Xeta Reality Handbook

Documentation for Modules and Programs

Version 0.8.1
23rd December, 2021
https://xetareality.com https://xeta.network

Address

Intro

An address represents an account on Xeta and is either a token, pool or private wallet. Every public key is automatically an address that can receive funds, and users can customize their address profiles with name, description, links, image previews, and other info.

Use Cases:
Token and pool addresses
Personal wallet address (equivalent to a public key)
Custom profiles (such as an address representing an NFT collection)

Functionality

When a user creates a managed or self-managed wallet, a public key is automatically generated, serving as a public wallet address. This address is public and can receive permissionless transfers from anyone. A user can update their connected wallet address to represent a profile or a collection.

Methods

Update Address

Update an address by adding custom info such as name, preview image, etc.

Requirements: `Xeta wallet`
Outputs: `NFT representing profile data`
Inputs:
`Name` - a name for the address
`Description` - a description for the address
`Links` - profile links
`Meta` - profile data as a JSON object
`Preview` - preview image for the address
`Category` - category of the address

Allowance

Intro

An allowance can be given to another address, which is then allowed to spend a particular fungible token amount on behalf of the creator.

Use Case: Allowing someone to spend funds on your behalf

Functionality

A user sets an allowance, granting another spender the right to spend a particular fungible token for a certain amount. The spender then can create transfers by specifying the from address to be the address that granted the allowance. The allowance creator can increase and decrease their given allowances at any time. The spender can make transfers until the creator's allowance is depleted or adjusted to zero.

Methods

Update Allowance

Allows a spender to spend your tokens for the specified amount.

Requirements: `Token balance`
Outputs: `Allowance to spend`
Inputs:
`Spender` - Public address of the spender
`Token` - Fungible-token address
`Amount` - Allowance amount

Claim

Intro

Claims represent a promise or contract that is made between two parties. A claim creator issues a claim to someone representing an option that can be redeemed based on specified terms such as timeframe or amount. The claim owner can redeem the claim once the condition of the claim is met. The fulfillment of promises made by claims is not guaranteed and is dependent on the issuers' integrity and trust. However, if a pool with a verifiable program issues the claim, the claim resolution can be guaranteed through code.

Use Cases:
Claims between two users (credit claims, ownership claims, contracts, promises)
Claims between a pool and a user (liquidity claim, deposit claim, vote claim, lottery ticket claim, etc.)

Functionality

A claim creator creates a claim based on a specific agreement and issues it to a user who has the right to claim once the conditions are met. Most frequently, pools use claims as a ticket to the user, who can redeem the ticket. Created claims can be updated by the creator at any time until resolution. Claim creators can also mark claims as resolved, at which point the claim becomes settled and cannot be updated any longer. Owned claims can be transferred as long as they are not frozen.

Methods

Create

Enables a pool or user to issue a claim to someone.

Requirements: `Claim token balance/ownership`
Outputs: `Claim assigned to specified owner`
Inputs:
`Owner` - Holder who receives the claim
`Token` - Claimable token
`TokenAmount` - Claimable token amount
`XetaAmount` - Claimable Xeta amount
`Expires` - Expiry date for the claim
`Unlocks` - Date at which the claim becomes claimable
`Frozen` - Forbids claim transfers, if set to true
`Category` - Claim category
`Meta` - Custom metadata as a JSON object
`Answer` - Claimed answer
`Number` - Claimed number

Transfer

Enables the claim owner to transfer claim ownership.

Requirements: `Unfrozen claim`
Outputs: `None`
Inputs:
`Claim` - Claim hash
`To` - To address

Update

Enables the claim creator to update specific claim values.

Requirements: `Claim creator`
Outputs: `None`
Inputs:
`Claim` - Claim hash
`TokenAmount` - Claimable token amount
`XetaAmount` - Claimable Xeta amount
`Expires` - Expiry date for the claim
`Unlocks` - Date at which the claim becomes claimable
`Frozen` - Forbids claim transfers, if set to true
`Category` - Claim category
`Meta` - Custom metadata as a JSON object
`Answer` - Claimed answer
`Number` - Claimed number

Resolve

Enables the claim creator to close a claim and tag it as resolved.

Requirements: `Claim creator`
Outputs: `None`
Inputs:
`Claim` - Claim hash

Pool

Intro

Pools are autonomous accounts that execute a set of programmed instructions. Pools are generic wrappers for assets to enable decentralized functionalities such as auctions, launches, lending, locks, loots, lotteries, royalties, staking, swaps, and votes. Pools can hold tokens, make transfers, and create claims based on their underlying and specified program chosen at pool creation.

Use Cases: Reusable autonomous wrappers which execute programmable instructions (equivalent to smart contracts), enabling DeFi functionalities like auctions, launches, lending, locks, loots, lotteries, royalties, staking, swaps, and votes

Functionality

A user creates a pool for a particular token while specifying the desired pool program. This program can be specified as auction (creating an NFT auction), launch (launch-pads for new tokens), lending (allowing others to borrow tokens against XETA collateral), locks (locking/vesting tokens), loot (randomized token drops), lottery (token lotteries), royalty (rewards for NFTs), swap (decentralized exchange), vote (voting/betting mechanisms) or any custom program as written in Xeta's Turing-incomplete programming language.

During pool creation, the user might configure the pool using predefined parameters and modify how program instructions are executed (for example, by specifying a minAmount, the pool might only accept transfers with a certain amount).

Methods

Create

Create a new pool that executes instructions based on the specified program for a specified token.

Requirements: `Token balance/ownership`
Outputs: `Pool for the specified token`
Inputs:
`Token` - Token for the pool
`Program` - Pool program id
`Name` - Name of the pool
`Description` - Description of the pool
`Mechanism` - Mechanism, as required by specific programs
`Candidates` - List of candidates
`Rate` - Rate between 0-1
`Percentage` - Percentage, between 0-1
`Number` - Custom number as required by specific programs

`Answers` - List of answers
`Meta` - Custom metadata as a JSON object
`Expires` - Datetime of expiry
`MinAmount` - Min. amount for transfers to the pool
`MaxAmount` - Max. amount for transfers to the pool
`MinTime` - Min. time of claims created by the pool
`MaxTime` - Max. time of claims created by the pool
`TransfersLimit` - Max. number of transfers to the pool
`ClaimsLimit` - Max. number of claims by the pool
`TokenTarget` - Token target balance of the pool
`XetaTarget` - Xeta target balance of the pool
`TokenLimit` - Token limit balance of the pool
`XetaLimit` - Xeta limit balance of the pool

Token

Intro

Tokens are fungible (divisible) or non-fungible assets. Fungible tokens are similar to shares and represent fractional ownership of something. Non-fungible tokens represent objects via object URL (IPFS, S3, etc) or content hash (e.g. MD5 of bytes content).

Use Cases:
Fungible tokens (representing fractional ownership of something)
Non-fungible tokens (representing objects via URL or content hash in physical and virtual worlds)

Functionality

A user can mint fungible or non-fungible tokens. Fungible tokens represent fractional shares and have a symbol, supply, and optionally a reserve, which can be used to mint additional tokens at a later point in time. Non-fungible tokens represent objects in physical and virtual spaces by referencing an object URL (e.g. IPFS link) or a content hash. When fungible tokens are minted, a decentralized swap pool is created automatically, enabling the decentralized exchange of tokens.

Methods

Create

Create a fungible or non-fungible token.

Requirements: `None`
Outputs: `Funigble or non-fungible token`
Inputs:
`Name` - Name of the token
`Description` - Description of the token
`Links` - Links for the token
`Meta` - Custom metadata as a JSON object
`Preview` - Preview image for the token, e.g. icon for FT, image for NFT
`Symbol` - Symbol for FT
`Supply` - Supply for FT
`Reserve` - Reserve for FT, mintable at a later point in time
`Owner` - Owner for NFT
`Object` - Object representing NFT
`Content` - Content representing NFT
`Mime` - Mime of NFT object
`Frozen` - Disables transfer of NFT, if true
`Category` - NFT category

Update

Allows the creator to update a fungible or non-fungible token.

Requirements: `Token creator`
Outputs: `None`
Inputs:
`Token` - Token address
`Name` - Update name
`Description` - Updated description
`Links` - Updated links
`Meta` - Updated meta
`Preview` - Updated image preview
`Mime` - Updated object mime
`Frozen` - Frozen NFT status
`Category` - Updated NFT category

Mint

Allows the creator of a fungible token to mint additional tokens from its unallocated reserve.

Requirements: `Token creator`
Outputs: `Token transfer from factory to creator`
Inputs:
`Token` - Token address
`Amount` - Amount to be minted

Transfer

Intro

A transfer allows the transfer of fungible and non-fungible tokens between addresses.

Use Cases:
Transferring an amount of a fungible token
Transferring a non-fungible token

Functionality

A user with a certain fungible-token balance can transfer all or parts of the balance to another address. Additionally, a user who owns a particular non-fungible token can create a transfer to assign ownership to another address.

Methods

Create

Create a new fungible or non-fungible token transfer.

Requirements: `Token balance/ownership`
Outputs: `Token transfer from sender to recipient`
Inputs:
`To` - To address for the transfer
`Token` - Token to be transferred
`Amount` - Transfer amount for FT, empty if NFT
`From` - From address in case of allowance
`Message` - Custom transfer message

Auction

Intro

Auctions allow users to offer non-fungible tokens for sale. Participants can submit XETA-bids until the auction expires or the XETA limit is reached. Auction programs automatically handle the resolution by transferring the highest bid and NFT to their new or previous owners (depending on the auction status).

Use Cases:
Auctioning an NFT that a user owns in exchange for XETA

Functionality

The NFT owner creates an auction pool and sets a XETA target (minimum bid to consider the auction successful), a limit value (instant-buy bid), and a time of expiration. The NFT owner then deposits the NFT to the auction pool, and participants can place bids to the auction pool. When a participant places a bid higher than the previous bid, the pool automatically returns the earlier bid to the previous leader.

The auction outcome is determined when the auction expires or when the limit bid is reached before the expiration time. An auction is considered successful when the limit bid is reached before expiration, or when the target bid is reached at the time of expiration. Auction pools charge a commission of 2.5% for every concluded auction.

Upon success/failure of the auction, any Xeta user can call the resolve/cancel method of the auction pool, which then handles the asset transfers to the new or old owners. On auction failure (target bid not reached on expiry), the NFT is returned to the pool creator, and the highest bid is returned to the highest bidder. On auction success, the NFT is transferred to the highest bidder, and the highest bid is transferred to the auction-pool creator.

Methods

Transfer

Transfer a XETA bid to the auction pool.

Requirements: `Bid higher than previous bid`
Outputs: `Bid transfer from sender to pool`, `Return transfer from pool to previous highest bidder`
Inputs:
`Amount` - Bid amount

Resolve

Resolve a successfully concluded auction pool.

Requirements: `Pool closed or expired and bid >= xetaTarget`, `Pool active and bid = xetaLimit`
Outputs: `NFT transfer from pool to highest bidder`, `XETA transfer from pool to pool creator`
Inputs: `None`

Cancel

Cancel an unsuccessful auction pool.

Requirements: `Pool closed or expired and bid < xetaTarget`
Outputs: `NFT transfer from pool to pool creator`, `XETA transfer from pool to highest bidder`
Inputs: `None`

Create

Create an auction pool for a particular NFT.

Requirements: `Pool creator`, `Pool token ownership`
Outputs: `Auction pool`
Inputs:
`Token` - NFT address
`XetaTarget` - Minimum bid for auction to be considered successful upon expiry
`XetaLimit` - Maximum bid, equivalent to instant-buy bid
`TransfersLimit` - Maximum limit of transfers
`Expires` - Date and time of expiry

Deposit

Deposit an NFT to the auction pool.

Requirements: `Pool creator`, `Pool token ownership`
Outputs: `NFT transfer from sender to pool`
Inputs:
`Unlocks` - Datetime when NFT claim can be unlocked and withdrawn

Close

Closes an auction before expiry. Allows anyone to call cancel or resolve methods immediately.

Requirements: `Pool creator`
Outputs: `None`
Inputs: `None`

Launch

Intro

Launch pools allow creators to raise funds for a project by offering fungible tokens representing the project in exchange for XETA. Launch pools have a fixed rate of raising funds and a target (minimum) and limit (maximum) XETA amount. A successful launch pool (target amount reached before closure/expiry) automatically handles the liquidity deposit to the fungible token swap pool. Anyone who participated in the launch pool can then claim fungible tokens or recover their initial contribution if the launch didn't raise the target amount.

Use Case: Raise funds for a project and automatically distribute raised funds to liquidity and/or creators.

Functionality

A fungible-token owner creates a launch pool for a fungible token and shares it with users who can participate by contributing XETA tokens. Pool creators can decide how many tokens they want to offer and how much they want to raise as a minimum and maximum amount. Once a launch expires or closes, participants can claim back either tokens (upon success) or their XETA contribution (upon failure). Since the pool automatically handles the liquidity deposit, successfully resolved launches are automatically tradeable at the token swap pool.

Methods

Transfer

Transfer XETA to the launch pool to participate.

Requirements: `Active pool`
Outputs: `XETA transfer from sender to pool`, `Transfer claim from pool to sender`
Inputs:
`Amount` - XETA contribution amount

Claim

Claim tokens upon success or XETA contribution upon failure.

Requirements: `Concluded pool`, `Transfer claim`
Outputs: `Token transfer from pool to sender on pool success`, `XETA transfer from pool to sender on pool failure`
Inputs:
`Claim` - Claim hash

Resolve

Resolve launch pool after expiry or if creator manually closed pool.

Requirements: `Pool XetaBalance >= XetaTarget`
Outputs: `Liquidity transfer from launch pool to swap pool`, `Funds transfer from launch pool to pool creator`
Inputs:
`Token` - Pool token

Create

Create a launch pool for a fungible token.

Requirements: `Token balance`
Outputs: `Launch pool`
Inputs:
`Token` - Token to launch
`XetaTarget` - Minimum XETA target
`XetaLimit` - Maximum XETA limit
`Percentage` - Percentage for liquidity vs. creator
`TransfersLimit` - Maximum number of participants
`Expires` - Datetime of expiry

Deposit

Deposit the fungible tokens and automatically determine the swap rate, based on 50% of the deposited amount allocated for swaps, 50% allocated for liquidity.

Requirements: `Pool creator`
Outputs: `Token transfer from sender to pool`, `Deposit claim from pool to sender`
Inputs:
`Amount` - Amount to be deposited
`Unlocks` - Unlocks datetime for claim

Withdraw

Withdraw previously deposited tokens, or remainder for a closed launch.

Requirements: `Pool creator`, `Token balance in the pool`
Outputs: `Token transfer from pool to sender`
Inputs:
`Claim` - Claim hash

Close

Creators can close the launch pool manually before expiry.

Requirements: `Pool creator`
Outputs: `None`
Inputs: `None`

Lending

Intro

Lending pools allow users to borrow tokens against XETA collateral. Lending pools allow "short selling" a token or resolving claims without owning the token. At the same time, it enables lenders to earn interest by making their tokens available for borrowers.

Use Cases:
Short-selling a borrowed token based on the assumption that tokens can be purchased back at a lower price (contributing to more efficient markets)
Earning interest by lending out fungible tokens to borrowers

Functionality

A lending pool is created with a certain collateralization and interest rate. Lenders deposit tokens into the lending pool. Borrowers can then use it to borrow tokens while specifying the desired collateral. As long as the collateralization is kept above the minimum, users can keep tokens for as long as they wish and pay the respective interest rate upon settlement. If, however, the collateralization rate falls below the minimum, any user can liquidate the borrower's claim, which applies penalties equivalent to a one-year interest rate to the borrower's collateral.

All collected funds, via interest rate paid by the borrower, including liquidations, are collected and proportionally divided between lenders based on lending duration and lending amount. Lending pools make it possible for lenders to earn a steady and predictable interest rate.

Methods

Transfer

Lend tokens from the lending pool at certain collateralization.

Requirements: `Collateralization >= Min. collateralization`
Outputs: `XETA transfer from sender to pool`, `Token transfer from pool to sender`
Inputs:
`Token` - Pool token (token to borrow)
`Collateralization` - Desired collateralization rate
`Amount` - XETA amount to be transferred as collateral

Settle

Return previously lent tokens while paying an interest rate based on the rate specified by the pool and the lending duration.

Requirements: `None`
Outputs: `Token transfer from sender to pool`, `XETA transfer from pool to sender`
Inputs:
`Claim` - Claim hash

Liquidate

Liquidate someone's claim that has dropped below 75% of the minimum collateralization required by the pool.

Requirements: `Claim collateralization < 0.75 * min. collateralization`
Outputs: `XETA transfer from pool to borrower for remaining collateral`, `XETA transfer from pool to liquidator for finders reward`
Inputs:
`Claim` - Claim hash `Token` - Pool token

Deposit

Deposit tokens as specified by the pool to earn a specific interest rate.

Requirements: `None`
Outputs: `Token transfer from sender to pool`, `Deposit claim from pool to sender`
Inputs:
`Amount` - The deposited amount
`Unlocks` - A datetime when the deposit claim can be redeemed

Withdraw

Withdraw tokens that have been deposited previously to the lending pool.

Requirements: `Active deposit claim`
Outputs: `Token transfer from pool to sender`, `XETA transfer from pool to sender for interest earned`
Inputs:
`Claim` - Deposit claim hash
`Token` - Pool token
`Percentage` - Percentage to withdraw

Create

Create a lending pool for a fungible token.

Requirements: `None`
Outputs: `Lending pool`
Inputs:
`Token` - Token to lend/borrow

`Percentage` - Collateralization rate
`Rate` - Annual interest rate

Lock

Intro

Lock pools make it possible for anyone with a fungible-token balance to lock tokens for a specific time. Locks can be set to unlock after a particular date or be valid only before a date through an expiration mechanism. Furthermore, lock pools allow the transfer of lock claim ownership to other addresses.

Use Cases:
Locking fungible-tokens for someone (e.g. allocated for another entity as a form of vesting)
Locking fungible-tokens until a specific datetime
Locking fungible-tokens claimable after a particular date, and before an expiry date

Functionality

A lock pool is created for a particular token. Users can then use this pool to lock their tokens in exchange for a lock claim. This claim can then be redeemed after the locking period expires. Lock claims can be assigned to another address, allowing project creators to distribute tokens to others in a locked state.

Methods

Transfer

Create a new lock-claim for the token as specified by the pool.

Requirements: `Token balance`
Outputs: `Token transfer from sender to pool`, `Lock claim from pool to sender`
Inputs:
`Amount` - Fungible-token amount to be locked
`Address` - Address for which to lock the tokens
`Unlocks` - Datetime from which the claim will be claimable
`Expires` - Datetime at which the claim will expire and become non-claimable

Claim

Allows participants with a valid lock-claim to unlock tokens from the lock-pool.

Requirements: `Active lock claim`
Outputs: `Token transfer from pool to sender`
Inputs:
`Claim` - Lock claim hash

Loot

Intro

Loot pools enable token-creators to promote their NFTs in a randomized fashion. Loot pools are equivalent to NFT drops. Users can participate by paying a certain fungible-token amount and receiving a random NFT with a certain probability.

Use Case: Distributing or launching an NFT collection through randomized token distribution process

Functionality

The creator or owner of NFTs creates a loot pool and sets parameters such as the participation amount and win-probability. The creator deposits NFTs to the pool to be distributed. Participants can then enter by transferring the entry amount to receive a random NFT with a specific probability from the pool.

Methods

Transfer

Participate in the loot pool by transferring the required participation amount.

Requirements: `Positive NFT balance by pool`
Outputs: `NFT transfer to sender based on win-probability`
Inputs: `None`

Create

Create a new loot pool.

Requirements: `None`
Outputs: `Loot pool`
Inputs:
`Token` - A fungible token to be used as participation token
`Percentage` - Probability to win, between 0 and 1
`MinAmount` - Participation amount
`Expires` - Datetime when the loot pool expires

Deposit

Deposit an NFT as loot to the loot pool.

Requirements: `Pool creator`, `NFT ownership`
Outputs: `NFT transfer from sender to pool`, `Deposit claim from pool to sender`
Inputs:

`Token` - NFT token to deposit to the loot pool
`Unlocks` - Time until which the token will remain locked in the pool

Withdraw

Withdraw a previously deposited NFT, which has not been distributed yet.

Requirements: `Pool creator`, `Active deposit claim`
Outputs: `NFT transfer from pool to sender`
Inputs:
`Claim` - Deposit claim hash

Clear

Clear all fungible-token proceeds held by the pool collected as entry amounts by participants by transferring them to the creator.

Requirements: `Pool creator`, `Token balance by pool`
Outputs: `Token transfer from pool to pool creator`
Inputs: `None`

Lottery

Intro

Lottery pools allow the distribution of fungible or non-fungible tokens in a randomized way. Lotteries can be based on specific criteria, such as a minimum token amount that someone must hold to participate or require a minimum amount to enter. After a lottery expires, it can be resolved to automatically distribute the NFT or make the fungible tokens claimable for winning participants.

Use Cases:
Distributing an NFT through a randomized mechanism and a single winner
Distributing fungible-tokens through a randomized mechanism

Functionality

A lottery pool can promote a fungible or single non-fungible token to participants, who can enter to win with a probability dependent on the number of entries. After creating the lottery pool, the creator must deposit the fungible tokens or a single NFT. Once the deposit is complete, any user can participate, either for free or the minimum amount required by the pool. Furthermore, pools can set a minimum amount of XETA that a user needs to hold to be eligible to participate. Requiring a minimum amount of XETA to participate improves incentives and prevents dishonest entries (especially if the lottery pool is sponsored).

After the lottery pool expires, the resolution call automatically transfers the NFT to a single winner chosen from all participants. If the lottery is promoting a fungible token, winners (based on the pools' probability) can claim their fungible-token rewards using their claimable ticket.

Methods

Transfer

Transfer the entry amount to participate in the lottery.

Requirements: `Token balance`, `XETA balance`
Outputs: `XETA transfer from sender to pool`, `Transfer claim from pool to sender`
Inputs: `None`

Claim

Claim a lottery ticket, and receive the pool token with a certain probability.

Requirements: `Active transfer claim`
Outputs: `NFT or token with a probability as specified by the pool`
Inputs:
`Claim` - Claim hash

Resolve

Resolve a lottery pool and determine the winner of an NFT lottery.

Requirements: `NFT lottery`
Outputs: `NFT transfer from pool to random participant`
Inputs: `None`

Create

Create a lottery pool for fungible tokens or a single non-fungible token.

Requirements: `None`
Outputs: `Lottery pool`
Inputs:
`Token` - The token which is to be promoted
`MinAmount` - Participation amount
`TokenTarget` - Amount of pool tokens that someone must hold to participate
`XetaTarget` - Amount of XETA, that someone must hold to participate
`TransfersLimit` - Maximum amount of participants
`ClaimsLimit` - Maximum number of winners
`Expires` - Datetime when lottery naturally expires

Deposit

Allows the creator to deposit the pool token.

Requirements: `Pool creator`
Outputs: `Token transfer from sender to pool`, `Deposit claim from pool to sender`
Inputs:
`Amount` - Amount in case of a fungible-token lottery
`Unlocks` - Datetime when the claim unlocks

Withdraw

Allows the creator to withdraw a previously deposited pool token.

Requirements: `Pool creator`, `Active deposit claim`
Outputs: `Token transfer from pool to sender`
Inputs:
`Claim` - Deposit claim hash

Close

The creator can close the lottery before expiration. If there are participants, the creator can only withdraw the unclaimable share.

Requirements: `Pool creator`
Outputs: `None`
Inputs: `None`

Clear

Allows the creator to clear XETA earnings collected by the lottery pool if the participation amount is positive.

Requirements: `Pool creator`, `XETA balance by pool`
Outputs: `XETA transfer from pool to pool creator`
Inputs: `None`

Royalty

Intro

Royalty pools allow NFT creators to allocate a particular token for royalty payouts of NFT holders. Anyone who owns an NFT specified by a royalty pool can claim royalties based on the holding duration.

Use Cases:
Reward NFT holders based on the length of holding a certain NFT Pay a passive income/royalties for anything in the physical or virtual reality, represented by an NFT

Functionality

A creator creates a royalty pool and specifies candidates and a linear pay-out rate of rewards. After depositing rewards, any owner of NFTs created by the creator (and as determined by the royalty pool) can then periodically claim royalties from the pool. Royalties accumulate every day until a claim is made (claims reset the daily counter).

Methods

Claim

Claim royalties for an NFT created by the royalty-pool creator, depending on the length of holding the NFT.

Requirements: `Claim answer in pool candidates, or no candidates`
Outputs: `Royalty transfer from pool to sender`
Inputs:
`Token` - Token address

Create

Create a royalty pool while specifying the eligible NFT candidates and a daily royalty rate.

Requirements: `None`
Outputs: `Royalty pool`
Inputs:
`Token` - A fungible token that is used as reward payout
`Candidates` - A list of NFT candidates, or empty if all NFTs by the creator should be eligible
`Rate` - Amount of pool-tokens that is paid for every day of holding the NFT
`Expires` - Datetime when the royalty pool is no longer valid

Deposit

Deposit a certain amount of pool-tokens available as reward payouts for NFT holders.

Requirements: `Pool creator`
Outputs: `Reward token transfer from sender to pool`, `Deposit claim from pool to sender`
Inputs:
`Amount` - Reward amount to be deposited
`Unlocks` - Datetime when the deposit claim can be withdrawn

Withdraw

Withdraw previously deposited reward tokens (partially if rewards have been claimed already).

Requirements: `Pool creator`, `Active deposit claim`, `Token balance by pool`
Outputs: `Token transfer from pool to sender`
Inputs:
`Claim` - Deposit claim hash

Close

The creator can close a royalty pool at any time, making it impossible to claim additional royalties.

Requirements: `Pool creator`
Outputs: `None`
Inputs: `None`

Staking

Intro

Staking pools allow creators to reward holders for locking (staking) their tokens for a definite period. After the period passes, the participant can unlock their staked tokens and receive an additional reward from the deposited rewards, based on the specified yearly interest rate and yearly interest bonus.

Use Case: Rewarding holders of a fungible token with a certain APY and bonus rate for committing their holding by locking tokens

Functionality

A creator creates a staking pool with a specific yearly APY rate and an optional bonus percentage. Optionally, a minimum/maximum amount and locking time may be set. After the pool is created, the creator deposits tokens as reward payouts. Any holder of the pool token can lock and commit their tokens for a reward payout based on staking time and APY, and bonus rate. Once the staking period passes, the participant can unlock their tokens and receive their initial stake, including the interest rate as defined by the pool.

Methods

Transfer

Stake tokens by locking them for a specific time.

Requirements: `Min/max amount valid`, `Min/max time valid`
Outputs: `Token transfer from sender to pool`, `Staking claim from pool to sender`
Inputs:
`Amount` - Staking amount
`Unlocks` - Datetime until which the amount will be locked

Claim

Claim back the initial stake, including rewards accumulated during the staking period.

Requirements: `Active staking claim`
Outputs: `Token transfer from pool to sender (with rewards)`
Inputs:
`Claim` - Claim hash

Withdraw

Withdraws reward tokens as creator or deposited tokens as a participant (but without rewards).

Requirements: `Active staking claim`
Outputs: `Token transfer from pool to sender (without rewards)`
Inputs:
`Claim` - Deposit claim hash
`Percentage` - Percentage to withdraw (defaults to 1 equivalent to 100%)

Create

Create a staking pool with a specific APY rate and bonus percentage.

Requirements: `None`
Outputs: `Staking pool`
Inputs:
`Token` - Pool token to be staked and paid as rewards
`Rate` - The yearly APY rate
`Percentage` - A bonus percentage, paid additional to the annual APY rate for one year of holding
`MinAmount` - Minimum staking amount
`MaxAmount` - Maximum staking amount
`MinTime` - Minimum staking time in seconds
`MaxTime` - Maximum staking time in seconds
`TransfersLimit` - Maximum number of transfers to the staking pool
`Expires` - Datetime when the staking pool expires and is not accepting new participants

Deposit

Deposit reward tokens to the staking pool.

Requirements: `Pool creator`
Outputs: `Token transfer from sender to pool`
Inputs:
`Amount` - Amount to be deposited as rewards
`Unlocks` - Lock time of the reward token claim

Swap

Intro

Swap pools are automatically created for every fungible token that is minted. These allow participants to swap tokens against XETA and vice versa in a decentralized way. The price is based on the liquidity that token creators and participants can contribute, and the trades that affect the liquidity.

Use Case: Decentralized exchange mechanism for fungible tokens

Functionality

A swap pool is automatically created with the creation of every fungible token. Creators can add liquidity via a launch pool, or directly by depositing the pool tokens and XETA. The amount of deposited liquidity determines the price, which is determined by dividing the number of tokens available as liquidity by the amount of XETA available as liquidity. Participants can swap XETA for tokens or tokens for XETA. Transfers in and out are entirely handled by the program in an automated way, charging 0.25% commission on every trade and automatically adding 1% of the trade value to the pools' liquidity.

Methods

Transfer

Swap tokens into XETA or XETA into tokens at the current exchange rate.

Requirements: `Pool liquidity`
Outputs: `Transfer from sender to pool`, `Transfer from pool to sender`
Inputs:
`Token` - XETA or the pool-token, to use as input token
`Amount` - Number of tokens to swap

Deposit

Deposit XETA and tokens to liquidity at a specific rate.

Requirements: `XETA balance`, `Token balance`
Outputs: `XETA transfer from sender to pool`, `Token transfer from sender to pool`, `Liquidity deposit claim from pool to sender`
Inputs:
`TokenAmount` - Amount of tokens to deposit
`XetaAmount` - Amount of XETA to deposit
`Unlocks` - Datetime when the deposit claim shall unlock

Withdraw

Allows withdrawing previously deposited liquidity tokens adjusted for the new liquidity rate.

Requirements: `Active liquidity deposit claim`
Outputs: `XETA transfer from pool to sender`, `Token transfer from pool to sender`
Inputs:
`Claim` - Deposit claim hash
`Percentage` - Percentage between 0-1 to withdraw

Vote

Intro

The voting pool allows participants to vote on a numeric outcome (such as bets, estimations, budgets, prediction markets) or on an outcome based on provided answers. Voting pools can require a contribution amount, which serves as a weight for the vote. Vote pools can be resolved using various mechanisms: transferring all collected tokens to the creator, the highest candidate, or the voters (by the highest voted answer or through oracle result).

Use Cases:
Voting on a numeric outcome (bets, estimations, budgets, prediction markets)
Voting on an answer outcome (guesses, candidates, tokens, proposals)
Voting by collecting funds and rewarding the vote pool creator
Voting by collecting funds and rewarding the highest voted candidate
Voting by collecting funds and rewarding voters of the highest voted answer
Voting by collecting funds and rewarding voters of the correct answer, as determined by an oracle

Functionality

A creator creates a voting pool and specifies the participation token, resolution mechanism, and optional candidates. Optionally, a minimum/maximum participation amount can be specified. Voters can then vote by providing a numerical answer (if the voting pool is a numerical vote) or by providing one of the specified answer candidates. Every vote can be weighted by the XETA participation amount. After the vote expires, it can be resolved based on the specified resolution mechanism. The mechanism rewards the creator, candidates or voters of the highest voted answer or the correct answer as determined by an oracle. If the mechanism rewards voters of the highest or correct answer, they can claim their rewards proportional to their weighted vote.

Methods

Transfer

Participate in the voting pool by providing a numerical answer or a candidate from the available choices.

Requirements: `None`
Outputs: `Transfer claim from pool to sender`
Inputs:
`Amount` - XETA amount to add weight to the vote
`Answer` - Choice from available candidates
`Number` - Number if the vote is numerical

Claim

Allows claiming rewards, proportional to the weighted vote divided by the total weight of the winning vote.

Requirements: `Claim with valid answer`
Outputs: `Reward transfer from pool to sender`
Inputs:
`Claim` - Claim hash

Resolve

Resolves vote using the specified vote-pool mechanism. Pays collected XETA weight amounts to either creator, top candidate or enables voters of the highest answer to claim their rewards.

Requirements: `Vote mechanism != oracle`
Outputs: `Rewards transfer from pool to creator/candidate`
Inputs: `None`

Create

Creates a voting pool, with a numerical or candidate type and a specific resolution mechanism.

Requirements: `None`
Outputs: `Vote pool`
Inputs:
`Mechanism` - Resolution mechanism, creator, candidate, voters, oracle
`Candidates`- List of candidates to vote on, numerical vote if empty
`MinAmount` - Minimum amount to participate in vote
`MaxAmount` - Maximum amount to participate in vote
`TransfersLimit` - Maximum number of votes
`ClaimsLimit` - Maximum number of winning votes for top candidate
`XetaLimit` - Maximum collected XETA as vote-weights
`Expires` - Datetime when the vote expires

Oracle

The creator can set the correct answer so voters who claimed that answer can claim their rewards.

Requirements: `Pool creator`, `Candidate pool`, `Pool mechanism = oracle`
Outputs: `None`
Inputs:
`Answer` - Correct answer

Bridge

Intro

The Xeta bridge swaps between ETH/BNB/BSC XETA and native XETA tokens, with slippage of around 1.5% (plus regular liquidity slippage for amounts that impact liquidity). This process also works vice versa. The swap process takes about 5 minutes and is completely automated in both directions.

Use Cases:
Swapping supported assets from Ethereum (ETH) or Binance Smart Chain (BNB, XETA) to Xeta native tokens, Swapping native Xeta tokens into assets on Ethereum (ETH) or Binance Smart Chain (BNB, XETA)

Functionality

The user makes an input transfer to Xeta's bridge wallet address on the chosen input chain. Xeta listens to all incoming transactions with a specifically formatted schema (the data/message field of transactions) and awaits a sufficient number of confirmations. After the transaction is confirmed, Xeta calculates the exchange rate based on swap size and its effect on liquidity to account for possible slippage. Slippage fees added on top of the base fee of 1.25%. Xeta then sends the output transfer on the chosen target chain to the wallet address connected by the user.

Methods

Create Bridge Request

Create a new bridge request.

Requirements: `ETH/BSC wallet`, `XETA wallet`
Outputs: `Transfer on target chain`
Inputs:
`ETH/BSC public address`
`XETA public address`
`Swap amount`
`Source-chain`
`Target-chain`

Faucet

Intro

Xeta's Faucet allows users to request a certain number of free XETA tokens on the Mainnet and Testnet. It allows users and creators to try out Xeta without purchasing or swapping tokens.

Use Cases:
Create a few simple transactions on Mainnet
Create, test, and launch any desired functionalities on Testnet

Functionality

Users who own a Gmail address can request free XETA tokens from the faucet feature. The user needs to confirm their email, after which the Faucet automatically makes two transfers to the users' connected wallet, one on Mainnet and one on Testnet.

Methods

Create Faucet Request

Create a faucet request and receive tokens on Mainnet and Testnet.

Requirements: `Valid Gmail address`, `Xeta wallet`
Outputs: `XETA transfer from faucet to wallet on Mainnet`, `XETA transfer to from faucet wallet on Testnet`
Inputs: `None`

Sponsorship

Intro

Xeta enables users to transfer a sponsored balance to addresses or pools, so other users interacting with these resources can do so without paying network fees.

Use Cases:
Recipient sponsoring incoming transfers to their address
Pool sponsoring incoming transfers and enabling participation without owning XETA
Project creators who would like to encourage participation in particular pools

Functionality

A user transfers sponsored XETA tokens to an address or pool. The transferred token is specified as sponsored XETA and is automatically swapped from the users' XETA balance into sponsored XETA and credited to the resource. Other users who interact with the sponsored resource won't be charged any fees until the sponsored balance of the resource is depleted.

Methods

Create

Swaps the users' owned XETA into sponsored XETA. Sponsored XETA cannot be withdrawn.

Requirements: `XETA balance`
Outputs: `Sponsored XETA transfer from sender to recipient`
Inputs:
`To` - Recipient resource to be sponsored
`Token` - Must be specified as 111111111111111111111111sponsored
`Amount` - Amount to sponsor
`From` - From address in case of allowance
`Message` - Custom transfer message

Wallet

Intro

Xeta's wallet allows users to create and manage their XETA blockchain wallets, either by managing keys yourself or letting a 3rd party provider (currently Xeta foundation) manage keys for you.

Use Cases:
Create a new self-managed or managed Xeta wallet
Connect an existing self-managed or managed Xeta wallet

Functionality

Users need a wallet to interact with the Xeta blockchain. Users can create a key pair (public key/private key) or let a provider manage this keypair for them (using a more memorable account and secret). Once a wallet is created through the self-managed or managed option, it can be connected to Xeta's apps, as long as the owner has access to the private key or account and secret (in the case of managed keys).

Methods

Create Self-Managed

Create a new XETA blockchain key pair.

Requirements: `None`
Outputs: `Public key`, `Private key`
Inputs: `None`

Create Managed

Create a managed wallet.

Requirements: `None`
Outputs: `Public key`
Inputs:
`Account` - Memorable account name, for example, email, phone number, or username
`Secret` - Memorable password

Connect Self-Managed

Connect a self-managed wallet.

Requirements: `Private key`
Outputs: `None`
Inputs: `None`

Connect Managed

Connect a managed wallet.

Requirements: `Account name`, `Account secret`
Outputs: `None`
Inputs: `None`