: accounts[args.accountId],
387         callee: args.otherAddress,
388         data: args.data
389     });
390     }
391 }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	32	11339	1264	1853	8222	518

Comments to Code 1853/8222 = 23%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
TypeScript	55	20465	1955	337	18173	846

Tests to Code 18173/8222 = 221%