Perp.exchange Process Quality Review

Score: 79%

This is a Process Quality Review of the Perpetual Protocol completed on April 13, 2021. It was performed using the Process Review process (version 0.6.2) and is documented here. The review was performed by Lucas of DeFiSafety. Check out our Telegram.

The final score of the review is 79%, a Solid Pass. The breakdown of the scoring is in Scoring Appendix. For our purposes, a pass is 70%.

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, token, 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

Code and Team

This section looks at the code deployed on the Mainnet that gets reviewed and its corresponding software repository. The document explaining these questions is here. This review will answer the questions;

1. Are the executing code addresses readily available? (Y/N)
2. Is the code actively being used? (%)
3. Is there a public software repository? (Y/N)
4. Is there a development history visible? (%)
5. Is the team public (not anonymous)? (Y/N)

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

> ✓ Answer: Yes

They are available at website https://metadata.perp.exchange/production.json as indicated in the Appendix.

Is the code actively being used? (%)

> ⓘ Answer: 70%

Activity is *7* transactions a day on contract *AdminUpgradabilityProxy.sol*, 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/perpetual-protocol/perpetual-protocol

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 341 commits and 1 branch, this is clearly a healthy software repository.

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

How to improve this score

Continue to test and perform other verification activities after deployment, including routine maintenance updating to new releases of testing and deployment tools. A public development history indicates clearly to the public the level of continued investment and activity by the developers on the application. This gives a level of security and faith in the application.

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

✓ Answer: Yes

Information about the team can be found on LinkdIn.

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 software functions documented? (Y/N)
3. Does the software function documentation 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 from software documentation to the implementation in codee (%)

Is there a whitepaper? (Y/N)

✓ Answer: Yes

Location: https://www.notion.so/Strike-Protocol-9049cc65e99246d886a230972d0cbd60

How to improve this score

Ensure the white paper is available for download from your website or at least the software repository. Ideally update the whitepaper to meet the capabilities of your present application.

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

✓ Answer: Yes

Perpetual exchange offers a multitude of resources, such as two different ABIs, as well as a development guide. However, neither of these exist for the exclusive purpose of documenting the functions in the code.

How to improve this score

Write the document based on the deployed code. For guidance, refer to the SecurEth System Description Document.

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

Answer: 30%

The ABI and javascript API code were all there but there was not really any functional code documentation.

Guidance:

100%	All contracts and functions documented
80%	Only the major functions documented
79-1%	Estimate of the level of software documentation
0%	No software documentation

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: 30%

Code examples are in the Appendix. As per the SLOC, there is 30% commenting to code (CtC).

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 from software documentation to the implementation in code (%)

Answer: 0%

As there is no real software documentation, there is no real software traceability.

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 traceability from requirements to code such that it is clear where each requirement is coded. For reference, check the SecurEth guidelines on traceability.

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%

with a TtC of 240%, there is clearly a reasonable set of tests.

This score is guided by the Test to Code ratio (TtC). Generally a good test to code ratio is over 100%. However the reviewers best judgement is the final deciding factor.

Guidance:

100%	TtC > 120% Both unit and system test visible
80%	TtC > 80% Both unit and system test visible
40%	TtC < 80% Some tests visible
0%	No tests obvious

How to improve this score

This score can improve by adding tests to fully cover the code. Document what is covered by traceability or test results in the software repository.

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

> Answer: 50%

There is no indication of code coverage but there is clearly a reasonably complete set of tests.

Guidance:
100% - Documented full coverage
99-51% - Value of test coverage from documented results
50%   - No indication of code coverage but clearly there is a reasonably complete set of tests
30%   - Some tests evident but not complete
0%   -   No test for coverage seen

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

Location: **https://github.com/perpetual-protocol/perpetual-protocol**

Packaged with the deployed code (Y/N)

✓ Answer: Yes

Report of the results (%)

⚠ Answer: 0%

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%

There is no formal verification testing evident.

Stress Testing environment (%)

✓ Answer: 100%

Location: https://docs.perp.fi/sdk-documentation/smart-contract-and-walletaddresses#perpetual-protocol-rinkeby-testnet-smart-contracts.

Audits

✓ Answer: 90%

PeckShield conducted an audit on september 7th, 2020.

Consensys Diligence also apparently conducted an audit (the link is dead)

Perpetual protocol was released on december 14th.

Guidance:

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 OR smart contract address' not found, question 1 (0%)

Appendices

Author Details

The author of this review is Rex of DeFi Safety.

Email : rex@defisafety.com Twitter : @defisafety

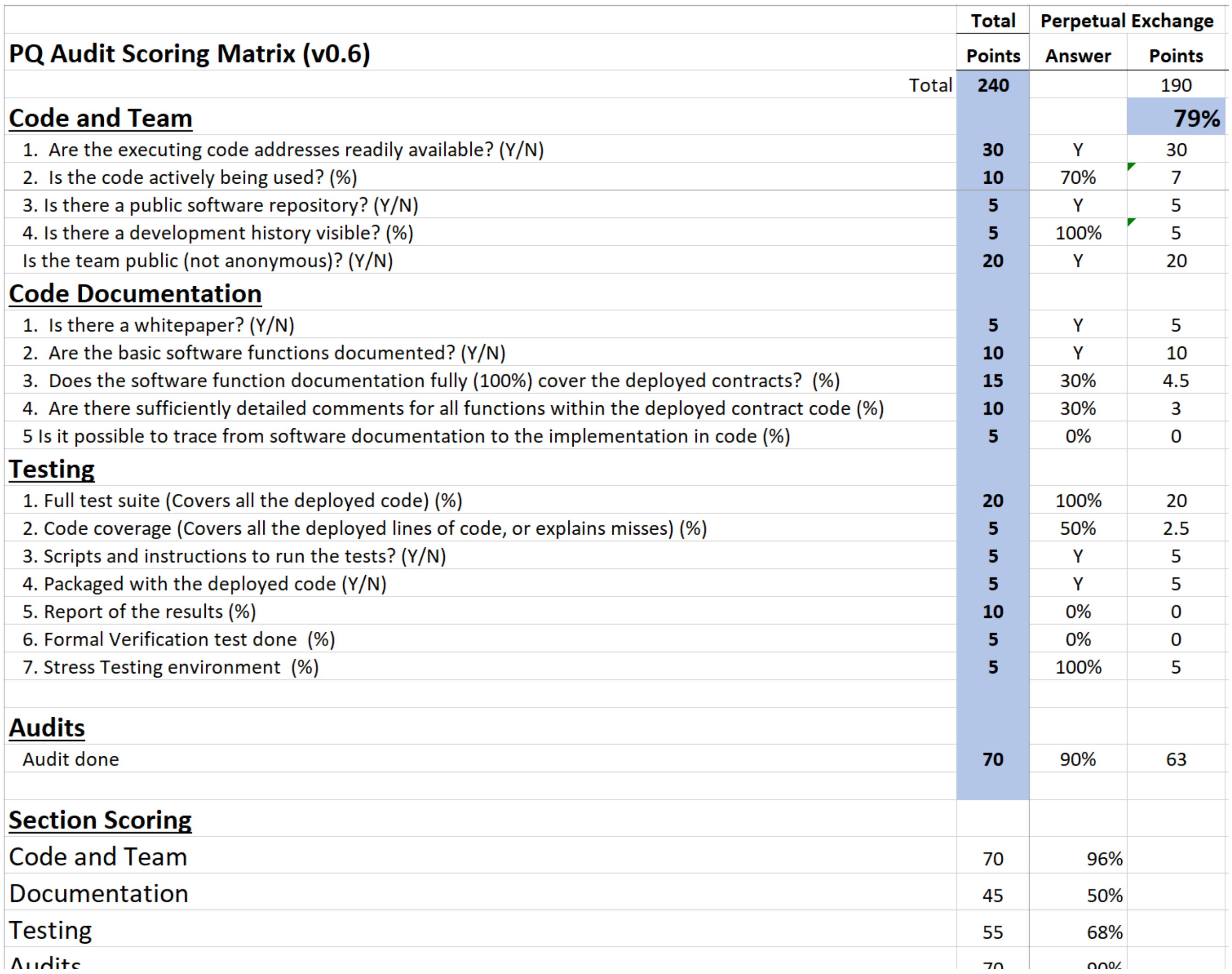
I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

I was coaxed to go to EthDenver 2018 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 Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

DeFiSafety is my full time gig and we are working on funding vehicles for a permanent staff.

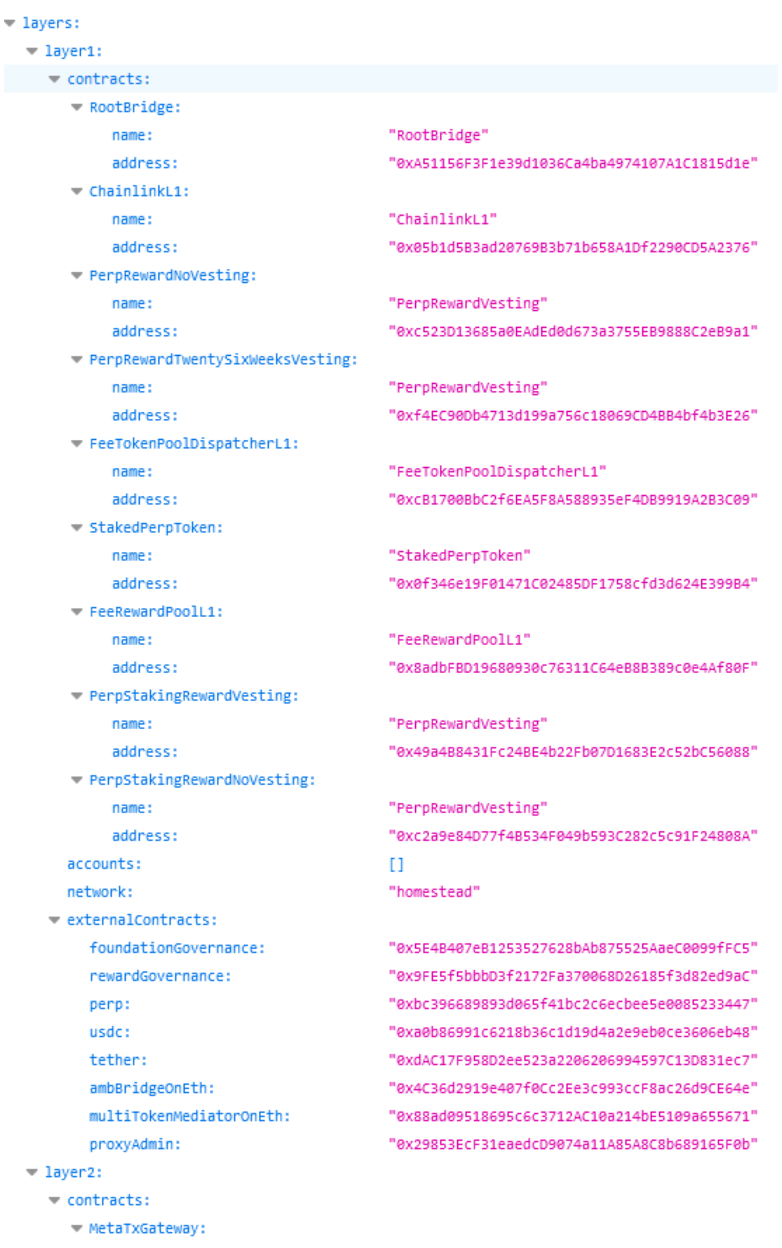
Scoring Appendix



	Total	Perpetual Exchange	
PQ Audit Scoring Matrix (v0.6)	**Points**	**Answer**	**Points**
Total	**240**		190
Code and Team			**79%**
1. Are the executing code addresses readily available? (Y/N)	**30**	Y	30
2. Is the code actively being used? (%)	**10**	70%	7
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**	30%	4.5
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**10**	30%	3
5 Is it possible to trace from software documentation to the implementation in code (%)	**5**	0%	0
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**	50%	2.5
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**	0%	0
6. Formal Verification test done (%)	**5**	0%	0
7. Stress Testing environment (%)	**5**	100%	5
Audits			
Audit done	**70**	90%	63
Section Scoring			
Code and Team	70	96%	
Documentation	45	50%	
Testing	55	68%	
Audits	70	90%	

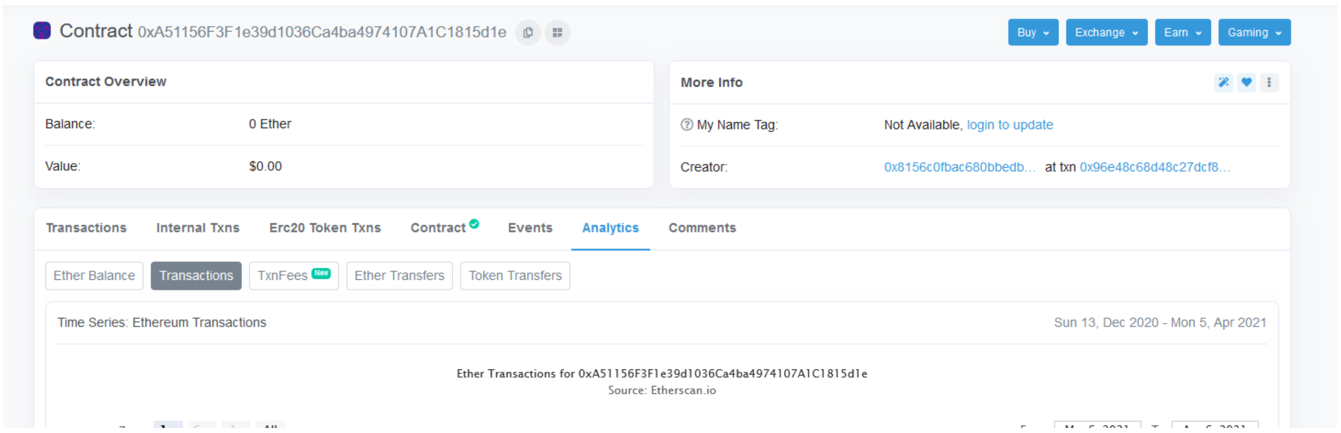
Audits	70	90%

Executing Code Appendix



```
layers:
  layer1:
    contracts:
      RootBridge:
        name:                                  "RootBridge"
        address:                               "0xA51156F3F1e39d1036Ca4ba4974107A1C1815d1e"
      ChainlinkL1:
        name:                                  "ChainlinkL1"
        address:                               "0x05b1d5B3ad20769B3b71b658A1Df2290CD5A2376"
      PerpRewardNoVesting:
        name:                                  "PerpRewardVesting"
        address:                               "0xc523D13685a0EAdEd0d673a3755EB9888C2eB9a1"
      PerpRewardTwentySixWeeksVesting:
        name:                                  "PerpRewardVesting"
        address:                               "0xf4EC90Db4713d199a756c18069CD4BB4bf4b3E26"
      FeeTokenPoolDispatcherL1:
        name:                                  "FeeTokenPoolDispatcherL1"
        address:                               "0xcB1700BbC2f6EA5F8A588935eF4DB9919A2B3C09"
      StakedPerpToken:
        name:                                  "StakedPerpToken"
        address:                               "0x0f346e19F01471C02485DF1758cfd3d624E399B4"
      FeeRewardPoolL1:
        name:                                  "FeeRewardPoolL1"
        address:                               "0x8adbFBD19680930c76311C64eB8B389c0e4Af80F"
      PerpStakingRewardVesting:
        name:                                  "PerpRewardVesting"
        address:                               "0x49a4B8431Fc24BE4b22Fb07D1683E2c52bC56088"
      PerpStakingRewardNoVesting:
        name:                                  "PerpRewardVesting"
        address:                               "0xc2a9e84D77f4B534F049b593C282c5c91F24808A"
    accounts:                                  []
    network:                                   "homestead"
    externalContracts:
      foundationGovernance:                    "0x5E4B407eB1253527628bAb875525AaeC0099fFC5"
      rewardGovernance:                        "0x9FE5f5bbbD3f2172Fa370068D26185f3d82ed9aC"
      perp:                                    "0xbc396689893d065f41bc2c6ecbee5e0085233447"
      usdc:                                    "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48"
      tether:                                  "0xdAC17F958D2ee523a2206206994597C13D831ec7"
      ambBridgeOnEth:                          "0x4C36d2919e407f0Cc2Ee3c993ccF8ac26d9CE64e"
      multiTokenMediatorOnEth:                 "0x88ad09518695c6c3712AC10a214bE5109a655671"
      proxyAdmin:                              "0x29853EcF31eaedcD9074a11A85A8C8b689165F0b"
  layer2:
    contracts:
      MetaTxGateway:
```

Code Used Appendix



Contract 0xA51156F3F1e39d1036Ca4ba4974107A1C1815d1e

Contract Overview

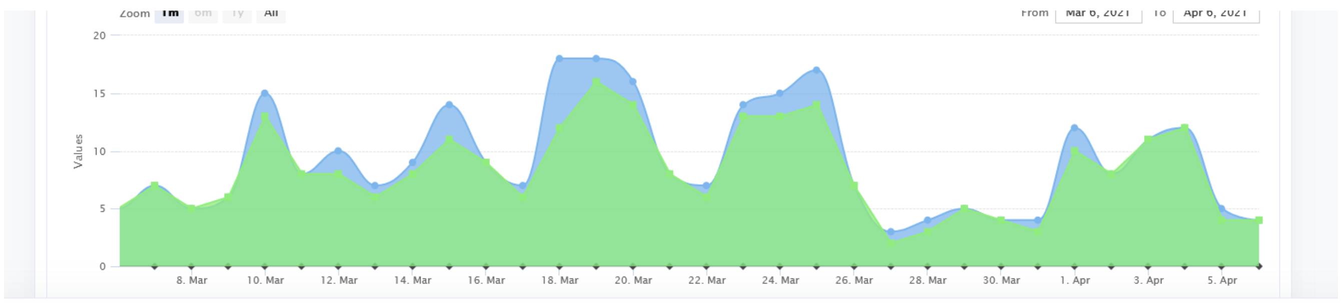
Balance:	0 Ether
Value:	$0.00

More Info

My Name Tag:	Not Available, login to update
Creator:	0x8156c0fbac680bbedb... at txn 0x96e48c68d48c27dcf8...

Time Series: Ethereum Transactions Sun 13, Dec 2020 - Mon 5, Apr 2021

Ether Transactions for 0xA51156F3F1e39d1036Ca4ba4974107A1C1815d1e
Source: Etherscan.io



Example Code Appendix

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity 0.6.9;
pragma experimental ABIEncoderV2;

import { AggregatorV3Interface } from "@chainlink/contracts/src/v0.6/inte
import { BlockContext } from "./utils/BlockContext.sol";
import { PerpFiOwnableUpgrade } from "./utils/PerpFiOwnableUpgrade.sol";
import { RootBridge } from "./bridge/ethereum/RootBridge.sol";
import { Decimal, SafeMath } from "./utils/Decimal.sol";

contract ChainlinkL1 is PerpFiOwnableUpgrade, BlockContext {
    using SafeMath for uint256;
    using Decimal for Decimal.decimal;

    uint256 private constant TOKEN_DIGIT = 10**18;

    event RootBridgeChanged(address rootBridge);
    event PriceFeedL2Changed(address priceFeedL2);
    event PriceUpdateMessageIdSent(bytes32 messageId);
    event PriceUpdated(uint80 roundId, uint256 price, uint256 timestamp);

    //**********************************************************//
    //    The below state variables can not change the order    //
    //**********************************************************//

    // key by currency symbol, eg ETH
    mapping(bytes32 => AggregatorV3Interface) public priceFeedMap;
    bytes32[] public priceFeedKeys;
    RootBridge public rootBridge;
    address public priceFeedL2Address;
    mapping(bytes32 => uint256) public prevTimestampMap;

    //**********************************************************//
    //    The above state variables can not change the order    //
    //**********************************************************//

    //▼▼▼▼▼▼▼▼ add state variables below ▼▼▼▼▼▼▼▼//

    //▲▲▲▲▲▲▲▲▲ add state variables above ▲▲▲▲▲▲▲▲▲//
    uint256[50] private __gap;

    //
    // FUNCTIONS
    //
    function initialize(address _rootBridge, address _priceFeedL2) public
        __Ownable_init();
        setRootBridge(_rootBridge);
        setPriceFeedL2(_priceFeedL2);
    }

    function setRootBridge(address _rootBridge) public onlyOwner {
        requireNonEmptyAddress(_rootBridge);
        rootBridge = RootBridge(_rootBridge);
        emit RootBridgeChanged(_rootBridge);
    }

    function setPriceFeedL2(address _priceFeedL2) public onlyOwner {
        requireNonEmptyAddress(_priceFeedL2);
        priceFeedL2Address = _priceFeedL2;
        emit PriceFeedL2Changed(_priceFeedL2);
    }

    function addAggregator(bytes32 _priceFeedKey, address _aggregator) ex
        requireNonEmptyAddress(_aggregator);
        if (address(priceFeedMap[_priceFeedKey]) == address(0)) {
            priceFeedKeys.push(_priceFeedKey);
        }
        priceFeedMap[_priceFeedKey] = AggregatorV3Interface(_aggregator);
    }

    function removeAggregator(bytes32 _priceFeedKey) external onlyOwner {
        requireNonEmptyAddress(address(getAggregator(_priceFeedKey)));
        delete priceFeedMap[_priceFeedKey];

        uint256 length = priceFeedKeys.length;
        for (uint256 i; i < length; i++) {
            if (priceFeedKeys[i] == _priceFeedKey) {
                // if the removal item is the last one, just `pop`
                if (i != length - 1) {
                    priceFeedKeys[i] = priceFeedKeys[length - 1];
                }
                priceFeedKeys.pop();
                break;
            }
        }
    }

    function getAggregator(bytes32 _priceFeedKey) public view returns (Ag
        return priceFeedMap[_priceFeedKey];
    }

    //
    // INTERFACE IMPLEMENTATION
    //

    function updateLatestRoundData(bytes32 _priceFeedKey) external {
        AggregatorV3Interface aggregator = getAggregator(_priceFeedKey);
        requireNonEmptyAddress(address(aggregator));

        (uint80 roundId, int256 price, , uint256 timestamp, ) = aggregato
        require(timestamp > prevTimestampMap[_priceFeedKey], "incorrect t
        require(price >= 0, "negative answer");

        uint8 decimals = aggregator.decimals();

        Decimal.decimal memory decimalPrice = Decimal.decimal(formatDecim
        bytes32 messageId = rootBridge.updatePriceFeed(
            priceFeedL2Address,
            _priceFeedKey,
            decimalPrice,
            timestamp,
            roundId
        );
        emit PriceUpdateMessageIdSent(messageId);
        emit PriceUpdated(roundId, decimalPrice.toUint(), timestamp);

        prevTimestampMap[_priceFeedKey] = timestamp;
    }

    //
    // REQUIRE FUNCTIONS
    //

    function requireNonEmptyAddress(address _addr) internal pure {
        require(_addr != address(0), "empty address");
    }

    //
    // INTERNAL VIEW FUNCTIONS
    //
    function formatDecimals(uint256 _price, uint8 _decimals) internal pur
        return _price.mul(TOKEN_DIGIT).div(10**uint256(_decimals));
    }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	41	5953	865	1195	3893	433

Comments to Code 1195/3893 = 30%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	33	12950	1827	1763	9360	85

Tests to Code 9360/3893 = 240%