Melon Protocol Specification

Melonport AG
team@melonport.com

***Abstract*—The abstract goes here.**

I. Introduction

Fund administration consists of four parts. Fund Custodian, Fund Accountant, KYC/AML and Risk Management. In the following paper we will describe how fund administration can be implemented using smart contracts.

II. Background

A. Custodian

B. Decentralized Execution

1) Assets: An example for such computer code is known as the ERC20 standard. Essentially a small (¡100 lines of code) piece of software implementing a bitcoin-like cryptocurrency.

2) Exchanges: Another example are exchanges. Given above concept one now can implement computer code one how exchange of above assets can be facilitated in a decentralised way.

3) Investment Funds: Using the concept that smart contracts can be custodian of assets. Once a smart contract holds assets there needs the be custom functions built into the smart-contract in order to spend those assets again. Lack of such function means that assets are forever lost.

we can now build smart contracts that act as fully functional investment funds.

To manage their holdings these investment funds use decentralised exchanges to buy and sell assets.

III. Investment Funds Design

A. Governance

The main functionality of the Governance contract is to add new protocol versions such as this Version contract and to shut down existing versions once they become obsolete.

Adding new protocol version is done by anyone proposing a version to be added and is executed once authority consensus has been established.

Shutting down an existing protocol version is done by anyone proposing a version to be shut down and is executed once authority consensus has been established.

Shutting down a version disables the ability to setup new funds using this version and enables anyone to shut down any existing funds of this version.

B. Investment Fund

Melon fund as a smart contract that acts as the custodian and accountant. Broadly speaking a Melon fund should be capable of accepting/executing subscription and redemption requests, make orders on an exchange, take orders on an exchange and receiving management and performance rewards.

A Melon fund is created by triggering the Solidity function version.setupFund(P) on the latest protocol version contract. Where the input parameters P are:

C. Modules

D. Data Feed

Data feeds are instances of smart contracts that route external data which include asset prices into smart contracts. These inputs could be staked and validated on-chain in order to prevent incorrect / manipulative inputs.

Data feed uses update to periodically update prices of all the assets in one pass.

getReferencePrice can be used to query the price of any given asset in reference to the reference asset. Reference asset or quote asset is set during contract creation.

Algorithm 1 Update algorithm

```
1: procedure UPDATE($assets, prices$)    ▷ Update asset prices
2:     $i \leftarrow nextUpdateId$
3:     for $j = 0$; $j < assets.length$; $j \leftarrow j + 1$ do
4:         $dataHistory[i][assets[j]] \leftarrow prices[j]$ ▷ To keep track of historial price updates
5:     end for
6:     $nextUpdateId \leftarrow nextUpdateId + 1$
7: end procedure
```

E. Participation

F. Risk Management

G. Interaction

1) Setup of a new Melon fund: A new Melon fund can be setup by specifying the following parameters via setupFund function of the version contract:

Name	Data Type	Description
name	string	A human readable name of the fund
referenceAsset	address	Asset against which performance reward is measured against
managementRewardRate	uint	Reward rate in referenceAsset per delta improvement
performanceRewardRate	uint	Reward rate in referenceAsset per managed seconds
participation	address	Participation module
riskMgmt	address	Risk management module
sphere	address	Sphere module

2) Participant invests in a Melon fund: A participant starts to invest in a fund F by first creating a subscription request R. Parameters to be specified are:

Name	Data Type	Description
giveQuantity	uint	Quantity of Melon tokens to invest
shareQuantity	uint	Quantity of fund shares to receive
incentiveQuantity	uint	Quantity of Melon tokens to award the entity executing the request

R parameters are checked against restriction rules specified in the participation module P by the boolean function P.isSubscriptionPermitted (E.g Participant being an attested Uport identity).

R is then executed in by any entity via F.executeRequest after certain conditions are satisfied. These conditions include if currentTimestamp - R.timestamp ¿= DF.INTERVAL (DF refers to datafeed module and INTERVAL corresponds to update frequency value) and if DF.getLastUpdateId ¿= R.lastDataFeedUpdateId + 2. This is to minimize unfair advantage from information asymmetries associated with the investor.

3) Participant redeems from a Melon fund: A participant can redeem from a fund by first creating a redemption request. Parameters to be specified are:

Name	Data Type	Description
shareQuantity	uint	Quantity of fund shares to redeem
receiveQuantity	uint	Quantity of Melon tokens to receive in return
incentiveQuantity	uint	Quantity of Melon tokens to award the entity executing the request

Request parameters are checked against restriction rules specified in the participation module P via the boolean function P.isRedemptionPermitted.

R is then executed in a similar way as mentioned earlier.

4) Manager makes an order: Manager can make an order by specifying asset pair, sell and buy quantities as parameters. Asset pair is checked against datafeed module DF through the function DF.existsData. Order parameters are then checked against restriction rules specified in the risk management module R via the boolean function R.isMakePermitted.

The specified quantity of the asset is given allowance to the selected exchanging via ERC20's approve function.

Order is then placed on the selected exchange through the exchangeAdapter contract E via E.makeOrder by specifying the exchange and order parameters as parameters.

The order is filled on the selected exchange (In future, can be any compatible decentralized exchange like OasisDex, Kyber, e.t.c) when the price is met.

5) Manager takes an orders: Manager can take an order by specifying an order id and quantity as parameters. Asset pair is checked against datafeed module DF through the function DF.existsData. Order parameters are then checked against restriction rules specified in the risk management module R via the boolean function R.isTakePermitted.

The specified quantity of the asset is given allowance to the selected exchanging via ERC20's approve function.

Order id must correspond to a valid, existing order on the selected exchange. Order is then placed on the selected exchange through the exchangeAdapter contract E via E.takeOrder by specifying the exchange and order parameters as parameters.

6) Manager converts rewards into shares: Manager rewards in the form of ownerless shares of the fund F can be allocated to the manager via F.convertUnclaimedRewards function. Ownerless shares refer to the quantity of shares, representing unclaimed rewards by the Manager such as rewards for managing the fund and for performance. First internal stats of F are calculated using F.performCalculations function. The quantity of unclaimedRewards is calculated internally using calcUnclaimedRewards function.

A share quantity of unclaimedRewards * gav (from Calculations) is assigned to the manager.

7) Manager shuts down the fund: A Manager can shut down a fund F he owns via F.shutdown function.

Investing, redemption (Only in reference asset, investors can still redeem in the form of percentage of held assets), managing, making / taking orders, convertUnclaimedRewards are rendered disabled.

IV. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.