USDX WHITEPAPER

Jan 2025

Alexander Reed
alex@alexandros-securities.com

ABSTRACT

The USDX protocol provides a convenient USD denominated way for people to earn low risk yield on chain. The primary protocol token (USDX) provides a clear unit of value and accumulates generated yield.

EXECUTIVE SUMMARY

USDX is a stablecoin designed to maintain a 1 USD valuation backed by short-term US Treasury bills. The yield generated by these T-bills is automatically distributed to USDX token holders through rebasing. This feature keeps the price of USDX stable while steadily increasing the balance of USDX for token holders.

X Protocol hosts the primary market that establishes the price peg. KYC-verified users can deposit collateral, either USDC or fiat currency, on our platform. In exchange, they receive USDX and can choose to redeem USDX for collateral or withdraw USDX to any Ethereum address. Since X Protocol offers an exchange rate of one dollar per USDX in both directions, any price movement in the external (secondary) market can be traded in the primary market to re-establish the peg.

USDX is a rebasing ERC-20 token with a set of additional features, including pausing, blocking/unblocking accounts, role-based access control, and upgradability. The contract's primary objective is to reflect the T-bills' annual percentage yield (APY) within the token's value. It achieves this through a reward multiplier rebasing mechanism, where the addRewardMultiplier function is called daily to adjust the reward multiplier, guaranteeing an accurate representation of yield from the underlying assets.

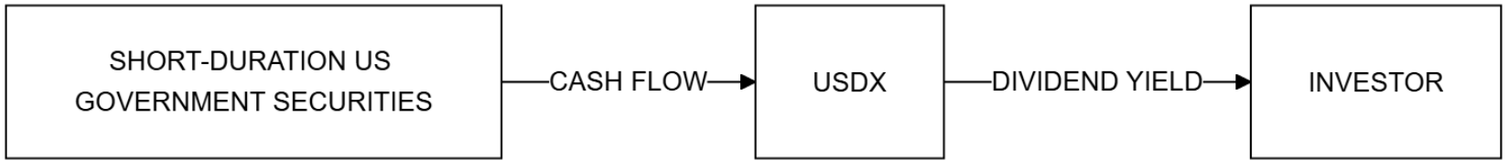
Acknowledging the complexities of handling rebasing tokens in the DeFi ecosystem, the wUSDX contract serves as a wrapped token, simplifying integration while preserving stability. The wUSDX contract is an ERC-462 token vault, enabling users to deposit USDX in exchange for wUSDX tokens. The USDX tokens are rebasing, whereas the wUSDX tokens are non-rebasing, making wUSDX ideal for seamless integration with protocols in the DeFi ecosystem.

The wUSDX contract incorporates the ERC-2612 permit functionality, allowing the use of signatures to grant token allowances. Additionally, the close relationship between the wUSDX and USDX contracts is also worth noting; the wUSDX contract leverages the account block list from the USDX contract to govern transfers and, specifically, prevent transfers from accounts included in the block list.
Finally, the wUSDX token transfers can be paused in two ways: either by being paused directly from within the wUSDX contract or in the event the USDX contract is paused.

TABLE OF CONTENTS

1 INTRODUCTION

The USDX token is a dollar denominated digital token. The protocol mints and redeems 1 USDX token for $ 1 USD value. USDX tokens earn yield generated by the protocol. The protocol earns yield by investing the funds and tokens used to mint USDX tokens into fixed income assets, initially short duration US Treasury bills.



1.2 MATRIX

SEGMENT	PROBLEM	SOLUTION
B2B Payments	Difficult to track High fees Slow Lack transparency	Stablecoins
Stablecoins	No yield distribution Taxes USA regulatory compliance Depegging	Rebasing Municipal bond underlying Investment Company Act of 1940 Proof of reserves
Liquidity Investing	Yield Stability Liquidity	Yield-bearing collateral

1.3 KEY CLIENTS

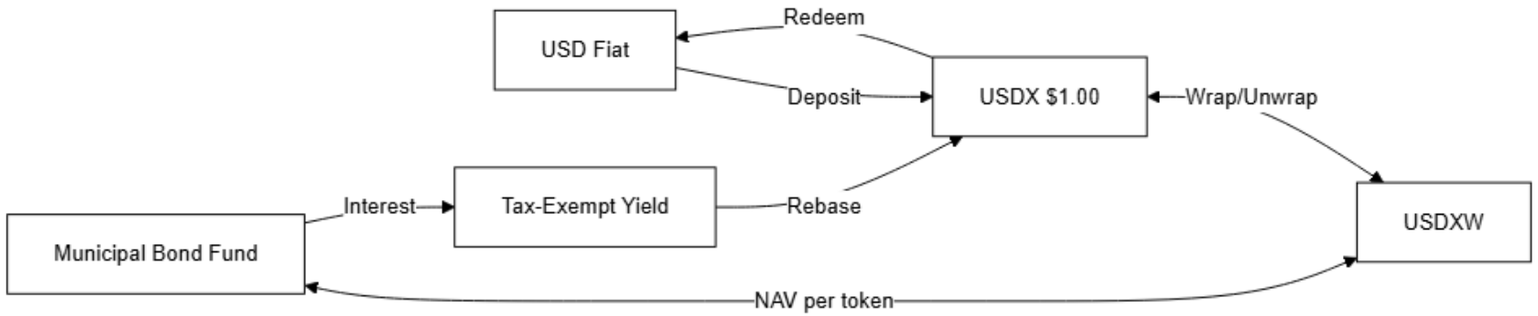
SEGMENT	USE CASES	BENEFITS
Mutual Funds	Redemption collateral	-15bps cash drag
Insurance	Float collateral	Tax-free daily yield
Neobanks	Cross-border	-30bps FX fee
Healthcare	Secure payment processors	Prevent $9T breaches
Corporate	Intracompany	Instant 24/7/365 global settlement
Unbanked	Mobile banking	Yield + access

1.x PROJECT REFERENCES

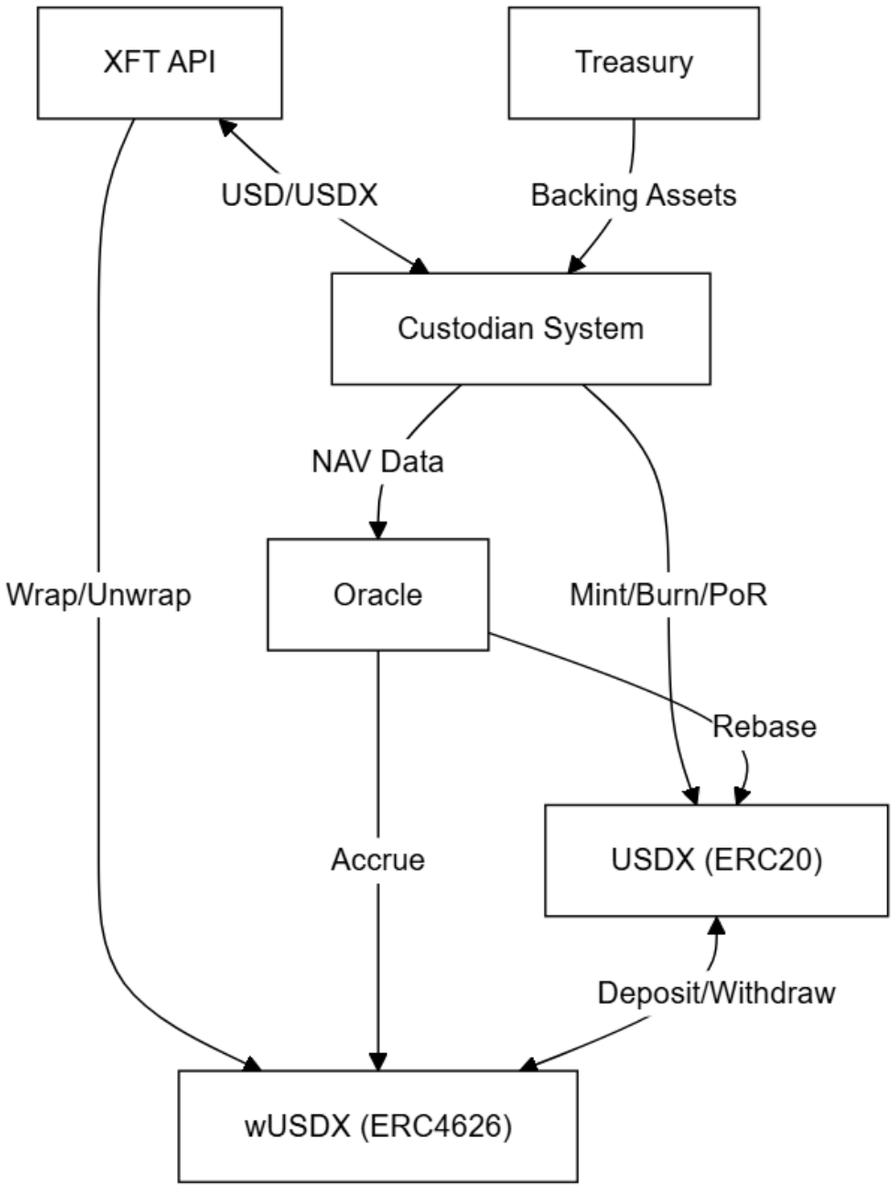
[HTML] How to make a mint, groups.csail.mit.edu/mac/classes/6.805/articles/money/nsamint/nsamint.htm
[HTML] XFT Docs 001, xft001.netlify.app
[Github] XFT Repo 001, github.com/X-Financial-Technologies/assets
[Github] XFT Repo 002, github.com/X-Financial-Technologies/monorepo
[Github] XFT Repo Protocol, github.com/amr080/x-protocol
[HTML] XFT Summary Prospectus, xft-sp001.netlify.app
[mp4] USDX Contract Testing, youtu.be/Veo5T6kqOm8
[mp4] USDX Coverage Report, youtu.be/RbeZ66vL21Y
[mp4] USDX Cash Management Demo, youtu.be/iZYY4Oc_Fac
[HTML] USDX Demo Hub, xft.netlify.app
[TXT] USDX Protocol, xft001.netlify.app/assets/protocol.txt
[Mermaid] High-Level System Architecture, github.com/X-Financial-Technologies/monorepo/blob/main/flow.mermaid
[Mermaid] Contract Architecture and Inheritance, github.com/amr080/x-protocol/blob/main/docs/contract_architecture_inheritance.mermaid

2 SYSTEM OVERVIEW

2.x FLOW OF FUNDS



2.x HIGH-LEVEL SYSTEM ARCHITECTURE



2.3 CORE COMPONENTS

	USDX	wUSDX
Token Type	Rebasing	Accumulating
Yield Accrual Profile	Reflected as a subdivision of USDX tokens into additional USDX tokens daily	Reflected via increasing redemption price (reference token price)
As Treasury Yield Accrues, the Token Price	Remains at $1.00	Increases
Best suited for	Yield-bearing collateral	Buy-and-hold cash management, collateral in smart contracts

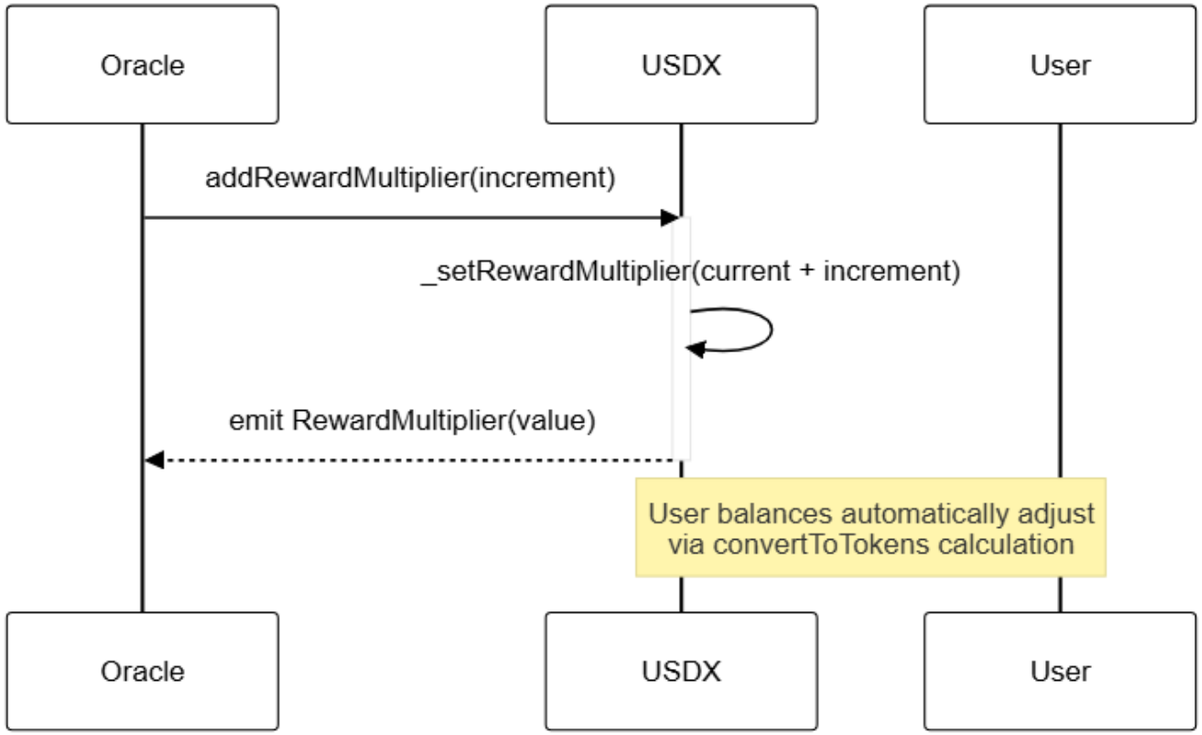
2.3.1 USDX TOKEN

The USDX Token is an ERC20 compliant fungible token. It implements administrative minting, burning, and rebasing functions to manage the supply of the token.

2.3.1.1 REBASING MECHANISM

The USDX token implements a rebasing mechanism to distribute yield to token holders via a reward multiplier system. Advantages:

- Gas-efficient yield distribution
- No transfer operations required
- Automatic balance adjustments
- Precise share-based accounting



1 Oracle calls addRewardMultiplier with yield increment
2 Internal multiplier updated
newMultiplier = current + increment
3 Event emitted with new multiplier value
4 User balances update automatically through conversion formula:
tokens = shares * rewardMultiplier / BASE

2.3.2 WRAPPED USDX TOKEN

The Wrapped USDX Token is an ERC4626 compliant fungible token vault. USDX is deposited into the Wrapped USDX smart contract in order to facilitate interoperability and composability with common blockchain applications and accounting systems. Users can withdraw USDX from the Wrapped USDX contract at any time.

2.3.3 RESERVE VAULT

The Reserves Vault is an onchain account holding the onchain portion of the assets backing the value of USDX including yield generating tokenized assets. The Reserves Vault will primary invest in short duration US Treasury bills directly or indirectly.

2.3.3 MINTER

The Minter handles USDX minting. It maintains list of approved payment tokens and their price oracles. USDX is always minted at 1 USDX = 1 USD. When a user calls ‘deposit‘ on the Minter smart contract, the deposit is processed immediately, sending payment token to the Reserves Vault and minting USDX at the current oracle price for that payment token.

2.3.3 REDEEMER

The Redeemer handles USDX burning. It also maintains a list of approved payment tokens and their price oracles. USDX is always burned at 1 USDX = 1 USD. Burning USDX and sending payment token to the user is a two step process that can take up to 2 business days to process. There may be delays that take up to 2 days to process when liquidating the underlying investments and converting to the requested payment token for distribution. When a user calls ‘requestRedeem‘ on the Redeemer smart contract, the USDX is escrowed on the Redeemer contract and the payment token price is taken at that time and the payment token amount is fixed for that request. The protocol will then fulfill the request if payment token is available in the Reserves Vault, or rebalance assets and tokens to make payment token available before fulfilling the redemption request.

2.3.4 PROOF OF RESERVES

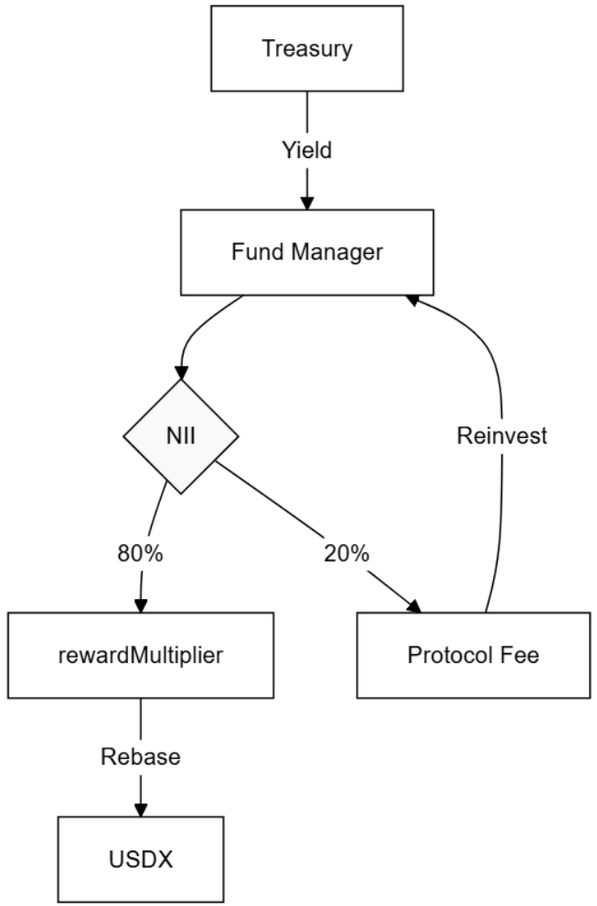
XFT maintains 1-to-1 backing in real time by determining if any security purchases are necessary and executing those purchases for every order request. When a request is received, if the current reserves are not sufficient to maintain at least 100% backing, purchases of the the underlying security are initiated. XFT publishes the latest order activity in real time and current vault balances at regular intervals. XFT vault accounts are held by a custodian and audited at random times by a third party to verify the integrity of accounting and reporting.

3 PROTOCOL ARCHITECTURE

3.1 OVERVIEW

The USDX protocol provides a convenient and effective access point to low risk yield on chain. The protocol is designed to be simple and transparent, and to provide a clear and stable value for users. It is also designed to be flexible and to be able to adapt to changing market conditions and user needs.

3.2 FEES AND BUSINESS MODEL



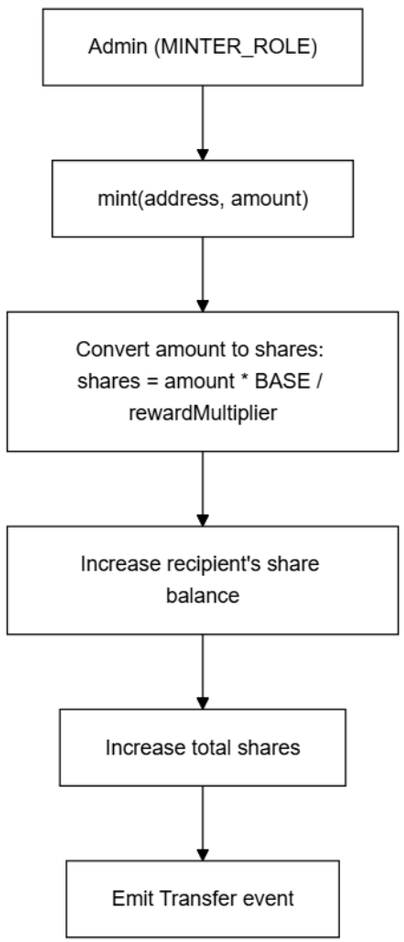
3.x PROTOCOL FLOWS

The USDX protocol implements four core operations: minting, transferring, yield distribution, and wrapping. Each flow represents a key interaction with the protocol.

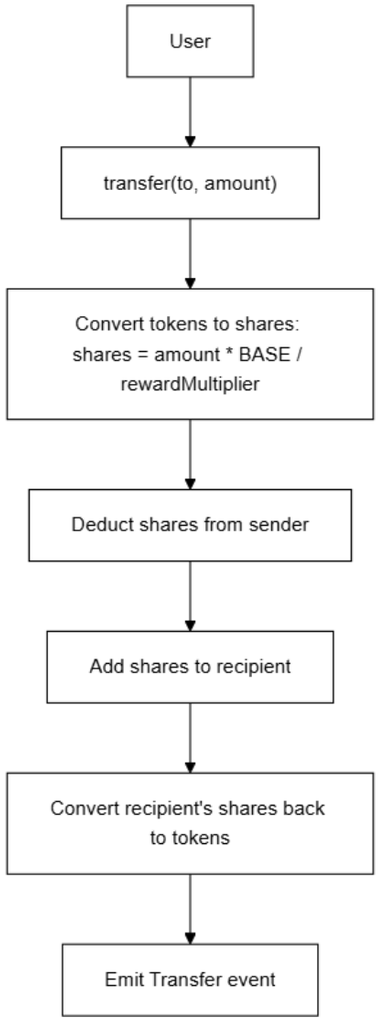
- MINT: User → API → Auth/KYC → Reserve Check → Custodian Webhook → Smart Contract → Mint Tokens
- YIELD: Fund Admin → Oracle → Rate Service → Smart Contract → Update Balances (Daily)
- REDEMPTION: User → API → Burn Request → Reserve Deduction → Custodian Payout → Blockchain Update

3 MINTING

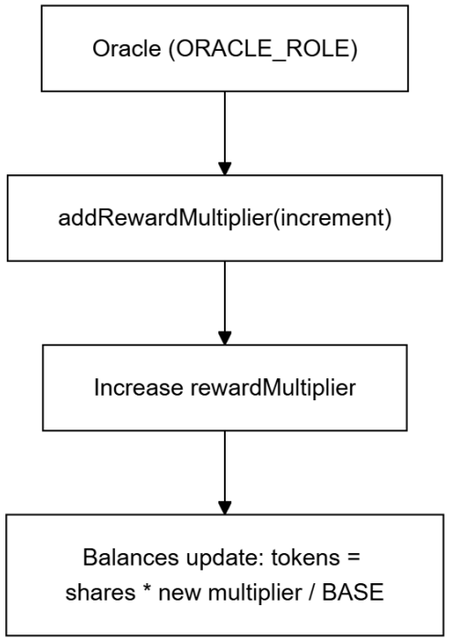
Shows how new USDX tokens are created and distributed.



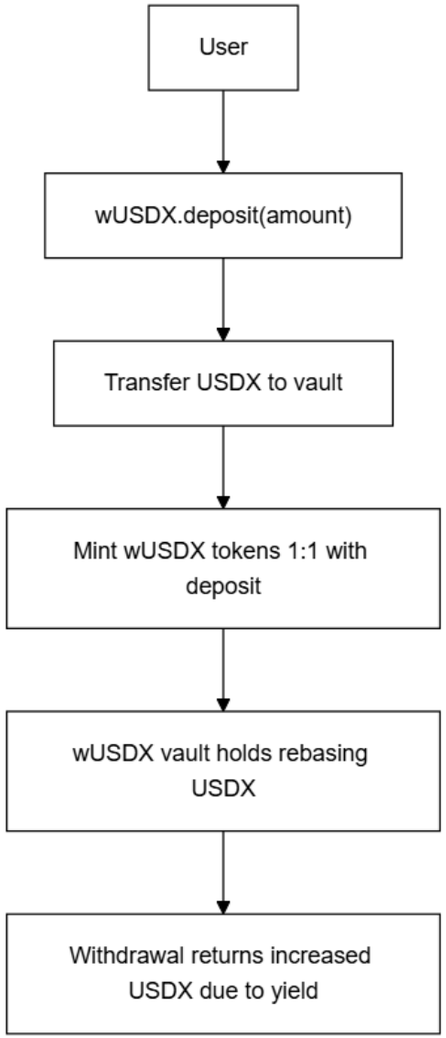
TRANSFER
Shows internal share-based accounting during token transfers.



YIELD DISTRIBUTION
Demonstrates how yield is automatically distributed via the multiplier.



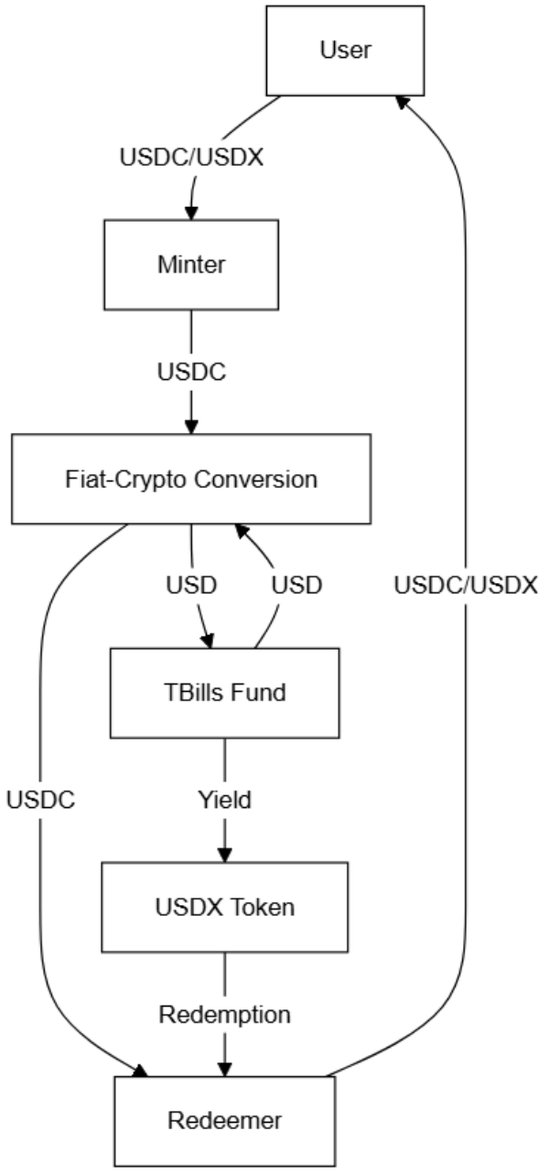
WRAPPING
Illustrates the interaction between USDX and wUSDX.



3.2 USDX SEQUENCES

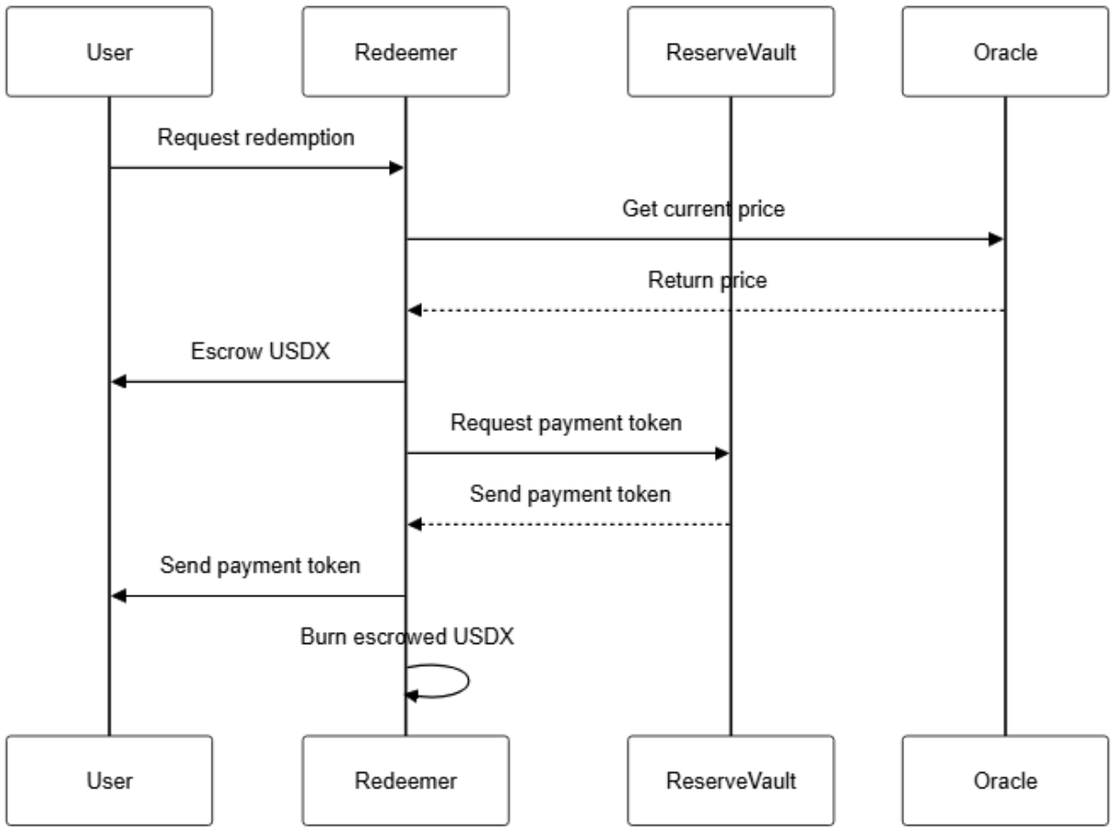
FLOW	USER	TECH
Mint	1 Admin with MINTER_ROLE initiates mint operation 2 Converts token amount to shares using current multiplier rate 3 Updates internal accounting (shares and total shares) 4 Records transaction with Transfer event	1 MINTER_ROLE calls mint(to, amount) 2 _beforeTokenTransfer invoked with from = address(0) 3 convertToShares computes shares = (amount * _BASE) / rewardMultiplier 4 _totalShares and _shares[to] are incremented 5 _afterTokenTransfer emits Transfer(address(0), to, amount)
Transfer	1 User initiates transfer with token amount 2 System converts to shares for internal movement 3 Adjusts share balances between accounts 4 Converts back to tokens for recipient	1 User calls transfer(to, amount) invoking _transfer(msg.sender, to, amount) 2 _beforeTokenTransfer checks isBlocked and paused state 3 Token amount converted to shares: shares = (amount * _BASE) / rewardMultiplier 4 Deducts shares from sender and adds to recipient 5 _afterTokenTransfer emits Transfer(from, to, amount)
Yield	1 Oracle updates reward multiplier 2 Increases global multiplier value 3 All token balances automatically increase	1 ORACLE_ROLE calls addRewardMultiplier(increment) 2 _setRewardMultiplier validates increment and updates rewardMultiplier 3 Emits RewardMultiplier event 4 Balances update automatically via convertToTokens(shares) using new multiplier
Wrap	1 User deposits USDX into wUSDX vault 2 Receives equivalent wUSDX tokens 3 Vault holds rebasing USDX 4 Withdrawal returns original + yield	1 User calls wUSDX.deposit(assets) 2 USDX.transferFrom transfers tokens to wUSDX vault 3 ERC4626.deposit converts assets to vault shares and mints tokens 1:1 4 Vault holds rebasing USDX; withdrawal returns USDX with accrued yield

3.2.1 MINT / REDEEM OVERVIEW



1 User sends USDC/USDX to Minter
2 Minter forwards USDC to Fiat-Crypto Conversion
3 FCC converts USDC to USD for TBills Fund
4 TBills returns USD to FCC
5 FCC sends USDC to Redeemer
6 TBills yields USDX tokens
7 USDX redeems through Redeemer
8 Redeemer returns USDC/USDX to User

USDX REDEMPTION



3.x.x USDX FUNDING STEPS

1 Investors deposit cash with USDX's custodian.
2 USDX selects and invests in money market securities according to the Investment Policy of the Fund.
3 Purchased securities are held at USDX's custodian on behalf of the Investors.
4 Returns on the portfolio may either be paid to investors periodically or reinvested in the fund.

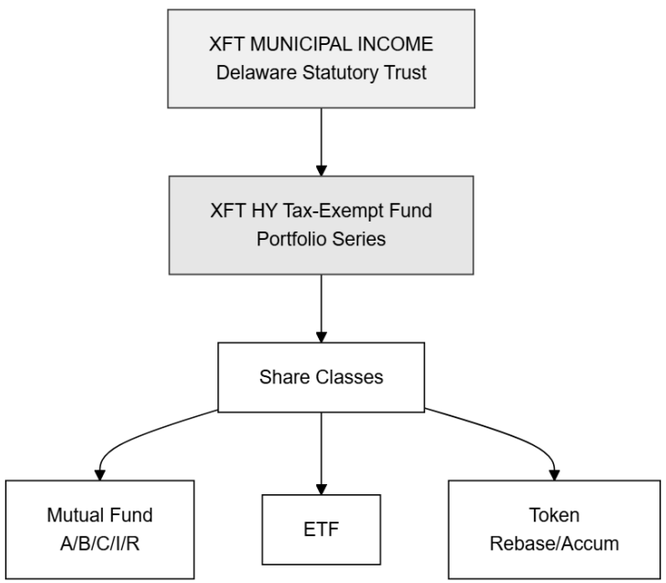
3.x.x.x MONEY MARKET FUND ENGINE

1 Investors deposit cash with MMF's custodian.
2 The MMF selects and invests in money market securities according to the Investment Policy of the Fund.
3 Purchased securities are held at the MMF's custodian on behalf of the Investors.
4 Returns on the portfolio may either be paid to investors periodically or reinvested in the fund.

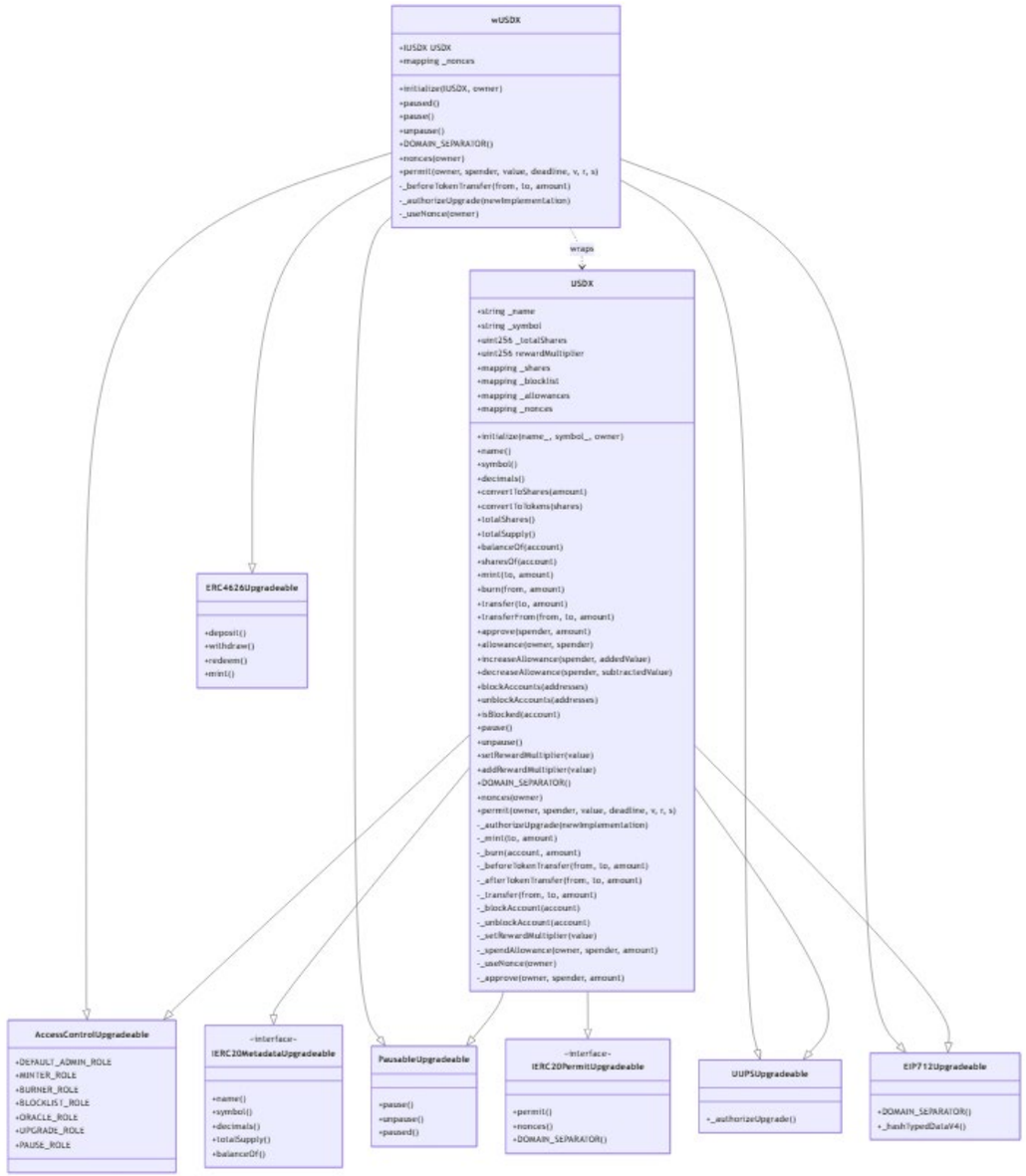
4 INVESTMENT COMPANY
4.1 PROSPECTUS TERMS
Structure Tender
Listed N/A
CEF Ticker XFT
Token Ticker USDX
NAV Ticker USDXW
Location NY
State of Incorporation DE
Maryland Act TRUE
Staggered Board YES
Distribution Policy Managed
Distribution Frequency D
Internally / Externally Managed E
Investment Objective High level of tax-free current income
Minimum Investment 2500
Suitability Non-Accredited
NAV Frequency Continuous
Incentive Fee TRUE
Incentive Type NNI
Incentive Details 20% of the incentive fee is calculated and payable quarterly in arrears based upon the Fund's "pre-incentive fee net investment income" for the immediately preceding quarter
Repurchase Frequency Discretion
Repurchased Securities Tokens
Securities Act FALSE
Non-diversified FALSE
Security Lending Authorized FALSE
Fund Lend Securities FALSE
Expense Limitation Arrangement TRUE
Expenses Waived TRUE
Rights Offerings FALSE
Secondary Offerings FALSE
Default on long term debt FALSE
Dividends in Arrears FALSE
Modified Securities FALSE
PORTFOLIO
Main Group National Muni Bond Funds
Sub-Group National Municipal (tax-free) Bond
Category Bond Funds
Sub-Category Muni Bond
Benchmark CEFA's National Municipal Bond Index
PARTIES
Sponsor XFT

Advisor Franklin Templeton
Subadvisor Guggenheim
Custodian The Bank of New York Mellon
Transfer Agent XFT
Underwriters Oppenheimer & Co. Inc.
Administrators The Bank of New York Mellon
Accountants KPMG LLP
Pricing Services ICE
Fiat Onramp Visa, Brex, GS, RBC

4.2 STRUCTURE



APPENDIX
CONTRACT ARCHITECTURE AND INHERITANCE



Enlarged image:
github.com/amr080/x-protocol/blob/main/docs/contract_architecture_inheritance.mermaid

SMART CONTRACTS

USDX.sol

Public and External Functions

initialize(string memory name_, string memory symbol_, address owner): Initializes the contract.
name(): Returns the name of the token.
symbol(): Returns the symbol of the token.
decimals(): Returns the number of decimals the token uses.
convertToShares(uint256 amount): Converts an amount of tokens to shares.
convertToTokens(uint256 shares): Converts an amount of shares to tokens.
totalShares(): Returns the total amount of shares.
totalSupply(): Returns the total supply.
balanceOf(address account): Returns the account balance.
sharesOf(address account): Returns the account shares.
mint(address to, uint256 amount): Creates new tokens to the specified address.
burn(address from, uint256 amount): Destroys tokens from the specified address.
transfer(address to, uint256 amount): Transfers tokens between addresses.
blockAccounts(address[] addresses): Blocks multiple accounts at once.
unblockAccounts(address[] addresses): Unblocks multiple accounts at once.
isBlocked(address account): Checks if an account is blocked.
pause(): Pauses the contract, halting token transfers.
unpause(): Unpauses the contract, allowing token transfers.
setRewardMultiplier(uint256 _rewardMultiplier): Sets the reward multiplier.
addRewardMultiplier(uint256 _rewardMultiplierIncrement): Adds the given amount to the current reward multiplier.
approve(address spender, uint256 amount): Approves an allowance for a spender.
allowance(address owner, address spender): Returns the allowance for a spender.
transferFrom(address from, address to, uint256 amount): Moves tokens from an address to another one using the allowance mechanism.
increaseAllowance(address spender, uint256 addedValue): Increases the allowance granted to spender by the caller.
decreaseAllowance(address spender, uint256 subtractedValue): Decreases the allowance granted to spender by the caller.
DOMAIN_SEPARATOR(): Returns the EIP-712 domain separator.
nonces(address owner): Returns the nonce for the specified address.
permit(address owner, address spender, uint256 value, uint256 deadline, uint8 v, bytes32 r, bytes32 s): Implements EIP-2612 permit functionality.

Private and Internal Functions

_authorizeUpgrade(address newImplementation): Internal function to authorize an upgrade.
_mint(address to, uint256 amount): Internal function to mint tokens to the specified address.
_burn(address account, uint256 amount): Internal function to burn tokens from the specified address.
_beforeTokenTransfer(address from, address to, uint256 amount): Hook that is called before any transfer of tokens.
_afterTokenTransfer(address from, address to, uint256 amount): Hook that is called after any transfer of tokens.
_transfer(address from, address to, uint256 amount): Internal function to transfer tokens between addresses.
_blockAccount(address account): Internal function to block account.
_unblockAccount(address account): Internal function to unblock an account.
_setRewardMultiplier(uint256 _rewardMultiplier): Internal function to set the reward multiplier.
_spendAllowance(address owner, address spender, uint256 amount): Internal function to spend an allowance.

_useNonce(address owner): Increments and returns the current nonce for a given address.
_approve(address owner, address spender, uint256 amount): Internal function to approve an allowance for a spender.

Events
Transfer(from indexed addr, to uint256, amount uint256): Emitted when transferring tokens.
RewardMultiplier(uint256 indexed value): Emitted when the reward multiplier has changed.
Approval(address indexed owner, address indexed spender, uint256 value): Emitted when the allowance of a spender for an owner is set.
AccountBlocked(address indexed addr): Emitted when an address is blocked.
AccountUnblocked(address indexed addr): Emitted when an address is removed from the blocklist.
Paused(address account): Emitted when the pause is triggered by account.
Unpaused(address account): Emitted when the unpause is triggered by account.
Upgraded(address indexed implementation): Emitted when the implementation is upgraded.

Roles
DEFAULT_ADMIN_ROLE: Grants the ability to grant roles.
MINTER_ROLE: Grants the ability to mint tokens.
BURNER_ROLE: Grants the ability to burn tokens.
BLOCKLIST_ROLE: Grants the ability to manage the blocklist.
ORACLE_ROLE: Grants the ability to update the reward multiplier.
UPGRADE_ROLE: Grants the ability to upgrade the contract.
PAUSE_ROLE: Grants the ability to pause/unpause the contract.

wUSDX.sol
Public and External Functions
initialize(IUSDX _USDX, address owner): Initializes the contract.
pause(): Pauses the contract, halting token transfers.
unpause(): Unpauses the contract, allowing token transfers.
paused(): Returns true if USDX or wUSDX is paused, and false otherwise.
DOMAIN_SEPARATOR(): Returns the EIP-712 domain separator.
nonces(address owner): Returns the nonce for the specified address.
permit(address owner, address spender, uint256 value, uint256 deadline, uint8 v, bytes32 r, bytes32 s): Implements EIP-2612 permit functionality.

Private and Internal Functions
_beforeTokenTransfer(address from, address to, uint256 amount): Hook that is called before any transfer of tokens.
_authorizeUpgrade(address newImplementation): Internal function to authorize an upgrade.
_useNonce(address owner): Increments and returns the current nonce for a given address.

Events
Transfer(from indexed addr, to uint256, amount uint256): Emitted when transferring tokens.
Approval(address indexed owner, address indexed spender, uint256 value): Emitted when the allowance of a spender for an owner is set.
Paused(address account): Emitted when the pause is triggered by account.
Unpaused(address account): Emitted when the unpause is triggered by account.

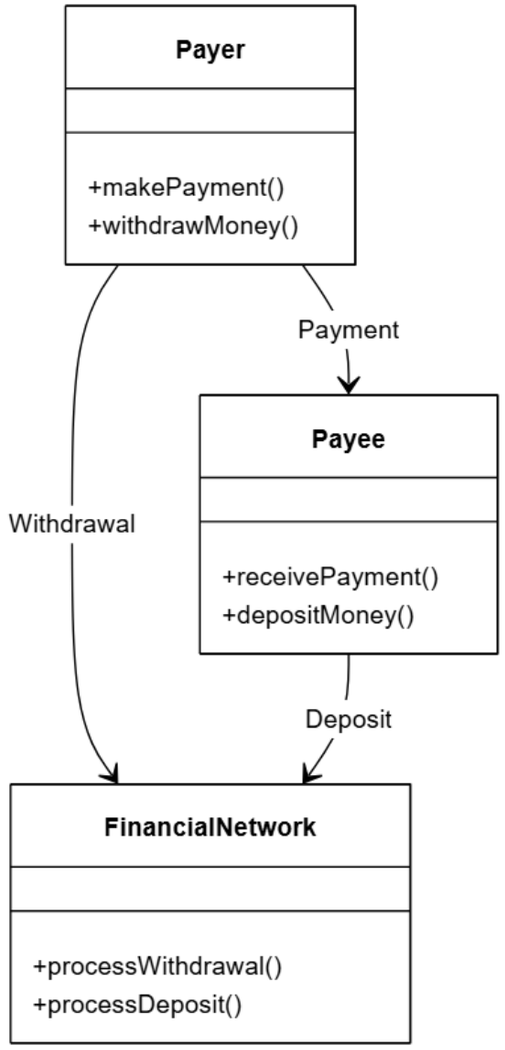
Upgraded(address indexed implementation): Emitted when the implementation is upgraded.

Roles
DEFAULT_ADMIN_ROLE: Grants the ability to grant roles.
UPGRADE_ROLE: Grants the ability to upgrade the contract.
PAUSE_ROLE: Grants the ability to pause/unpause the contract.

DEPLOYMENTS

	IMPLEMENTATION	PROXY
USDX	0x00232a7793CA38320A12893c53547543C79546b8	0x81536233C3FfaEa0198D7B5Ce8dEceDf3C520A66
wUSDX	0xfA52C849F1d5EC11D83a48281b9368856A16f1e0	0xDB45A2137EfdfBe8a1F4EaDCCda8b56990B22361

HOW DO PAYMENTS WORK?

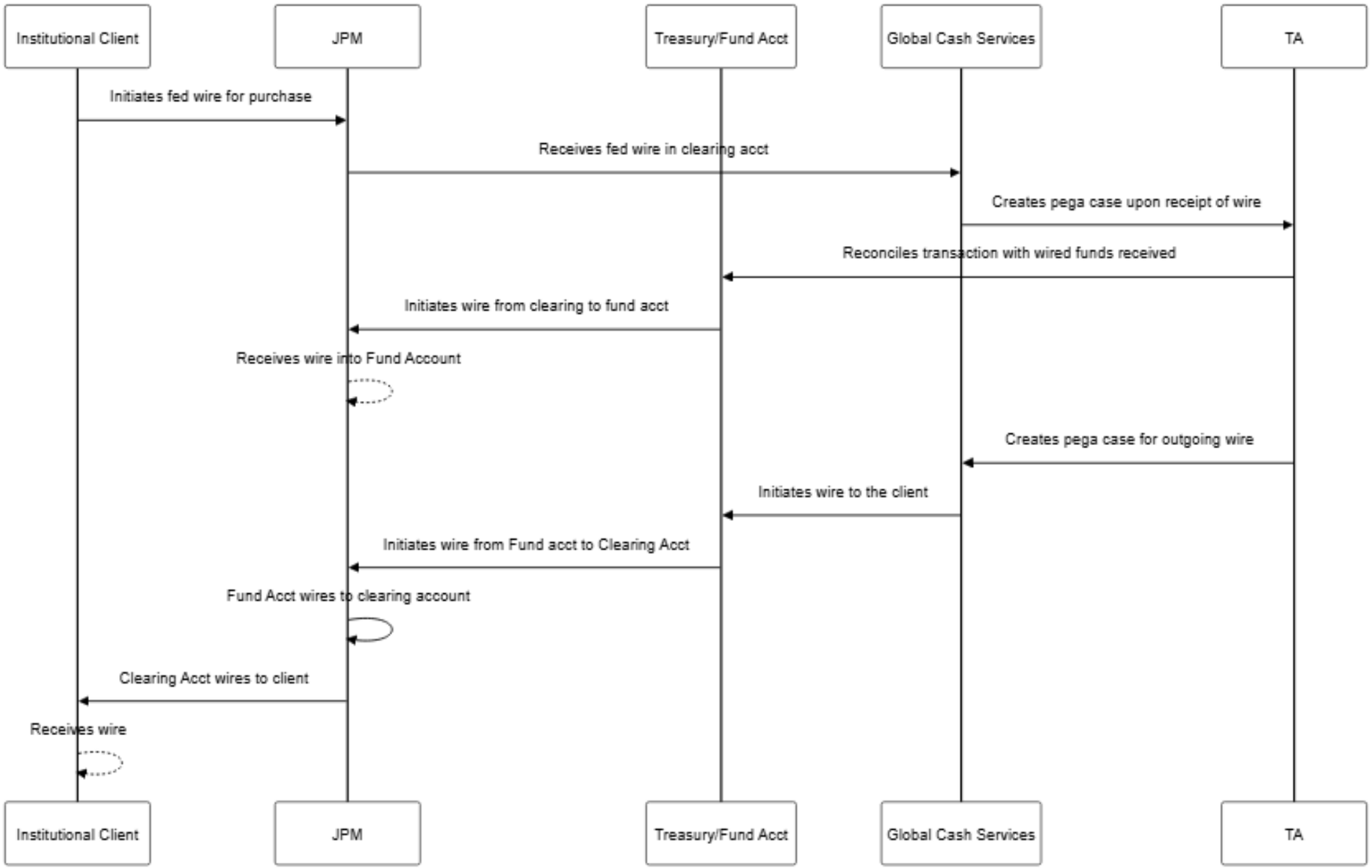


WITHDRAWAL
1 Alice sends a withdrawal request to the Bank.
2 Bank prepares an electronic coin and digitally signs it.
3 Bank sends coin to Alice and debits her account.

PAYMENT/DEPOSIT
1 Alice gives Bob the coin.
2 Bob contacts Bank5 and sends coin.
3 Bank verifies the Bank's digital signature.
4 Bank verifies that coin has not already been spent.
5 Bank consults its withdrawal records to confirm Alice's withdrawal. (optional)
6 Bank enters coin in spent-coin database.
7 Bank credits Bob's account and informs Bob.
8 Bob gives Alice the merchandise.

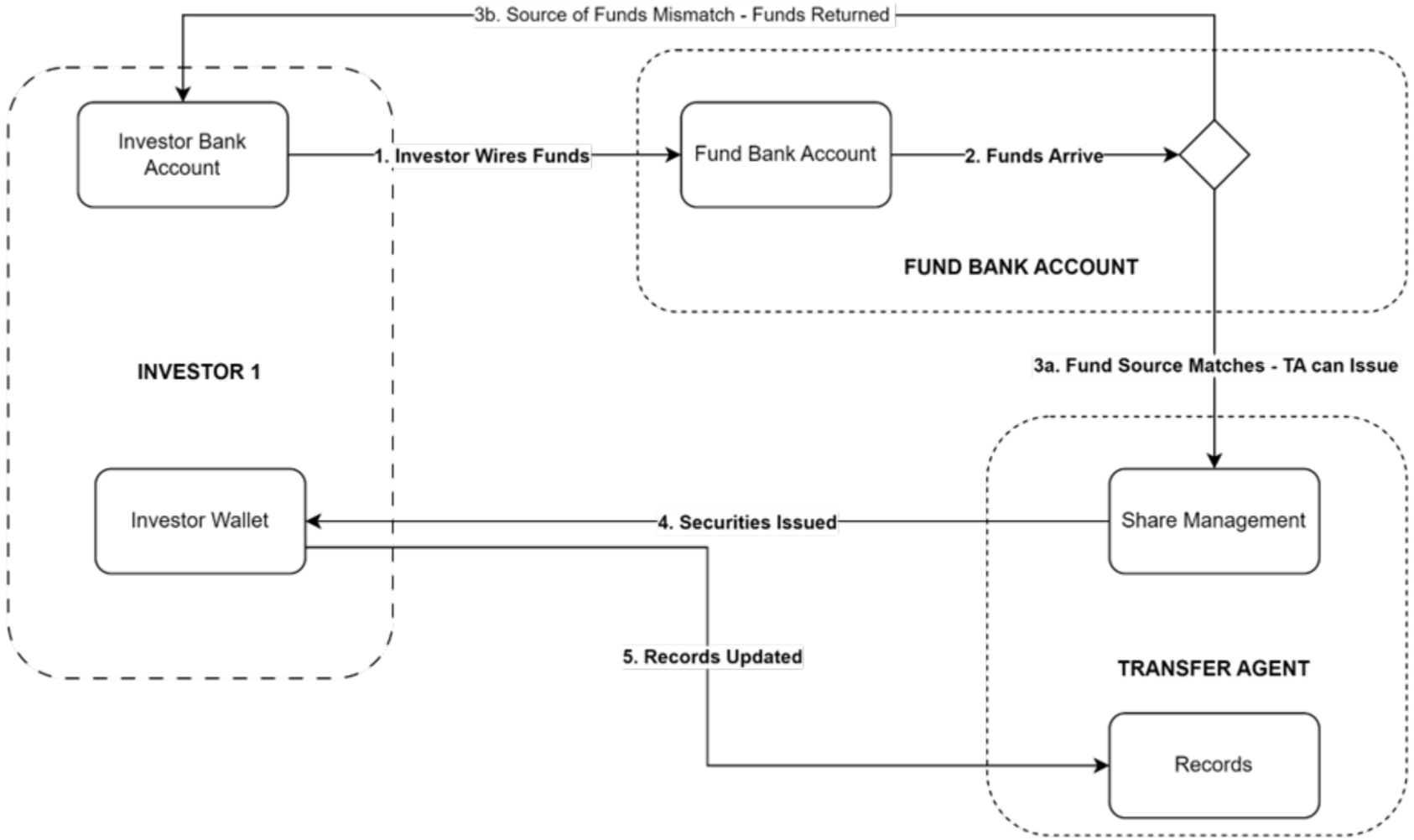
One should keep in mind that the term "Bank" refers to the financial system that issues and clears the coins. For example, the Bank might be a credit card company, or the overall banking system. In the latter case, Alice and Bob might have separate banks. If that is so, then the "deposit" procedure is a little more complicated: Bob's bank contacts Alice's bank, "cashes in" the coin, and puts the money in Bob's account.

COMPARABLES
FOBXX CASH MANAGEMENT



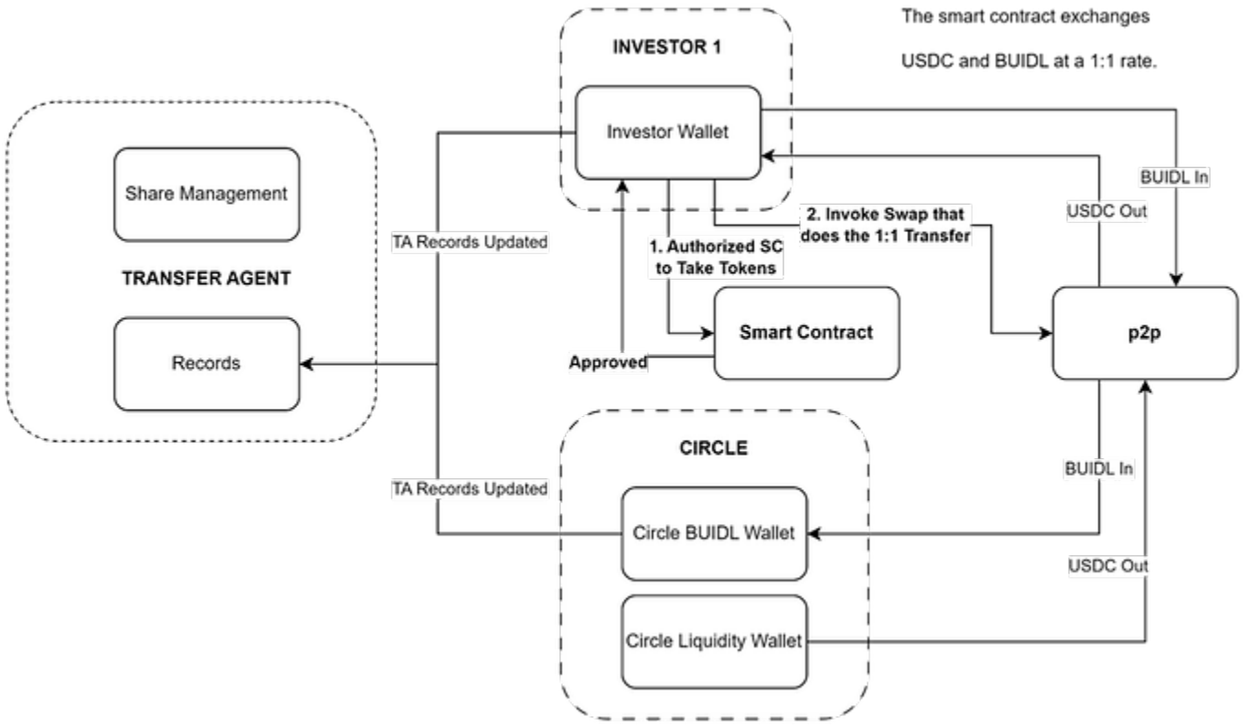
BUIDL / SECURITIZE
USD SUBSCRIPTION

For subscription into BUIDL, the investor must first complete onboarding with Securitize. This includes all KYC/B requirements for approval into the fund. Once approved, the investor will sign the BUIDL subscription agreement and will be provided with USD funding instructions via the Securitize platform UI. Upon receipt of funds, tokens will be minted to the wallet address provided during the subscription process. Funds must be received prior to the daily cutoff time (2:30 PM ET each business day) in order to be minted the same day. Newly minted tokens will have a 24-hour lock-up period.



USD REDEMPTION
Investors may redeem directly with the Fund to receive USD via wire transfer to the bank account stored on the records of the Transfer Agent. Redemptions are initiated by sending tokens to the redemption wallet, which can be facilitated through the Securitize platform UI or by directly sending tokens to the redemption wallet. Tokens must be received into the records of the Transfer Agent by the cutoff time (3:00 PM ET each business day) for same day processing of the wire payment. Redemptions received after the cutoff time will be paid during the next operating day of the fund.

SALE FOR USDC VIA SECONDARY MARKET
BUIDL shareholders may also engage directly with Circle in a secondary market transaction to transfer their BUIDL position to Circle in exchange for USDC, in which a smart contract that facilitates a 1:1 atomic swap of BUIDL for USDC. This secondary market is available 24/7/365, but is at the discretion of Circle and is not guaranteed… Transactions can be facilitated through the Securitize platform UI, or by directly interacting with smart contracts. The smart contract facilitates the exchange by sending USDC to the sending address of the BUIDL.



[INTENTIONALLY BLANK]

[INTENTIONALLY BLANK]

[INTENTIONALLY BLANK]

[INTENTIONALLY BLANK]