Set: A Protocol for Baskets of Tokenized Assets

Felix Feng, Brian Weickmann

v1.2

April 8, 2019

Abstract

We present a protocol ("Set") that facilitates the creation, issuance, redemption, and rebalancing of fungible, collateralized baskets of tokenized assets ("Set tokens" or "Sets"). The protocol allows utilization of liquidity from decentralized exchanges for Set issuance and redemption. Users who desire to track an index or have their portfolios automatically updated based on a trading strategy can subscribe to a Rebalancing Set. The protocol is open, non-rent seeking, and aims to utilize decentralized governance.

Contents

1 Introduction

1.1 The Tokenization of the World

With the creation of the Bitcoin blockchain and subsequently the Ethereum blockchain, anyone can create their own digitized token to represent securities, goods, services, real world assets, etc. As of April 2019, there are 2,100 tokens listed on CoinMarketCap that represent over]$160 billion in value, not to mention a long-tail of obscure tokens not tracked by CoinMarketCap [1]. Tokens are being created for numerous purposes, including but not limited to:

- **Currency on Blockchains / DAGs**: Blockchains (e.g. Bitcoin, Ethereum, NEO, Zcash), directed acyclic graphs (e.g. IOTA, HashGraph), and their forks (e.g. BitcoinCash, Ethereum Classic) have their own native tokens that incentivize parties to process transactions.
- **Second Layer Protocols**: Protocols such as 0x Project [2], Kyber Network, and Raiden are issuing utility tokens that are used to transact on their networks and support decentralized governance.
- **Digital Goods**: Gaming communities, streaming services, content platforms, and gambling sites are using tokens to represent virtual goods, incentivize direct payment of content creators, and create economic systems that bypass traditional intermediaries.
- **Decentralized Financial Products**: Projects such as dY/dX [3] and Dharma [4] are allowing anybody to issue options, short sell, and create loans that are represented by tokens. MakerDAO and Fragments are creating "stablecoins" that aim to resist fluctuations in value. Projects such as Augur and Gnosis allow anyone to interact with predictions markets.
- **Real World Assets**: Projects such as TrustToken and DigixDAO are bringing the tokenization of real-world assets.
- **Securities**: Blockchain Capital, a cryptocurrency hedge fund, has issued tokenized securities to represent stakes in their fund. Harbor and Polymath are building platforms for parties to issue tokenized securities and investors to compliantly trade securities on secondary markets. Numerai has taken a different approach, awarding data scientists with profit-share tokens for their work in generating profits for the hedge fund.

We think that, as more people and organizations turn to tokens for economic coordination in the coming decades, millions or billions of different tokens could exist.

1.2 The Pains of Increasing Token Cardinality

There is an increasing complexity in dealing with an ever-growing number of tokens. We attribute this to the lack of a standardized protocol for multi-token use cases. Some of these pains are detailed below:

- **Transaction Costs**: Existing transaction infrastructure was not designed with multi-token use cases in mind. External accounts (standard addresses in Ethereum) need to construct, sign, broadcast, and pay a transaction fee for each token transfer. The transaction fees and effort of dealing with a multitude of tokens grow linearly with the number of tokens transacted. If one wants to send 100 different tokens using a standard Ethereum external account, one would need to sign and broadcast 100 transactions and thus pay 100 transaction fees. This limits the scalability of decentralized apps and results in a subpar user experience.
- **Cognitive Overload**: As the number of tokens that users need to interact with grows, the cognitive load required for management of tokens follows suit. For each token they own, users must understand each token's qualities (e.g. price, monetary policy, utility, liquidity, interface, etc). Each token needs to be treated individually, and users must retain everything in their head at once.

- **Client Limitations**: Ethereum clients such as Parity, Mist, and Metamask are limited in their ability to manage groups of tokens. Clients don't have features that allow for the grouping together of tokens or batch transfers. Thus, users have no option but to manually and repeatedly perform common operations like viewing, transferring, and function calls across multiple token smart contracts.
- **Investor Pains**: Investors who hold a multitude of tokens lack the tools to actively acquire, manage, and reassess their investments. Standard active portfolio management practices such as rebalancing, bucketing, and transfers become onerous and repetitive processes. Analyzing groups of tokens requires individually aggregating the prices of tokens and bucketing tokens must be done manually. These inconveniences even have financial repercussions when one considers how fees accumulate as a result of token-by-token exchange trades and withdrawals.
- **Developer Pains**: Developers lack a standard for hiding the details of multiple tokens from users on the protocol level. Currently, information hiding must be done on the application-layer of the stack vs. natively in the protocol layers. Developers also need to understand all the tokens that their system deals with. Finally, it is difficult to create a good user experience to onboard new users if the users are inundated by the details of tokens.

1.3 Abstraction as a Solution

In software engineering and computer science, abstraction allows programmers to think on a certain level of complexity while hiding away details not relevant to the problem at hand. We use abstractions to prevent overloading the end user with details when they care more about higher level concepts. This lets developers focus user attention on what objects represent within their platforms, rather than the nuts and bolts.

In traditional finance, we have abstractions like the American stock market indices (e.g. SP 500, Dow Jones Industrial Average (DJIA)) that represent hundreds or thousands of individual stocks. In the insurance industry, we purchase policies comprised of a set of services in exchange for paying an insurance premium, without the need for fretting over individual coverage scenarios.

For cryptocurrencies, we can envision an abstract token or meta token, a single token representing a basket or portfolio of its underlying tokens.

2 Set Token

2.1 Overview

We introduce a Set token ("Set"), an abstract, fungible token fully collateralized by its underlying tokens. While Sets will eventually interoperate across blockchains, Sets are described in this document as smart contracts on the Ethereum blockchain that adhere to the ERC20 token standard.

2.2 Properties

- **ERC20 token**: Sets comply with ERC20 [5], a standard interface of functions and events that an Ethereum token contract must implement. The ERC20 standard includes functions such as transfer, approve, totalSupply, and balanceOf. This means that Sets can be transferred, traded, approved to transfer: just like any other ERC20 token.
- **Collateralized**: Sets are collateralized by their underlying tokens (components). This means that the Set trustlessly has custody of the components and can only be accessed through its exposed methods. Sets are created by transferring the underlying tokens to Set's Vault and issuing a new Set token representing those tokens.
- **Redeemable**: Sets can be redeemed or traded for their components. As Sets are backed by the underlying tokens, users can be certain that their Set can be traded for their components.

- **Specified tokens and units**: Sets are specified by a list of underlying tokens and their respective quantities. Each token minted from a Set contract cannot deviate from the specified tokens and ratios as defined during the construction of the Set contract. As long as the desired tokens issued or redeemed matches the predefined Set weights, it is possible to issue and trade fractions of Sets.
- **Composable**: As long as the Sets conform to the ERC20 standard, Sets can be composed of other Sets. This makes it possible to have a single token to represent a limitless number of other tokens without hitting the block gas limit.
- **Trustless**: Set is open source and functions only as programmed. Set has been designed so that no owners or administrators can cause changes to the held collateral tokens.

2.3 Benefits

- **Save gas**: Without Sets, acquiring and transferring multiple ERC20 tokens requires paying transaction fees on transfers of each token. By transacting using Sets, users only need to pay transaction fees on a single transaction for the tokens they represent.
- **Focus on higher-level concepts**: Sets allow users to think about higher-level concepts. Since Sets are composable, it is possible to build up multiple layers of abstraction. This makes Sets a very generalizable and a powerful tool for building and reasoning about complex decentralized applications.
- **Underlying value**: The intrinsic value of a Set can be determined by summing the value of its underlying components. We expect that Sets will be priced at a premium on token exchanges relative to the cost of their underlying tokens, reflecting the cost of minting the Sets.
- **No Counterparty Risk**: Traditional "higher-level assets" such as ETFs often trade at a discount, because these assets are not without counterparty risk. Because a trustless, autonomous smart contract holds custody to the underlying tokens, there is no third party that can fail to live up to its contractual agreements.

3 Use Cases

3.1 Index Token/ETF

In the traditional financial industry, market indexes are collections of assets that track the overall movement of a specific theme or sector in the equity markets. They represent the aggregate value of their underlying securities. Investors usually use indices to track the value of the market over time, gauge the market's financial health, and benchmark their own returns. Some of the most widely cited market indexes include the SP 500, which tracks the 500 largest companies having common stock on the largest US stock exchanges, and the DJIA, which tracks 30 large publicly owned companies in the US. For investors, index fund investments are a passive form of fund management and are considered ideal core portfolio holdings for many people's accounts. They usually have lower management expense ratios, allow for diversification, and lower trading costs for investors. In the cryptocurrency world, a Set is the natural tool for the representation of a collection of publicly traded tokens. As a market index that tracks tokens, Sets can be used for investment, tracking, market-making, and advanced derivatives:

- **Investment**: As investment vehicles, Sets offer a number of advantages, such as cost reductions, asset diversity, and lower maintenance requirements.

  Without Sets, acquiring a basket of tokens can be prohibitively expensive in terms of fees and energy. Currently, investors who want exposure to a market index need to purchase the individual underlying tokens themselves. The investor would need to 1) register and provide personal identification information for numerous exchanges, 2) pay maker-taker trading fees for each token purchase, and 3) pay network transaction fees to withdraw/transfer each token into their personal wallet. With Sets, the investor could purchase a single representative token from a single exchange, greatly reducing trading fees and saving time.

Owning Sets allows investors to diversify and manage their risk across any particular asset class. Sets makes it easy for retail and institutional investors who do not know which individual crypto assets to purchase to get exposure to the entire market or one of its sub sectors. For example, an investor wanting exposure to the top 10 tokens by market capitalization for long-term investment can invest by purchasing a single Top10 Set composed of the top 10 ERC20 tokens. For a desired Set that does not exist, the investor can invent / create their own using the protocol.

- **Tracking**: Sets are well suited to be a measurement of value for the token market. Sets can be used by investors and analysts to describe the market and compare returns on investments. Our current tools for tracking are quite limited. CoinMarketCap only provides an aggregate market capitalization of cryptocurrencies. Current tools are also unstandardized. Sets can represent a construct that the industry relies on for evaluating the health of cryptocurrencies and performing macro analyses.
- **Rebalancing**: Because Sets are passively managed and not meant to be actively traded, Sets are low-maintenance, only requiring infrequent rebalances. As the price of the underlying assets fluctuate, traditional market indices are rebalanced or updated by adjusting the collection of assets that are included in the index. This is a practice that is usually done quarterly, semi-annually, or annually. Set investors can easily rebalance their portfolios by trading for an updated Set that reflect the current collection of tracked tokens or issuing a Rebalancing Set. Rebalancing is discussed in detail in a subsequent section.
- **Market-Making**: Under the hood, Sets are most similar to exchange-traded-funds (ETFs). An ETF is a type of fund that owns the underlying assets and divides ownership into shares. Supply of ETF shares is regulated through creation (trading of underlying assets for ETF shares) and redemption (trading of ETF shares for the underlying assets). Creation and redemption involve very specialized investors known as authorized participants (AP). The AP assembles the required portfolio of assets and turns that basket into the fund for newly created ETF shares. Set issuance and redeem mechanics are synonymous with the ETF's creation and redemption, and thus there will be the equivalent of APs in this system that will be issuing and redeeming Sets.

  In the token world, traders can serve as liquidity-providers who are incentivized to normalize Set price deviation, preventing large deltas between the Set price and the Set's aggregate component value. Set's permissionless nature allow traders to take advantage of these risk-free arbitrage opportunities by issuing and redeeming Sets or by trading for other Sets.
- **Advanced Derivatives**: It is possible to create sophisticated financial products such as basket default swaps, collateralized debt obligations, or basket of futures by combining Sets with primitives such as derivatives and loans (e.g. dY/dX [3] and Dharma [4], respectively). These instruments allow investors to manage risk and better facilitate token price discovery.

3.2 Multi-Token Decentralized Applications

As more second-layer protocols go live, decentralized applications ("dApps") will be allowing complex operations that incorporate multiple protocols. Usage of these protocols usually requires protocol tokens, and users of these dApps would need to purchase tokens for each protocol operation. Thus, users may need to hold dozens of tokens for each of the various protocol operations. Sets are well suited to represent units of composite-work in multi-protocol applications.

For example, let's say a developer wants programmatically to process a file in a MapReduce fashion on computers all over the world using Golem and then store it in a decentralized manner using Storj. Using a decentralized exchange protocol such as 0x protocol, the developer can programmatically purchase a single ComputeStore Token which is composed of 100x Golem and 100x Storj tokens using wrapped Ether, ether that has been converted into the ERC20 specification. Then the developer can call the redeem function in the ComputeStore Token contract to get the underlying Golem and Storj tokens for usage on the Golem and Storj networks. Any unused tokens can be traded, sold, or issued into a new Set.

3.3 Decentralized Finance

There are numerous emergent, crypto-native use cases for Sets that have yet to be fully explored. Some of these use cases include:

- **Composite StableCoin**: A Set composed of other stable coins to smooth out volatility and diversify risk of de-pegging with even distribution across various approaches.
- **Protocol Collateral**: Complex Sets that are used for collateralization in second layer protocols (e.g. basket of goods for use as collateral in the MakerDao CDP System).
- **Hedged Bet**: A Set containing both a long position and a short position.
- **Fractional Ownership of Non-Fungible Assets**: A Set representing fractional ownership of a collection of CryptoKitties or other pieces of digital art.

4 Architecture

4.1 Overview

Set defines procedures for creating, issuing, rebalancing, and redeeming Set tokens using a collection of smart contracts and integrations with liquidity pools. Set's architecture has been inspired by 0x Protocol and dYdX. Set was designed to be a collection of modular smart contracts where additional components (e.g. modules, factories) can be seamlessly integrated or removed.

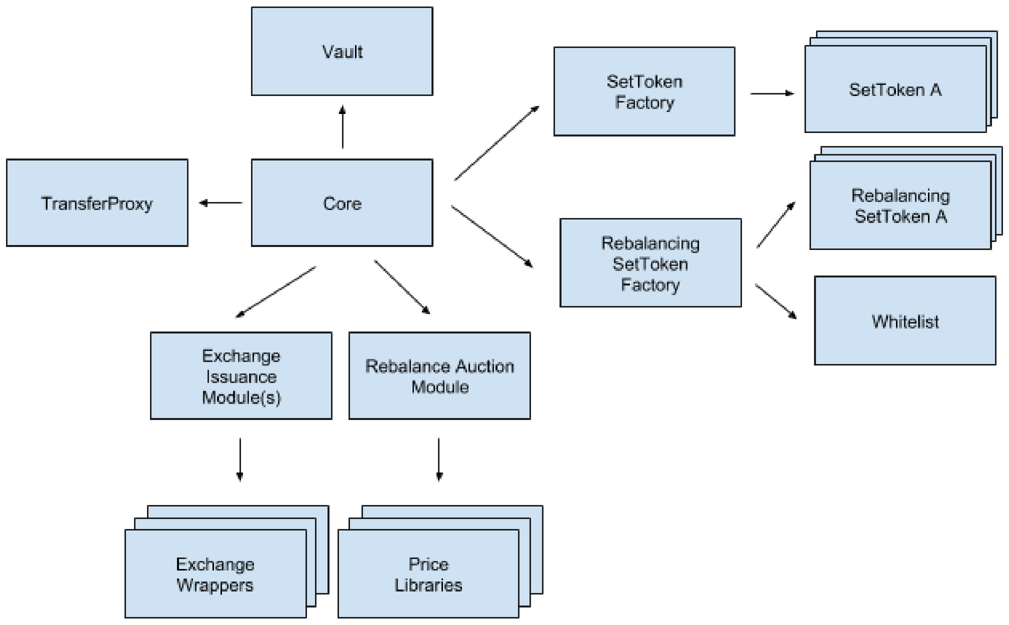

Figure 1: Contract Architecture Diagram

4.2 Smart Contracts

Set consists of a system of smart contracts [6] deployed on the Ethereum network. These include:

- **Core**: This public-facing smart contract contains all the business logic and external functions associated with creating, issuing, and redeeming Sets. Core serves as a kernel that orchestrates interactions between the various other smart contracts. In addition, Core exposes privileged functions that are available to Modules (defined below), which allow the extension of the System's functionality.

- **TransferProxy**: This smart contract facilitates the transfer of ERC20 tokens for all transactions. Users who wish to utilize their tokens must authorize the TransferProxy contract to move their tokens by calling the ERC20-compliant token's approve function. The design of a centralized TransferProxy contract is intended to minimize the number of overall number of ERC20 approve transactions required in the system.

- **Vault**: This smart contract stores all the Set's component and contains mappings to track ownership of assets. Vault's accounting interfaces are only available to Core and indirectly to Modules through Core.

- **Factory(s)**: This smart contract contains the template for a specific type of Set and facilitates the creation of all Set token contracts. Creators can create a new Set token by calling Core's createSet function and supplying the Factory address and the required parameters. For a Factory to be valid, it needs to be registered with Core. Initially, there will be a SetTokenFactory (creating standard Sets) and RebalancingSetTokenFactory (a factory for creating Rebalancing Set tokens).

- **Set Token(s)**: This ERC20 compliant smart contract represents a unique Set with specified ERC20 components and units and tracks balances of the Set tokens. Each Set token smart contract is created by a Factory and is tracked by Core. Each smart contract exposes mint and burn functions that only Core could call during issuance and redemption.

- **Rebalancing Set Token(s)**: In addition to sharing the interface and functionality of a Set Token, this smart contract stores rebalancing functionality-related state and contains all the business logic related to how the Rebalancing Set is rebalanced.

- **Module(s)**: This smart contract serves to extend the functionality of Core without requiring a complete redeployment or upgrade of Core. Modules are tracked contracts that have access to Core's exposed module functions (e.g. batchDepositModule, redeemModule). Some modules appended to the system include: ExchangeIssueModule, RebalanceAuctionModule, and RebalancingSetExchangeIssuanceModule.

- **ExchangeWrapper(s)**: This smart contract serves as a conduit between decentralized exchanges and Core to facilitate component-acquisition during issuances that require usage of outside liquidity. Each exchange wrapper contract exposes a common exchange function that accepts exchange metadata and a bytes string that encodes exchange messages. The ExchangeWrapper is responsible for parsing call data conforming to an exchange's interface ("exchange messages") and executing each order as the taker of each transaction. Initially, we will be supporting tokens from 0x Exchange V2 and Kyber Network.

- **Price Library(s)**: This library contains the logic around how the current price is calculated during a Rebalancing Set's dutch auction process and rebalance proposal validity. More about rebalancing is detailed in a further section.

- **WhiteList**: This smart contract contains a list of components deemed suitable for inclusion as a base Set component in rebalancing Sets. This contract is used during the proposal phase of Rebalancing to ensure only vetted components are rebalanced into.

4.3 Participants

There are a number of participants in the network: Creators, Users, and Traders:

- **Creators** utilize their domain knowledge, creativity, and intuition to design, create, and deploy static and rebalancing Sets that have market appeal.

- **Users** can gather components from various centralized and decentralized liquidity sources to issue Sets for trading, holding, and usage. They can also redeem their Sets for the components.
- **Traders** are economically-driven participants who compete to profit by bidding during rebalance auctions as well as facilitating price discovery by issuing and redeeming undervalued / overvalued Sets in the market.

5 ERC20 Standard Considerations

5.1 Approvals Race Condition

The ERC20 standard has a known race condition issue that could lead to unintended movement of theft of tokens. The traditional method for allowing a smart contract to interact with a user's tokens involves calling the approved method (granting the contract access to user funds), and the transferFrom method (spending the value that has previously been approved).

This scheme is susceptible to a race condition when the user calls approve a second time on a spender that has already been granted an allowance. If the spender sees the transaction containing the function call in the mempool before it has been mined, the spender can call transferFrom to transfer the previous value and still receive the authorization to transfer the new value - causing potential unintended consequences.

When users are approving tokens to the TransferProxy, it is recommended that users utilize the increaseApproval and decreaseApproval non-ERC20 functions on the token versus the traditional approve function.

5.2 Token Pausability

Many Ethereum ERC20 tokens implement "Pausable" functionality, which allows the creator of the token to prevent approvals, transfers or transferFrom functions from executing properly. Some tokens that implement Pausable functionality include Status and Augur. Since Set's redeem operations requires that every single transfer to complete properly, a single token transfer failure could hold a Set's remaining components hostage.

Set circumvents this risk by separating redemption (burning the Set and attributing components to the user in the vault) and withdrawal (transferring the tokens from the Vault to the user) into two separate steps. With this design, users will invariably be able to retrieve a Set's remaining unpaused tokens from the Vault.

5.3 Variant ERC20 Decimals

Although ERC20 tokens have standardized around 18 decimal places, many teams have decided to implement their own choice of decimal places. Some of these include Airswap (4 decimals), Zilliqa (12 decimals), and RChain (8 decimals). Decimal place variance makes it difficult to create Set Tokens that target a specific price point because tokens with lower decimal values have significantly more valuable base units. As an example, if a user wanted to issue a full Set Token (18 decimal places) containing Airswap, at a minimum, they would need to provide 10^{14} Airswap tokens (Airswap only plans to have 5×10^9 tokens ever in circulation).

5.3.1 Natural Unit

To solve this issue, Set introduces the concept of natural unit, which is the atomic, least divisible unit of a Set that can be issued or redeemed. Set creation enforces that the natural unit is larger than the transferable unit of the token with the smallest decimals place. This is represented by:

$$MinimumNaturalUnit = 10^{18-min(componentDecimals)} \tag{1}$$

The ideal natural unit that you use in Set creation is based on a number of parameters including:

- **Desired Price**: The price of a single unit (1×10^{18} base units) of a Set
- **Component Token Decimals**: The divisibility of the ERC20 component denoted by the decimals field. If an ERC20 token doesn't implement the decimal parameter, it is assumed to be 18.
- **Component Token Price**: The price of a single unit of a component
- **Set Composition**: The desired portfolio allocation of the components in the Set

Example: Alice wants to create a Decentralized Exchange Token (DEX) that contains Airswap (AST) and 0x (ZRX) tokens in a 50/50 allocation priced at \$10. AST is trading at \$0.10 and ZRX is trading at \$1. In order to achieve a 50/50 split of AST and ZRX, each Set needs 50 AST tokens (5 105 base units of AST) and 5 ZRX tokens (5 1018 base units of ZRX).

Alice faces problems when she issues partial quantities of DEX Set Tokens, which have 18 decimals. The formula for calculating the amount of component token, i, required for an issuance is as follows:

$$amountRequired_i = \left(\frac{quantity_{issue}}{quantity_{Set}}\right) \times componentUnits_i \tag{2}$$

Where $quantity_{Set}$ is the base unit amount of 1 Set Token (10^{18}). If a user wants to issue 10^{12} base units of the DEX Set Token, the ZRX amounts work as expected:

$$amountRequired_{zrx} = \left(\frac{10^{12}}{10^{18}}\right) \times (5 \times 10^{18}) = 5 \times 10^{12} \text{ base units of ZRX} \tag{3}$$

However, the same math with AST leads to an untenable number:

$$amountRequired_{ast} = \left(\frac{10^{12}}{10^{18}}\right) \times (5 \times 10^{5}) = 0.5 \text{ base units of AST} \tag{4}$$

Since base units are not divisible and all decimals are rounded down, in order to issue this Set the user would "transfer" 0 base units of AST to collateralize the Set Token. However, the system would still credit the user with the collateral for the Set Token, in essence leaving the system not fully collateralized. Natural units help fix this attack vector by requiring all issue amounts to be greater than the natural unit. Adding the natural unit changes Equation 2 to the following:

$$amountRequired_i = \begin{cases} \text{revert()} & quantity_{issue} < naturalUnit_{Set} \\ \left(\frac{quantity_{issue}}{naturalUnit_{Set}}\right) \times componentUnits_i & quantity_{issue} \geq naturalUnit_{Set} \end{cases} \tag{5}$$

The natural unit can be calculated using Equation 1, which returns 10^{14}. Additionally, the components units must be updated to target the correct amount of base units per Set Token (5×10^{18} ZRX and 5×10^{5} AST):

$$5 \times 10^{18} = \left(\frac{10^{18}}{10^{14}}\right) \times componentUnits_{zrx} \tag{6}$$

$$componentUnits_{zrx} = 5 \times 10^{14} \tag{7}$$

$$5 \times 10^{5} = \left(\frac{10^{18}}{10^{14}}\right) \times componentUnits_{ast} \tag{8}$$

$$componentUnits_{ast} = 50 \tag{9}$$

Thus, for every 10^{14} base units of the DEX Set Token issued, the user must contribute 5×10^{14} ZRX base units and 50 AST base units. With this configuration, the target specifications are achieved, one DEX Set Token equals \$10 and has a 50/50 allocation split.

However, this equation is not complete, imagine the scenario where a user attempts to issue 1.05×10^{14} base units of the DEX Set. Again, the ZRX amount is valid, however the AST amount leaves the ability for a user to not fully collateralize their issuance:

$$amountRequired_{zrx} = \Big(\frac{1.05 \times 10^{14}}{10^{14}}\Big) \times (5 \times 10^{14}) = 5.25 \times 10^{14} \text{ base units of ZRX} \tag{10}$$

$$amountRequired_{ast} = \Big(\frac{1.05 \times 10^{14}}{10^{14}}\Big) \times 50 = 52.5 \text{ base units of AST} \tag{11}$$

A further restriction must be put on the issuance amount, in this case the issuance amount must also be a multiple of the natural unit. This guarantees that the component unit amount in Equation 5 will always be multiplied by a whole number, thus guaranteeing the required amounts are always whole numbers. The final equation for calculating required units for component token i, using the natural unit, is as follows:

$$amountRequired_i = \begin{cases} \text{revert()} & quantity_{issue} \mod naturalUnit_{Set} \neq 0 \\ \Big(\frac{quantity_{issue}}{naturalUnit_{Set}}\Big) \times componentUnits_i & quantity_{issue} \mod naturalUnit_{Set} = 0 \end{cases} \tag{12}$$

5.4 Variant Transfer Receive Quantities

Although many ERC20 tokens implement standard transfer and transferFrom functions, certain ERC20 tokens contain variant implementations of transfer and transferFrom functions where there is an asymmetry between the amounts of tokens sent from an address and amount received. This is common in tokens that include fees in their transfer and transferFrom functions. Transferring these tokens into the Vault could result in transactions where fewer tokens are received by the Vault than expected.

To ensure full collateralization, Set enforces checks that the amount received and transferred is the amount the transfer specifies. This comes with a conscious trade-off that tokens with non-zero transfer fees will not be supported by the protocol.

5.5 Variant Transfer Return Values

The ERC20 specification outlines certain success return values for transfer and approval functions. However, some projects have decided to return different values or no value at all, which causes checks to make sure a valid transfer occurred to fail. In order to handle this, Set implements an ERC20 Wrapper that allows for null or true return values to evaluate to true for this specific subset of functions, instead of reverting on a null response. This still means that tokens returning a non-boolean/null value will revert and not be usable in the Set system.

6 Specification

6.1 Overview

Core is the smart contract that contains the business logic for everything in a Set's lifecycle including creating, issuing, and redeeming. Initially, a Set is designed and created through the Core contract and its specified Factory. Sets are then funded or issued through the standard Issuance Flow (if the user has all the underlying components). Users who wish to have their Sets updated can choose to do so by issuing a Rebalancing Set Token. Finally, a Set can be redeemed by burning the owner's Set token and retrieving the components.

6.2 Create

Creation is the process of instantiating a new ERC20-compliant Set token contract. These contracts are instantiated by Factories, which serve as the blueprint for a Set token contract to be constructed. Anybody can create a Set by calling the Core's createSet function by passing in the required parameters below:

Property	Type	Description
factoryAddress	address	The address of the factory to create from
components	address[]	The addresses of the component tokens
units	uint256[]	The amounts of each component token (per naturalUnit)
naturalUnit	uint256	The minimum unit to be issued or redeemed
name	bytes32	The bytes32 encoded name of the Set
symbol	bytes32	The bytes32 encoded symbol of the Set
callData	bytes	If required, additional data in bytestring format

There are no restrictions to how many different ERC20 tokens can be included, aside from block gas limits, and data input limits. Since Set tokens are ERC20 tokens, Sets can be composed of other Sets. Deploying a Set token creates an ERC20 token contract with 0 initial supply.

Initially, there will be two Factories that create the following:

- **Standard Sets**: Vanilla Sets whose composition and units have been pre-defined.
- **Rebalancing Sets**: Sets whose composition can only be one other Set and is programatically updated based on predefined rebalancing mechanics and inputs from a data source. We discuss Rebalancing in further detail in a later section.

In the future, new Factories can be introduced that create Sets with additional functionality and properties such as:

- **Regulatory Compliant Sets**: Sets with securities regulations, KYC/AML policies, and tax law compliance baked in, adhering to open standards such as R-Token or PolyMath.
- **Sets with Fees**: Sets where certain function calls impose small fees that accrue to the Creators or other specified parties, rewarding them for devising desirable compositions.
- **Sets of Non-fungibles**: Sets where tokens represent fractional ownership of collections of non-fungible tokenized assets that adhere to the ERC721 standard (e.g. art, title deeds, and sports teams)
- **Issue-Only**: Sets where the redemption feature has been disabled. Use cases may be securities of two companies that have participated in a corporate merger or acquisition.
- **Stake-able Sets**: Sets where the components can be contributed to performing work on staking protocols (e.g. Livepeer) or governance (e.g. Token Curated Registries).
- **Interest-Generating Sets**: Sets where components can be loaned out via flash lending protocols or longer-term loans (e.g. Compound) to generate interest on idle components.

The creation flow is formalized as follows:

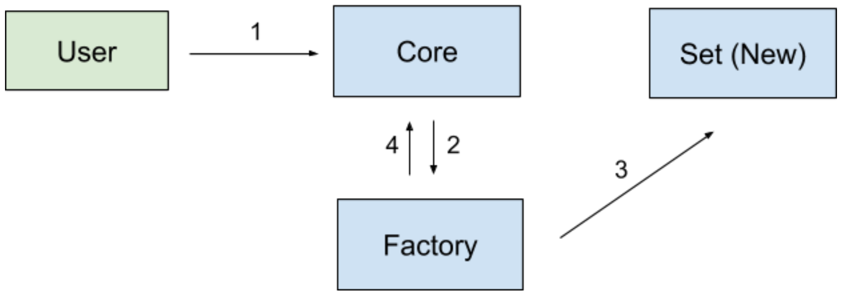

Figure 2: Creation Flow

1. User calls the createSet function on Core, providing as arguments the Factory address, component addresses, component units, natural unit, name, symbol, and other data (optional).
2. Core checks that the Factory address is authorized. If so, Core passes the argument data to the Factory.
3. The Factory deploys a new Set token smart contract with the provided parameters.
4. The Factory returns to Core the new Set address, which Core records in its mapping of tracked Sets.

6.3 Issue

Issuance is the process of minting new tokens for a particular Set, increasing the Set supply. With a valid deployed Set, anybody can call the Core's issue function to convert a specified mix of ERC20 tokens into a Set token. For a standard issuance, users need to contain all components of a Set in a single public address and/or the Vault.

Issuance is a two-step process of depositing funds into the Vault and then issuing (attributing components to the Set and minting a Set token to the user). Ahead of any deposits, users need to approve their tokens to the TransferProxy. Users can incrementally deposit tokens into their vault before issuing, as issuance uses a combination of the user's tokens in their wallet as well as their tokens in the Vault.

The issuance flow is formalized as follows:

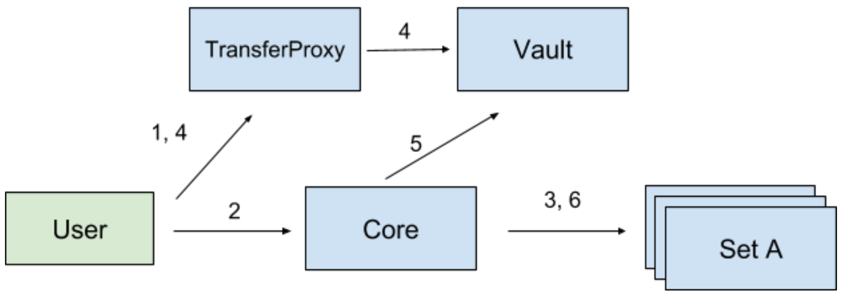

Figure 3: Issuance Flow

1. User approves components for transfer to the TransferProxy.
2. User calls the issue function on Core, targeting Set A and specifying an issue quantity. Core validates that the Set A is a known/tracked Set. The issue amount must be a multiple of the Set's natural unit.
3. Core reads the targeted Set's components, units, and naturalUnit.
4. For each component, Core calls the TransferProxy contract's transfer function and moves the funds from the user into the Vault.

5. The Core then increments the balance in the Vault for each of the transferred components to the Set.
6. When all the transfers are complete, Core calls Set A's mint function (callable only by the Core), which mints the corresponding number of Sets the user has issued.

6.4 Redemption

Redemption is the process of converting a Set into its underlying component tokens. Redeeming tokens reduces the token supply of Set tokens in the contract. It involves burning a Set token and attributing the components to the end user. Users who wish to redeem a token can call Core's redeem or redeemAndWithdraw function, specifying the Set address and the quantity to redeem

The redemption flow is formalized as follows:

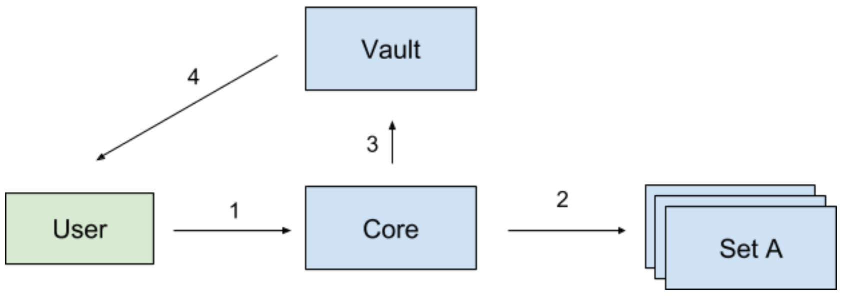

Figure 4: Redeem Flow

1. User calls the redeem function on Core, targeting Set A and specifying a redemption quantity. Core validates that the Set A is a known/approved Set.
2. Core queries Set A for the user's balance and checks the user has sufficient Set A tokens to redeem. If the user has sufficient balance, it queries the Set for its components, units, and naturalUnits. To prevent re-entrancy attacks, Core call Set A's burn function, decrementing the user's balance.
3. For each component, Core calls Vault to decrements the specified units attributed to the Set A. In each decrement, the Vault decrements the corresponding Set A's ownership of a token and increments the User's.
4. The user can subsequently call Core's withdraw function to retrieve the tokens from the Vault.

6.5 Deposit and Withdraw

Depositing is the process of moving components from a user's wallet into the Vault, and withdrawing is the process of moving components from the Vault to the user's wallet. Set Protocol exposes single as well as batch versions of these functions.

6.6 Modules

Modules are smart contracts that augment the Core's capabilities of creating, issuing, and redeeming Sets. To allow efficient facilitation of components and minimize state modifications, Modules are granted privileged capabilities such as transferring tokens using the Transfer Proxy and incrementing/decrementing tokens from the Vault. Modules can be appended and removed through protocol governance.

Initially, the following modules will be appended to the protocol:

- **ExchangeIssuanceModule**: This module helps facilitate the issuance and redemption of Sets using liquidity from decentralized exchanges in a single transaction.

- **RebalanceAuctionModule**: This module serves as an interface for bidding during a Rebalancing Set's dutch auction and for withdrawing components if a Rebalancing Set is put into a drawdown state. More about this module is detailed in the Rebalancing section.
- **RebalancingSetIssuanceModule**: This module is a smart contract that simplifies the process of issuing and redeeming Rebalancing Sets. It allows a user to acquire or dispose of a Rebalancing Set directly to and from ether in a single transaction utilizing liquidity from decentralized exchanges.

7 Exchange Issue

7.1 Overview

Acquisition of components for Set issuance is a tedious process. Users must take multiple steps to aggregate components into a single address including: gaining access via various centralized (requiring KYC/AML) or decentralized exchanges, transferring components into a single address in the correct quantities, approving tokens to the TransferProxy, and calling issue function in Core.

Exchange Issue is a sequence of steps in which users can issue a Set using liquidity from various decentralized liquidity pools or exchanges in a single step. Initially, Set will support orders from 0x V2 [2] and Kyber Network [6]. Eventually, we are exploring supporting other liquidity sources and aggregators such as Uniswap, OasisDex, and Bancor.

To execute an Exchange Issue, users compile exchange messages (defined below) and specify their desired Set, the payment tokens ("Send Tokens") they are willing to pay, and the components they need to acquire ("Receive Tokens"). Generally, users will pay with a single Send Token such as ETH or Dai, though the system is configured to allow multiple payment tokens. Services that host exchange orders and provide a user interface can make it easier for users to aggregate exchange orders and execute exchange issuances.

The exchange issue is initiated when the exchange issue call data and a list of exchange orders are submitted to the ExchangeIssuanceModule contract. If the call data is valid, the following sequence occurs in a single transaction:

- The send tokens are transferred from the caller to the appropriate ExchangeWrappers (defined below).
- The exchange orders are routed to their appropriate ExchangeWrappers where each order is sequentially executed utilizing the funds offered by the caller.
- Utilizing the caller's existing components and the newly acquired tokens, the module mints a new Set token to the caller.

7.2 Exchange Issuance Parameters

Exchange issuance utilizes a composite data type ("Exchange Issuance Parameters") that is designed to handle transactions that can involve variable numbers of send tokens, receive tokens, and supported exchanges. Any number of payment tokens can be used to pay for any number of receive tokens on any number of exchanges. This data type is used for both exchange issue and redeem operations (redeeming a Set into specific receive tokens). This data type is passed in to the ExchangeIssuanceModule when a user initiates an exchange issuance transaction.

The exchange issuance parameters is defined as follows:

Exchange Issuance Params	Type	Description
Set Address	address	Address of the Set to Issue or Redeem
Quantity	uint256	Quantity of Set to Issue or Redeem
Send Token Exchange Ids	uint8[]	Ids of exchange wrappers to transfer send tokens to. Exchange wrappers must approved in Core to be valid
Send Tokens	address[]	List of tokens to pay with. Duplicate Send Tokens are allowed to allow transfer to various exchange wrappers. During exchange redeem, these must be the Set components.
Send Token Amounts	uint256[]	Quantity of each send token.
Receive Tokens	address[]	Address of the tokens to receive. During exchange issue, these must be the Set components. Duplicates are not allowed.
Receive Token Amounts	uint256[]	Minimum send tokens to receive after exchange order execution.

7.3 Exchange Messages

Exchange messages are data packets that are included in the execution of an exchange issuance operation. These messages encode the metadata required by each Exchange Wrapper to execute trades on their respective exchanges. The requirement for valid exchange orders is that the results of the orders cumulatively generate the receive tokens and receive token amounts as specified in the Exchange Issuance Parameters.

Because the data requirements are different from each exchange, we encode the data in a bytestring with a common exchange-header. For example, 0x exchange orders accept a signed order, fill amount, and signature. Developers can use the setProtocol.js javascript library to format and encode the data into a format that is parsable by the associated exchange wrapper.

Exchange messages are grouped by exchange type, concatenated, and encoded in bytestring format. The design of the data packet has been inspired by networking protocol design. See below for a high-level diagram of the order data encoding:

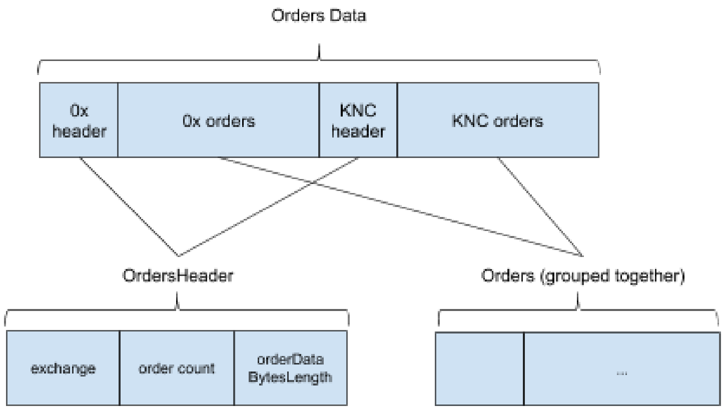

Figure 5: OrderData (input)

The header and body for each exchange's order data is formalized as follows:

Order Data	Type	Description
Exchange	uint8	Enumerated exchange wrapper identifier. The enumeration mapping state is stored on Core
Order Count	uint8	The number of orders for the current orders body for the particular exchange
Order Data Bytes Length	uint256	Length in bytes of orders in orders body
Orders	bytes	Orders concatenated in byte format

All exchange orders for a particular exchange must be packed together. The protocol currently supports two exchange wrappers out of box: 0x v2 and Kyber Network. One single exchange order does not need to net the full required amount for a component. For instance, a user may submit two 0x orders or one 0x order and one Kyber trade for a component of the Set to issue. Order bodies for each wrapper can be found below:

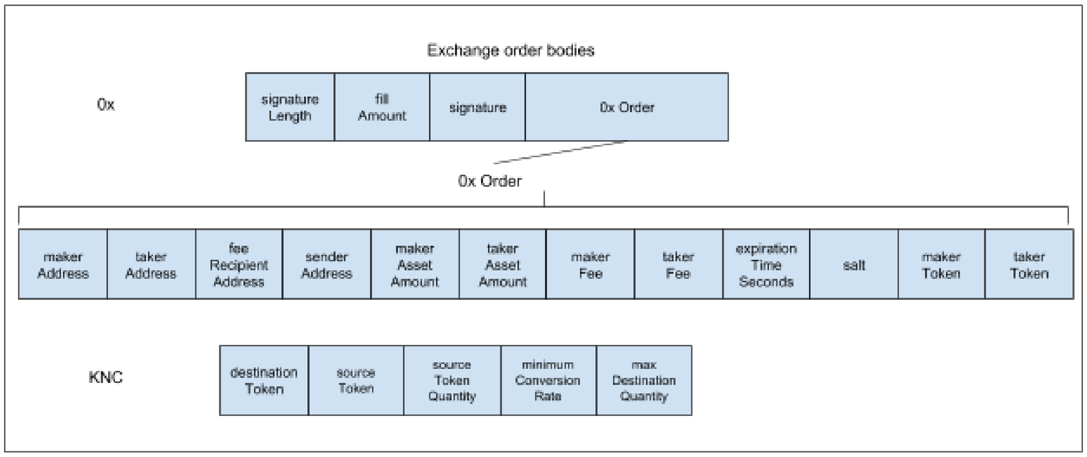

Figure 6: Exchange Order Bodies (KNC and 0x)

7.3.1 0x

To fill a 0x Protocol order, the user must supply the order data, the associated order's maker signature, and taker token fill amount. In addition, since 0x supports variable length signatures. the signature length must also be included to correctly unpack the maker signature.

During exchange issue, the maker token address of the 0x order is the receive tokens that fulfills the issuance. During redemption, the taker token is the Set's component that the user offers as the send token.

The order data is formalized below:

0x Order Header	Type	Description
Signature Length	uint256	Byte array length of the 0x order signature
Fill Amount	uint256	Fill amount of the 0x order quantity
Signature	bytes	Byte array representation of ECDSA signature

0x Order	Type	Description
Maker Address	address	Address of the 0x order maker
Taker Address	address	Address of the 0x order taker, usually 0x0. The exchange wrapper resolves the order as the taker
Fee Recipient Address	address	Address of the fee recipient of the 0x fee
Sender Address	address	Address of the 0x order taker if filling on behalf of another party, usually 0x0
Maker Asset Amount	uint256	Quantity of maker token in the order
Taker Asset Amount	uint256	Quantity of taker token in the order
Maker Fee	uint256	ZRX fee quantity attributed to the maker
Taker Fee	uint256	ZRX fee quantity attributed to the taker
ExpirationTimeSeconds	uint256	Expiration time of the 0x order in unix time in seconds
Salt	uint256	Arbitrary number to facilitate order uniqueness
Maker Token	address	Address of the ERC20 that the 0x maker wishes to trade
Taker Token	address	Address of the ERC20 that the 0x taker wishes to trade

7.3.2 Kyber

Kyber trades are a data type that encodes information for swapping source tokens (send token) for destination tokens (receive token). Since Kyber trades do not encompass an off-chain element, the caller only needs to specify the quantity of source token to trade and the minimum acceptable rate between the source and destination token. The user will need to assess the rate through Kyber Network contracts before submitting the order to ensure the destination token quantity can be acquired.

During exchange issue, the destination token of the Kyber Trade should be the component / receive token. During exchange redemption, the Set's component should be the source token.

The data is formalized below:

Kyber Trade	Type	Description
Destination Token	address	Address of the receive token
Source Token	address	Address of the send token
Source Token Quantity	uint256	Quantity of send token to use for this trade
Minimum Conversion Rate	uint256	Minimum conversion rate allowed in order to guarantee the quantity of destination token is achieved by the trade
Max Destination Quantity	uint256	Quantity of destination token to acquire through this trade

The exchange issue flow is formalized as follows:

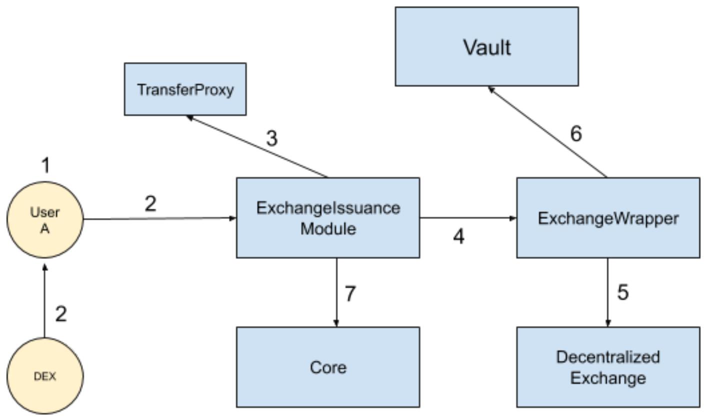

Figure 7: Exchange Issuance Flow

1. User desires to issue a Set (A,B,C) composed of 1x A, 1x B, 1x C. User specifies the address of A,B,C, quantity to issue, send tokens, receive (or component) tokens, receive tokens quantities needed, and send token amounts willing to pay in the Exchange Issuance Params.
2. User then looks at a separate decentralized exchange for exchange orders that fill 1x A, 1x B and 1x C. User A is incentivized to find orders that minimize cost.
3. The User's send tokens gets transferred into the appropriate ExchangeWrapper(s) via the Transfer-Proxy.
4. For each exchange, the Exchange Issuance Module sends the order data to associated ExchangeWrapper.
5. The order gets parsed by the ExchangeWrapper and gets executed on the decentralized exchange with the ExchangeWrapper as the taker of the order.
6. Each of the acquired receive tokens then gets transferred to the Vault and attributed to the User. Any extra send or received tokens is returned to the User.
7. Now that the User has all the required component tokens to issue A,B,C, the issue function in Core is called, minting the specified quantity of A,B,C.

8 Exchange Redeem

8.1 Overview

Exchange Redeem is the process of breaking up a Set into its components and using liquidity from decentralized exchanges to consolidate the value into a single or multiple ERC20 tokens in a single atomic transaction. Mechanically, it is the inverse of Exchange Issue and utilizes the same ExchangeIssuanceParams data structure and exchange wrappers. The ExchangeIssuanceParam inputs differ in that the send tokens must be the Set's components and the receive tokens the token to consolidate into.

The exchange redeem flow is formalized as follows:

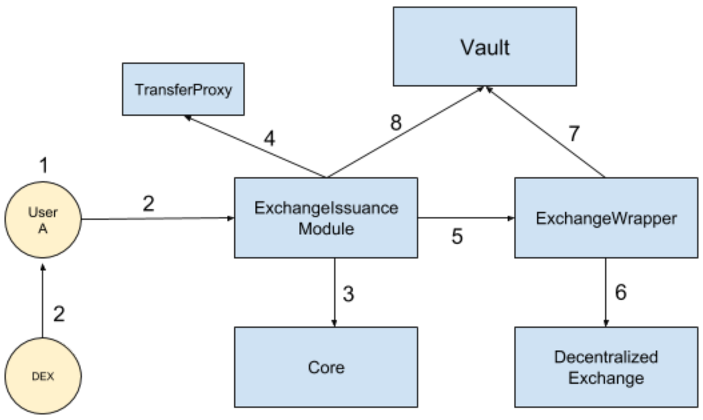

Figure 8: Exchange Redemption Flow

1. User desires to redeem a Set (A,B) composed of 1x A, 1x B into receive token (R). User specifies the address of A,B, quantity to redeem, send tokens, send quantities, receive token address, and receive token quantities.
2. User then looks at a separate decentralized exchange for exchange orders that dispose of 1x A and 1x B for the receive token. User is incentivized to finds orders that maximize the total receive token quantity. The User then submits the orders along with the ExchangeIssuanceParams to the Exchange Issuance module.
3. The Exchange Issuance module redeems the user's Set Token for components A and B into the vault.
4. The Components / Send Tokens are then transferred from the vault to the appropriate Exchange Wrappers.
5. For each exchange order, the Exchange Issuance Module sends the order data to ExchangeWrapper.
6. The order gets parsed by the ExchangeWrapper and gets executed on the decentralized exchange with the ExchangeWrapper as the taker of the order.
7. Each of the acquired receive tokens then gets transferred to the Vault and attributed to the User. Any unattributed tokens are returned to the User in the Vault.
8. Finally, the receive tokens are then withdrawn from the Vault to User A.

9 Rebalancing

9.1 Overview

Rebalancing is the process of realigning the weights of a portfolio, which involves periodically purchasing and selling components to maintain a certain allocation. Typically, users rebalance their portfolios actively by reviewing their portfolio, calculating the optimal allocation, and then trading their excess tokens for the missing tokens needed to create the new portfolio. In the traditional finance world where users purchase Index Funds or ETFs, fund managers (the counterparty) would perform the portfolio composition selection and rebalancing execution on behalf of users.

We introduce Rebalancing Set Tokens ("Rebalancing Set(s)"), a Set Token composed of another Set Token ("current Set" or "Base Set"). Using logic added to Rebalancing Sets, the currentSet can be transitioned to a new Set Token utilizing the guidance of a manager entity. Rebalancing Sets take a novel approach to rebalancing execution which completely removes counterparty risk. Rebalancing Sets can be configured to retrieve inputs regarding the most up to date Set Token through trusted or trustless sources.

Rebalancing Sets have many use cases including:

- **Tracking the Market Index**: Users who own Rebalancing Sets can passively track the market index by owning a single ERC20 token. Once users have acquired a Rebalancing Set, future rebalances execute behind the scenes, and the users do not need to perform any additional transactions, mimicking the user experience of traditional ETF and Index Funds.
- **Automating Trading Strategies**: Using a smart contract, trading strategies can be mechanized and automatically executed by holding a Rebalancing Set token.
- **Delegating Portfolio Reallocation**: Users can create a Rebalancing Set that represents their portfolio and delegate future reallocations to a trusted third party or to the wisdom of the crowds

Users can generate Rebalancing Sets by acquiring the currentSet defined on the Rebalancing Set and calling issue. Managers can then make updates by creating a proposal to modify the currentSet to another Set Token. During the proposal period, users are free to maintain their Rebalancing Set if they agree with the proposed change or redeem their Sets for the collateral backing if they disagree. Rebalancing execution is performed through a Dutch Auction, in which the outdated Set Token's components are auctioned for the new components needed in the proposed Set Token. When the auction has concluded, the user's Rebalancing Set Tokens are backed by the new Set Token at the auction conversion rate.

In a Rebalancing Set system, the participants include:

- **Users**: Rebalancing Sets enables users to get exposure to an automatically balanced crypto portfolio by simply acquiring an ERC20 token on an exchange or by minting a Rebalancing Set themselves. Only if the user decides to opt out of a rebalance proposal, does he or she need to get involved.
- **Manager**: Each Rebalancing Set has a manager that creates proposals (defined later) which dictate how a Set is rebalanced. The manager entity could be a person or a smart contract that has rebalance rules programmed in.
- **Rebalance Bidders**: During a Rebalancing Set's rebalancing period, profit-driven traders can help execute the rebalance by injecting components of the new Set Token in exchange for the ejected components of the old Set Token.

9.2 Lifecycle

The Rebalancing Set life cycle has four possible states: Default, Proposal, Rebalance, and Drawdown. Each state (outside of Drawdown) enables functions that help facilitate rebalancing from one Set Token to another. The Set can only move one direction through the cycle; there is no way to reverse or fast forward states.

See a diagram of a typical Rebalancing SetToken lifecycle below:

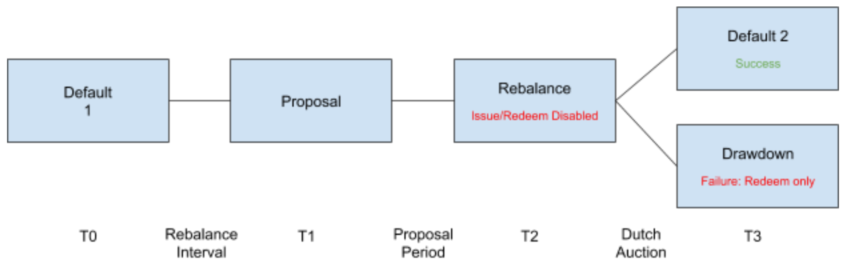

Figure 9: Rebalancing Set Token Lifecycle

9.2.1 Default

The Default state of a Rebalancing Set is the dormant stage in between rebalances. The Rebalancing Set Token will be in this state the majority of the time. In this stage, a Rebalancing Set Token operates no differently from a vanilla Set Token. Anyone can issue and redeem Rebalancing Set Tokens by calling the issue and redeem functions in Core.

9.2.2 Proposal

A Rebalancing Set moves from the Default to Proposal when the Rebalancing Set Token manager successfully submits a rebalance proposal. A proposal can only be submitted after the Rebalancing Set's Rebalance Interval has elapsed since the last rebalance. The proposal includes data about the suggested Set and auction execution parameters.

During the Proposal period, token owners can review the proposal and opt out of the rebalance if they disagree with the terms. The length of the proposal period is defined upon contract instantiation. All the basic Set token functions including issue and redeem remain available to all Rebalancing Set holders. While still in the Proposal state, a rebalance can also be re-proposed in the circumstance where there is component price volatility or an error is made. When the proposal period duration has elapsed, the Rebalancing Set can be transitioned from the Proposal state into the Rebalance state.

9.2.3 Rebalance

After the Proposal period has elapsed, the Rebalancing Set can be moved to the Rebalance Stage. During this stage, the Rebalancing Set facilitates the transformation of the obsolete Set into the proposed Set. During this state, issue and redeem functionality is paused.

The facilitation of converting the obsolete Set into the next Set is performed in a series of steps. First, the obsolete Sets are redeemed for their components. Then, the Rebalancing Set runs a dutch auction to facilitate the exchange of obsolete components for the required new components (see next section for more detail). Finally, the updated Set is issued using the new components. If this process has been carried out successfully, the Set can be transitioned back into the Default stage - now tracking the new proposed Set.

9.2.4 Drawdown

In the event that the dutch auction time has passed the pivot point, a bid has been performed, and auction can't be settled, a Rebalancing Set token can be transitioned into the Drawdown stage through the endFailedRebalance function. This would be called in a situation where something unexpected has occurred during an auction, and the user's component tokens need to be salvaged.

During the Drawdown stage, issuances and redemptions are disabled, and transitioning out of this state is not allowed. Token owners can retrieve their collateral by calling redeemFromFailedRebalance on the RebalanceAuctionModule. No further rebalances can be carried out, effectively disabling the token.

9.3 WhiteList

An auction could result in undesirable results if the proposed Set included a component that was faulty or lacked liquidity. Some of these adverse scenarios include:

- **Pausable Tokens**: If a token is paused during a rebalance, that token will not be able to be transferred. A manager could include a pausable token in the proposal, pause it during an auction, which causes all bids to revert. The manager could then unpause the token when sufficiently advantageous prices have been reached and collude with miners to front-run any bids and receive the collateral for a large discount. Known tokens with pausability (i.e. Status) should be treated with caution.

- **Manager-issued/Owned Tokens**: A manager could propose a rebalance that includes a token that only he/she owns. By waiting for the price to increase, they could effectively retrieve all of the collateral from the rebalancing set token by trading this unique malicious token.
- **Rebalancing Set Token**: If a rebalancing Set token was included as a component of the collateral Set, it could lead to rebalance auction liquidity issues when the component Rebalancing Set Token happens to rebalance at the same time as the parent. This is due to the fact that issuances of parent Set requires the child components to be tradable.

To combat these problematic scenarios, Set Protocol utilizes a whitelist, a smart contract with a list of approved tokens, to prevent proposals that can result in incongruent auctions. During a proposal, the proposed Set's component tokens are checked against the whitelist.

9.4 Dutch Auction

During a rebalance, the Rebalancing Set initiates a public auction where outdated components are exchanged for the latest components. The Rebalance Set uses an auction structure where the lowest price ratio (current components / next components) is initially offered, and the price ratio increases as a function of a predefined proposal price curve.

Ideally, each auction is configured in a manner that 1) allows exploration of fair value, 2) minimizes slippage for the Rebalancing Set Token holders, 3) is not manipulable by the manager, and 4) ensures a completion of the auction in a reasonable amount of time. The protocol's auction price libraries (defined later) and rebalancing set token factory contain validations and restrictions that aid to constrain auction instances that result in undesired and invalid behavior.

During the auction, users ("Bidder") can submit transactions to execute a trades ("Bid") to exchange the new components for the old at the current price ratio. As the price ratio increases, the price becomes more and more favorable for Bidders. The auction is concluded when all the old components have been replaced with the new components.

9.4.1 Bidding

In the dutch auction, bidding is the act of executing a rebalance. When users bid, they inject new Set components to facilitate a rebalance and in return immediately receive the old Set components. Bidders are incentivized to transact when there is a spread between the cost of acquiring the injected components and the received components. Bidding is open to anyone, and multiple parties can transact as long as the dutch auction is running.

Bidding has been designed to minimize the capital required to participate and price risk for bidders. We achieve this by only requiring injections of constituent tokens that are involved in the rebalance and providing access to returned tokens as soon as the bid is recorded. This way, the bidder only needs to acquire the components required by the system to undergo the rebalance.

9.4.2 Price Determination

During an auction, the bid transaction is represented by a combination of two lists, inflow and outflow tokens ("token flows"). The inflow tokens represent the tokens that a trader contributes during a bid. The outflow tokens represents the tokens that a trader receives for contributing the inflow tokens.

The magnitude of the inflow and outflow tokens changes as a function of the price ratio, which is defined by a priceNumerator and priceDivisor on the Price Curve Library. The price ratio is calculated using the auction parameters in the proposal and the time since the auction began. Each rebalance references a Price Curve Library that is used to calculate the price of the bid. Price Curve Libraries are smart contracts that contain mathematical logic for dictating the shape the auction price as a function of time (e.g. linear, log, cubic, etc.).

In the auction set up, token flows are calculated for both the currentSet and nextSet relative to a standardized bid amount (set to be the maximum of the two Set's naturalUnits). These flows define the amount of each component token needed to redeem and issue a standardized bid amount of the currentSet and nextSet, respectively.

The Rebalancing Set's "getBidPrice" function takes the current price ratio and uses the previously calculated token flows to determine the net token flows required to issue a standardized bid amount of the nextSet, given by the following equation:

$$standardTokenFlow_i = nextSetComponent_i - priceRatio \times currentSetComponent_i \tag{13}$$

Where a positive number reflects a token inflow (from bidder) and a negative number represents a token outflow (to bidder).

Since bid quantities represent amounts of currentSet the bidder wishes to rebalance, the equivalent amount of nextSets being rebalanced to at the current price ratio must be found and multiplied by the standard token flows to determine the final inflow and outflow for the submitted bid:

$$totalTokenFlow_i = \left(\frac{standardBidAmount}{priceRatio}\right) \times standardTokenFlow \tag{14}$$

9.4.3 Initial Price Curve Libraries

The LinearAuctionPriceCurve will be the initial deployed price library - which employs a 2-part price algorithm. It has a linearly increasing slope until it reaches the pivot time, in which the slope becomes exponential. The curve is designed to disallow problematic proposals and allow sufficient exploration of fair value.

The exploit scenarios the price curves were designed to mitigate are:

1. **Starting auction prices that exceed fair value**: A manager could initiate an auction where the price has already passed fair value. The manager could then collude with miners and front-run any bids in order to transfer the token collateral at a heavy discount.
2. **Auction parameters such that fair value will not be reached**: A proposal could be generated with an end time that is realistically unachievable (e.g. an end time of year 3000) - resulting in a rebalance where the funds could be locked indefinitely.
3. **Insufficient price exploration through fair value**: Managers could propose a rebalance where the auction duration is insufficient for ample price exploration. For example, a one minute rebalance would be insufficient to allow enough bids due to block generation time limitations.

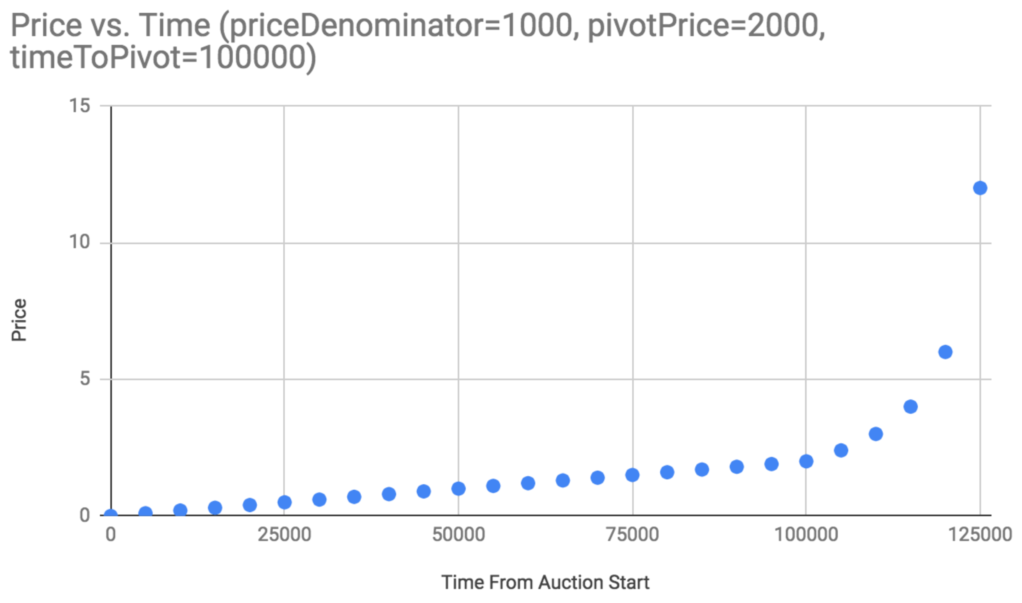

Figure 10: Example Price Curve

The price curve has three stages:

- **Stage 1**: A linear curve starting at 0 with manager defined end price and elapsed time. This section of the curve is the only part the manager has input on, however the inputs are limited to an acceptable range. The priceNumerator is the only output that is changed, the priceDivisor remains fixed.
- **Stage 2**: Here the protocol takes over to attempt to hasten an end to the auction. The curve is exponential and lasts for 500 minutes, losing price granularity as the auction continues. The exponential price curve is achieved by decaying the priceDivisor by .1% of the original priceDivisor every 30 seconds.
- **Stage 3**: The final stage is also protocol driven and is just an extension of Stage 2 where the price moves linearly but at a rate equal to the pivotPrice (i.e. if pivotPrice is 2000, the slope of the curve will be 2000). Here the priceDivisor is fixed at 1 and the priceNumerator increases linearly.

9.5 Specification

The Rebalancing Set is built to conform to the Set token interface so that issuance, redemption, and creation can all be executed through Core. The additional functionality and state held on the Rebalancing Set is used to facilitate the rebalancing process.

9.5.1 Creation

During Rebalancing Set creation, the creator passes along the typical information required to create a Set such as factoryAddress, components, units, and naturalUnit as well as addition information in an otherData parameter that is used to govern permissions and state on the token. The required parameters are defined below:

Property	Type	Description
factoryAddress	address	The address of the factory to create from
components	address[]	The address of initialSet, placed at the index 0 of the components array. Must be length of one and passed in an array in conformity to the Set Token interface.
units	uint256[]	The amount of initialSet that equals one share, aka unitShares. Must be length of one and passed in an array in conformity to the Set Token interface.
naturalUnit	uint256	The least divisible issuance value of the Rebalancing Set
name	bytes32	The name of the new Rebalancing Set
symbol	bytes32	The symbol of the new Rebalancing Set
otherData	bytes	The manager, proposalPeriod, and rebalanceInterval are all encoded and passed as bytes

In the constructor function for the Rebalancing Set, the following parameters (along with their definition) are passed in:

- **manager**: Ethereum address that has the capability to propose rebalances and set a new manager.
- **factory**: Factory used to create the Rebalancing Set. The currentSet parameter is initiated in the constructor.
- **currentSet**: The address of the Set currently underlying the Rebalancing Set. The currentSet parameter is initiated in the constructor and updated after each rebalance.
- **unitShares**: The quantity of the currentSet tokens that corresponds to one Rebalancing Set token. The unitShares parameter is initiated in the constructor and updated after each rebalance.
- **proposalPeriod**: The proposal period is the minimum length in seconds between a proposal and the beginning of a rebalance. The proposalPeriod parameter is initiated in the constructor.
- **rebalanceInterval**: The rebalance interval is the minimum length in seconds between end of previous rebalance and beginning of the next possible proposal period. The rebalanceInterval parameter is initiated in the constructor.
- **naturalUnit**: The natural unit is an integer value that represents the least divisible issuance value of the Rebalancing Set.

9.5.2 Issuance/Redemption

The process of issuing and redeeming a Rebalancing Set Token is identical to the process of issuing and redeeming a standard Set token. A user calls the issue or redeem function in Core, specifying the Rebalancing Set and the quantity they want to issue/redeem. A Rebalancing Set Token is issued/redeemed using the currentSet as collateral at the rate of the current unitShares. Issuances and Redemptions are disabled during the Rebalance stage.

9.5.3 Propose

Propose is a function on the Rebalancing Set Token that allows a Rebalancing Set Token's manager to pass the address of the Set to rebalance to and metadata associated with the auction parameters. A proper proposal is one that is in line with the token holder's expectations and generates auction parameters that allow for an efficient auction.

The function accepts the following as input:

Property	Type	Description
nextSet	address	The Set that is being rebalanced in to. This address becomes new currentSet at end of Rebalancing period. The Set must be created by a Factory through Core.
auctionLibrary	address	Address of the auction price library to be used for Dutch Auction pricing. The Auction Library must be tracked by Core.
auctionTimeToPivot	uint256	The time in seconds after the auction start time in which the price pivots from a Linear to Exponential curve. The time to pivot must be within the range specified (e.g. 6 hours to 3 days).
auctionStartPrice	uint256	The price numerator that the auction starts at. The start price supplied must be within the range allowed by the Price Library.
auctionPivotPrice	uint256	The price numerator in which Linear price curve switches from linear to exponential.

The proposal interactions are formalized as follows:

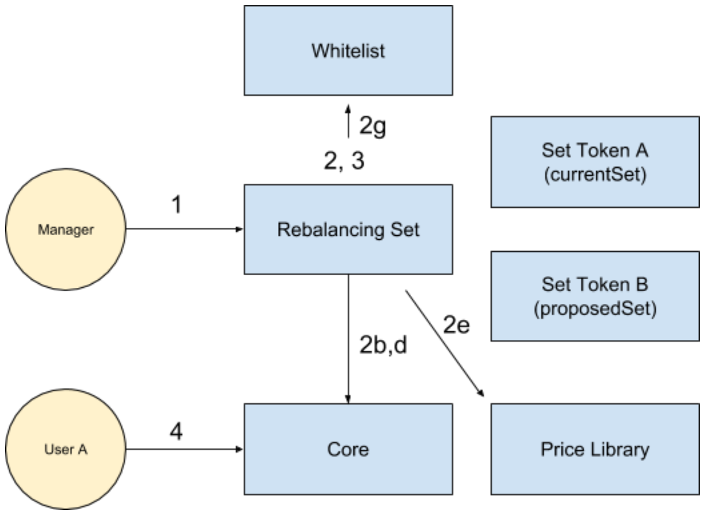

Figure 11: Proposal

1. The Manager calls proposal and passes in the address of Set Token B (the next or proposed Set), the address of the auction library, the auction time to pivot, auction start price, and auction pivot price.
2. The Rebalancing Set makes the following validations:
   (a) The sender is the Rebalancing Set's manager
   (b) The proposed Set address is valid and is tracked by Core
   (c) The price library address is valid and is tracked by Core
   (d) The current timestamp is greater than the sum of the last rebalance timestamp and the rebalance interval
   (e) The start price, pivot price, and time to pivot are valid inputs for the price library
   (f) The proposed natural unit of the Set is a multiple of the current set natural unit or vice versa. This ensures that there are no rounding errors when calculating token inflows and outflows. Rounding errors in the initial calculation would propagate throughout all the bids, and potentially short Rebalancing Set holders.

   (g) The components of the proposed Set are approved in the whitelist

3. The Rebalancing Set then updates the current state parameters and updates the rebalance State to the Proposal Phase.
4. During the Proposal Phase, User A can evaluate the Manager's proposal and opt out. If User decides not to participate, the User calls Redeem.

9.5.4 Start Rebalance

Start Rebalance is a function on the Rebalancing Set Token that initiates the auction process. It can only be called when the function is in the Proposal state and after the proposal period has elapsed. This function can be called by anyone. When Rebalance is called, the Rebalancing Set moves into a "Rebalance" state where issue or redeem functionality is disabled.

The rebalance initiation process is formalized as follows:

1. **Redeems existing Set**: The Rebalancing Set Token calls redeem on its balance of existingSet. These components are stored in the Vault under the Rebalancing Set Token's custody.
2. **Sets up bidding parameters**: To track and calculate token flows during the auction, three arrays and two parameters need to be created:
   - **combinedTokenArray**: Array containing the union of tokens present in the currentSet and rebalancingSet.
   - **combinedCurrentUnits**: Array containing the units for the currentSet ordered to match the combinedTokenArray . Units are calculated using the maximum between the current and next Set natural units as a normalizing quantity. If token isn't present in currentSet then it is marked as 0.
   - **combinedNextSetUnits**: Array containing the units for the nextSet ordered to match the combinedTokenArray ordering. Units are calculated using the maximum between the current and next Set natural units as a normalizing quantity. If token isn't present in nextSet then it is marked as 0.
   - **minimumBid**: Value correlating to the minimum currentSet quantity that can be inputted as a bid. This is calculated by taking the maximum of the currentSet natural unit and nextSet natural unit and multiplying it by the auction library's price divisor.
3. **Update State**: The Rebalancing Set then updates the current state parameters and updates the rebalance State to the Rebalance Phase.

9.5.5 Bid

Bid and bidAndWithdraw are functions on the Rebalance Auction Module ("Auction Module") smart contract that allow for the transacting of input and output components during a rebalance dutch auction. Bid injects the required component tokens where the ejected tokens remain in the Vault, and bidAndWithdraw allows the ejected tokens to be sent directly to the user's wallet.

To perform a bid, a trader calls the Rebalance Auction Module's ("Auction Module") bid function and specifies the address of the Rebalancing Set as well as the quantity of current Set to rebalance.

From there the Auction Module calls the "getBidPrice()" function on the Rebalancing Set in order to get the amount of each token that need to be taken from the bidder's wallet and also attributed to the bidder in the vault.

The bidding interactions are formalized as follows:

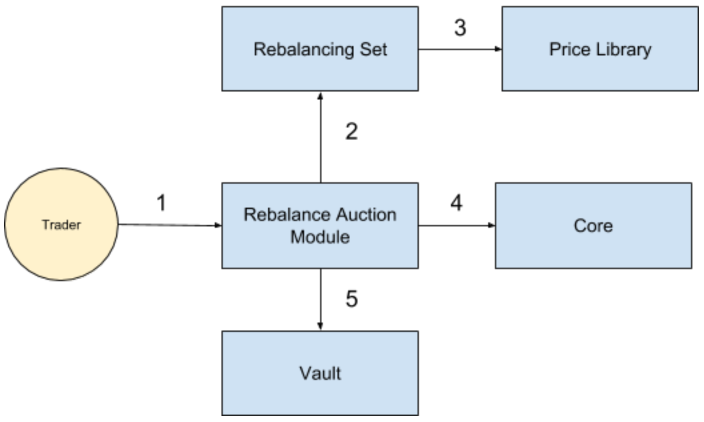

Figure 12: Bid

1. A trader calls the bid function on the Rebalance Auction Module, passing along Rebalancing Set Token Address and the Quantity as parameters.
2. The Rebalance module calls Rebalancing Set's placeBid function to retrieve the required tokens for the trader to deposit and withdraw based on the current price in the Rebalancing Set's auction.
3. The Rebalancing Set then calls the Auction Price Library to retrieve the current price, expressed as a price numerator and divisor. The bid price multiple is then calculated and returned the the Rebalance Auction module.
4. The Rebalance Auction Module then calls Core to deposit the Trader's inflow tokens into the Vault, attributing them to the Rebalancing Set.
5. The Rebalance Auction Module then calls Vault to withdraw the outflow tokens from the Vault to the Trader (during bidAndWithdraw)

9.5.6 Settle Rebalance

Upon the completion of the auction, settleRebalance is a function on the Rebalancing Set Token that is called to transition the state of the Set back to Default. The settlement interact is formalized as follows:

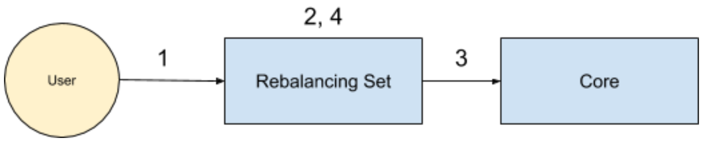

Figure 13: Settle Rebalance

1. A user calls the Rebalancing Set's settleRebalance function. Settlement can only be called when two conditions are met (1) the amount of currentSets available to be rebalanced is less than the minimum bid quantity, and (2) the rebalance state is "Rebalance".
2. The Rebalancing Set calculates the nextSet issue quantity and the rebalancing Set's new unit shares. The nextSet issue quantity is calculated by taking the minimum of the implied nextSet quantity issuable with each of the received components. The unitShares is calculated by taking the nextSet issue quantity and dividing it by the natural units outstanding, defined as the supply of rebalancing Set divided by the rebalancing Set natural unit. For a settlement to be called successfully, the unitShares calculation must be greater than 0.

3. Upon settlement, the Rebalance Token calls issue on Core, converting component tokens received in the bidding process into the nextSet.

4. Finally, four final state changes are performed: (1) nextSet is changed to currentSet, (2) unitShares is changed to the new calculated quantity, (3) the lastRebalanceTimestamp is set to the block timestamp, and (4) rebalanceState is set to "Default".

9.5.7 End Failed Rebalance

If the auction time has exceeded the auction pivot time, the endFailedRebalance function on the RebalanceAuctionModule can be called to transition the Rebalancing Set either to the Default or Drawdown state. If the auction time has exceeded the pivot time and there are no bids made, the Rebalancing Set is transitioned back to the Default state. If a bid has successfully been made, the function is called to transition a Rebalancing Set from the Rebalance to the "Drawdown" state.

If there have been no bids placed, the auction is reverted, the collateral is re-issued back into the underlying Set Token, and the state is changed back to "Default". The Rebalancing Set is operational and another proposal can be made after another Rebalance interval period.

If there have been bids placed during the auction, it is impossible to re-collateralize the Rebalancing Set to either the nextSet or the currentSet. Thus, the Rebalancing Set is placed in Drawdown state where token owners are only able to redeem their portion of the remaining collateral backing the Set. Once placed in Drawdown, there is no way to transition out of the state, issuances and redemptions are not allowed. Token owners must call redeemFromFailedRebalance on the RebalanceAuctionModule which transfers the owner's portion of their collateral to them in the Vault (thus there will be no reversions on the collateral transfer since it nevers hits the component token's contract).

10 Governance

Set Protocol intends to begin with a centralized governance in the early, formative days and shift over time to a community-based governance system. The governors will have the capability to perform the following actions:

- Add Modules, ExchangeWrappers, Price Libraries, and Factories. Each addition is a Time-Locked operation, which is an operation that requires a minimum period of time to elapse before being enacted.
- Remove Modules, ExchangeWrappers, Price Libraries, and Factories. In the event there is a problematic module, the governors have the capabilities to immediately sever a non-functioning smart contract without a timelock operation.
- Add (Time-Locked operation) and remove authorizable addresses in Vault and TransferProxy.
- Add (Time-Locked operations) and remove tokens from the Whitelist contract. The Whitelist is currently intended to enforce a Rebalancing Set proposal's base Set components.
- Disable and re-enable problematic Sets that may have been associated with a faulty Factory.
- Put the protocol into a ShutDown state where only Redeem and Withdraw functions are available.

Early on, governance actions will be administered a Gnosis multisignature wallet - where folks who have a vested stake in Set Protocol's success will be able to propose adjustments to the protocol. Individual DAO-member private keys will be held in cold storage.

In the early days when there are limited funds in the system and we expect rapid iteration of features, the time lock period will be set to 0. When the value in the vault exceeds ¿ the value of our company, then a 7-14 day timelock period will be enforced and the team will begin handing over governance to the broader community.

Generally, administrator intervention will occur in the following scenarios:

- **Critical Vulnerability**: A critical vulnerability in the modules, exchange wrappers, price libraries, and factories is found. In this event, administrators will remove access of the faulty smart contract from the system. If needed, the administrators will also disable any problematic Sets associated with such vulnerability.
- **Upgrades**: The team and open-source contributors will continue to develop improvements to the protocol and will introduce these upgrades through modules, exchange wrappers, and factories. Updates to Vault or TransferProxy authorizations would only ever be made if a new Core system were to be put into place.
- **WhiteList Introductions**: Over time, we may add additional components to the Whitelist that enable components to be rebalanced.
- **System Shutdown**: If the entire system is to be overhauled, the protocol will be put into a ShutDown state.

10.0.1 Alternative Governance Scenarios

Component Token Migrations: In the event that a significant component has been migrated to a new token address, the administrators of the protocol can add a TokenMigrationModule - which allows the attribution of a specific token to the owners of the retired token. This form of token migration could also potentially be used for airdrop attribution as well.

11 Future Work

Because we are still very early days in the development of Set, there are still many outstanding areas and features that we plan on developing. We plan on building Set to as meet the needs of the broadest possible community. Some of these include:

11.0.1 Additional Modules

- **Flash Lending Module**: A flash loan (first coined by Marble[8]) is a loan where the token must be repaid with a fee by the end of a transaction. It is useful for a variety of purposes including decentralized exchange arbitrage. Collateral held in the Vault smart contract can be lent out for any purpose in a flash loan.
- **Rehypothecation Module**: Users who wish to generate interest / additional return can lock up their collateral and automatically lend their components via lending protocols including Compound, Dharma, etc.

  To govern the terms and parameters associated with rehypothecation, it is plausible to introduce a Maker-style governance token where a percentage of the interest generated is used to burn of compensate token holders. However, there are currently no plans for a protocol token.

11.0.2 Alternative Asset Support

- **ERC-721 Sets**: Sets that represent fractional ownership of a collection of ERC721 Assets[7]. Some of these use cases include fractional ownership of non-fungible tokenized assets such as art, title deeds, and sports teams.

  Implementing issuances of ERC721 Sets is trivial, but redemption logic is a bit more complicated. Some governance questions that arise include: 1) How do the token-holders decide when to redeem their shares for non-fungibles? 2) What happens when there is disagreement between shareholders (e.g. divorce) 3) In the case that shareholders decide to sell, how are assets distributed? Do they get auctioned for fungible assets first?

12 Summary

We introduce the idea of a Set, which is a single token that represents a basket of tokens. Sets are a powerful tool for allowing us to hide away the details of underlying tokens and focus on higher-level concepts. It represents a new asset class that allows more efficient transactions, is fully collateralized, and is trustless. Sets serve as a fundamental building block for composing more complex financial instruments. Sets are a primitive and serve as a powerful tool that allows developers and product-creators to build increasingly complex decentralized applications that are user-friendly and intuitive. There is a limitless number of use cases for Set.

13 References

1. Coinmarketcap. https://coinmarketcap.com/ (Accessed November 2017)
2. 0x project. https://0xproject.com/pdfs/0x_white_paper.pdf
3. dy/dx. https://dydx.exchange/
4. dharma. https://dharma.io/whitepaper/
5. ERC20 https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
6. Kyber Network. https://kyber.network/https://kyber.network/
7. ERC721. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md
8. Marble Protocol. https://marble.org/