DXBL Technical Paper

Tracing how the Proof of Trade protocol works through its Smart Contract architecture.

Proof of Trade (aka "PoT") aims to promote awareness and use of the Dexible **Smart Order Routing engine**. The Dexible team invented **Proof of Trade** to promote awareness and increase engagement and retention for the Dexible platform.

To recap the DXBL Litepaper, Proof of Trade mints new tokens as a reward for volume passed through the protocol. Only successful transactions will prove trading activity occurred and be counted towards minting. Trading is but one part of a strategy, and swaps are a fundamental primitive at the basis of all DeFi strategies. At the same time, Dexible enables use cases not available elsewhere it engineered over 2+ years of development. So with Proof of Trade in Dexible version 2.0, it is likely traders will weigh the incentives for proving trading volume with Dexible's set-and-forget capabilities that offer fundamental value for risk mitigation and profit taking.

This **Technical Paper** provides a comprehensive overview of the underlying **Smart Contract Architecture** and logic that powers the Proof of Trade protocol.

The **source code** for the relevant contracts can be accessed through the following GitHub repository: https://github.com/BUIDLHub/dexible-pot

Smart Contract Architecture

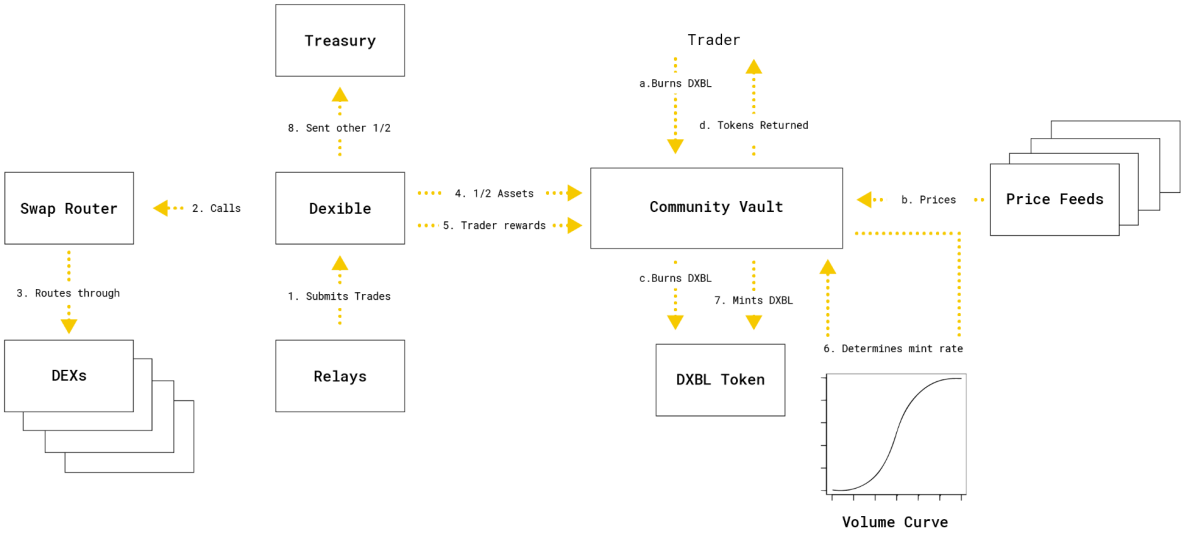
What are the various contracts designed for Trading and Redeeming tokens?

The Dexible team engineered the **Proof of Trade** protocol to incentivize positive feedback loops through deeper engagement with the Dexible platform's functionality.

This protocol rewards traders with a rebate token tied to the volume of trades and a speculative market on the potential for increasing or decreasing trade volume. The rebate token also serves as a discount mechanism for protocol fees.
There are two primary interactions associated with the protocol:

1. Executing trades.
2. Redeeming tokens for rewards.

The flow of these interactions is illustrated in the diagram below. The diagram outlines the various stages of the **Trading Process** and the steps involved in **Token Redemption**.
The following overview serves as a foundation for a more in-depth examination of each contract and its requirements, which will be addressed in subsequent sections.



DEXIBLE

v2.0 Proof of Trade Protocol

Trading Flow

1. Trader or Dexible relay submits a swap request to the protocol
2. Dexible contract uses a **Swap Router** contract to execute instructions in the request
3. The swap router calls one or more DEXs to execute the swap.
4. After computing fees, 50% of the bps revenue is sent to the **Community Vault**.
5. Dexible requests the vault reward the trader with tokens.

6. The vault uses the **current mint rate** against traded volume to compute the number of tokens to reward.
7. The vault requests $DXBL to mint new tokens to the trader (since it has the **"minter" role** on the **$DXBL token contract**).
8. The other 50% of bps revenue + gas is sent to the **Treasury**.

Redemption Flow

A. Trader initiates the burn function through the Community Vault. They specify the amount of $DXBL to redeem, their desired vault reward token (one of the **Approved Vault Tokens**), and their minimum expected amount (in the form of a **Slippage Tolerance**).

B. The pool determines **Net Asset Value (NAV)** using the current **Assets Under Management** USD values from Approved Vault Token price feeds powered by Chainlink oracles, and then computes how many reward tokens to issue to the trader.

C. The Community Vault contract makes a burn request to the $DXBL token contract, since it has "minter" role.

D. The Community Vault sends the reward tokens to the trader.

Note: Using "minimum expected amount" in the Redemption Flow exists because other traders may be burning their $DXBL in the same block, causing the amount of the specific vault assets available to deplete (example: other traders want to withdraw USDC until none is left in the vault). If the expected USDC to redeem drops so much that the amount doesn't meet the trader's expected minimum output, the transaction will fail. The trader can switch their selected reward token or wait until the vault recovers fee tokens to withdraw.

The next sections will dive into each contract in more detail, how the protocol is implemented, and how requirements are satisfied.

Design Principles

Requirements that specify how the protocol can operate.

The success of the Dexible **Proof of Trade (aka "PoT")** protocol requires a set of well-defined requirements for its smart contracts. These requirements are categorized as follows:

Security

To ensure the security and stability of the upgradeable smart contracts, the following measures must be implemented:

- All PoT protocol contracts MUST NOT be upgradeable or changeable by a **single EOA wallet**.
- All PoT protocol contracts MUST enforce a **grace period** for changes to take effect. This allows the community to review the changes being made.
- All PoT protocol contracts, at a minimum, **MUST be reviewed** by community members to uncover gross oversights or mistakes. Ideally, an audit would be performed on all contracts as budgets allow.
- All PoT protocol contracts, except the token contract, **MUST be pausable** in case of emergencies.

Trading

Previous Dexible contracts were known as **"Settlement" contracts**, accessible only through Dexible's **"relay" wallets**. This hybrid solution offered on-chain settlement for privately managed trades. That trading information was held off-chain, thereby concealing the intention of the traders. Dexible had a secure process for handling sensitive information and preventing alpha from leaking, including fee computations. The drawback of this approach meant there could be no public direct interaction with the protocol. The hybrid approach also resulted in inaccurate gas estimates, leading to uneven fee earnings and the trader paying more in expected gas than they would come time of execution.

To mitigate these challenges, the following measures will be implemented in Dexible version 2.0:

- The Dexible contract MUST allow **public access** so anyone can submit trades to mine $DXBL tokens.
- The Dexible contract MUST also support **restricted "relay" submissions** containing affiliate and other sensitive information.

Gas Fees

The oldest approach to fee management, developed in the Dexible Alpha in January 2021, resulted in serious friction. The Alpha used a **Gas Tank** model, where every trader would need to predeposit assets for fees. At the same time, accurate fee

estimations were challenging. This resulted in thousands of dollars sitting idly in a Gas Tank contract until the trader would submit orders.

The Settlement contract was adjusted in Dexible version 1.0 to rely on fees from the swap pair. To do this, Dexible selected the token from the swap pair with the highest liquidity to draw fees from and apply the bps fee. However, this method resulted in higher on-chain computation costs. Moreover, it relied on the accuracy of asset pricing from 3rd parties to validate the appropriate fee.

To improve this process, the following requirements will be implemented in Dexible version 2.0:

- The PoT protocol contracts MUST use **on-chain pricing** to convert gas fees to fee-token units.
- Because of the previous requirement, the contracts MUST **restrict vault tokens** to only those with an on-chain price feed.

Community Vault

The Community Vault is an integral component for ensuring effective management of the supply and price of the $DXBL token. To fulfill this objective, the following requirements must be met:

- The Vault MUST maintain an accurate 24-hour rolling window of trade volume
- The Vault MUST compute the USD value of all Approved Vault Token assets (AUM)
- The Vault MUST compute the current NAV of each $DXBL using AUM / $DXBL supply
- The Vault MUST allow $DXBL owners to burn tokens for equivalent vault tokens based on current NAV and vault-token price
- The Vault MUST reward traders with $DXBL tokens according to the Volume Curve outlined in the DXBL litepaper and only allow these requests from the Dexible contract
- The Vault MUST adjust the minting rate according to 24hr trade volume

Token

As a **standard ERC20 token**, many of the requirements have already been established by the Ethereum community. However, the $DXBL token has unique characteristics that must be taken into consideration, as follows:

- The token contract MUST allow for minting and burning, but only from the Community Vault contract
- The token MUST maintain and apply a **discount rate per token owned**
- The token SHOULD compute and adjust fees according to a trader's balance

It's not critical for the token to compute fees. It's an additional feature that has already been implemented but is not required. The token operates just fine without a function to compute fees. It's a convenient function for traders to apply token-balance discounts to fees.

Change Process

What is the process for upgrade Proof of Trade contracts?

The protocol changes review process has been implemented to promote transparency and community involvement in the decision-making process. The **two-phase review process** will ensure that all changes are thoroughly reviewed and approved before being implemented in the protocol. In this context, “change” refers to migrating the CommunityVault to a new version or forking the Dexible settlement contract with a completely new version. No upgrades are possible on these contracts and any new versions must be reviewed prior to deploying new contracts.

“Migrating” the CommunityVault means creating a new contract, likely with new features and settings, and asking the current “live” vault to migrate its information and token balances to the new version.

Review Phase 1

The process begins with submitting a Pull Request (PR) on the Proof of Trade smart contract repository, which will be announced on Dexible's official channels. This allows the community to review the proposed changes and provide feedback or suggestions. The first review phase will last seven days. Any changes made during this period will extend the **review period** to ensure that all relevant parties have adequate time to review the proposed modifications.

Review Phase 2

In the second review phase, the Dexible team will submit an on-chain request for change utilizing a **multisig mechanism**. This submission will initiate a **7-day grace period**, which will serve as a final opportunity for the community to review the

proposed changes before they are applied. The grace period is designed to ensure that those who do not follow the official channels but monitor the contract for change events have a final opportunity to review the proposed changes.
Once the grace period has ended, the Dexible team will submit the final approvals through a Gnosis Safe MultiSig to make the changes official. This mechanism ensures that all changes are thoroughly reviewed and approved by multiple parties before being implemented on the protocol, ensuring the security and stability of the network.

Multisig Approach

What does it mean to have a Multisig in practice?

Dexible will use a Gnosis Safe contract on each network it operates on to manage changes. Some changes require a grace period (like new CommunityVault migrations) while others only require a multi-party signature (config changes). In all cases, the community review process will be used to ensure everyone understands how things are changing. The Gnosis Safe ensures that no one wallet is responsible for changes to the protocol and in the case of a signing wallet compromise, Gnosis helps mitigate bad actors by allowing other signers to remove them as a signer before they wreak havoc on the Safe and thus the protocol.

Configuration

Most of the Proof of Trade protocol contracts have settings that can be changed using the previously discussed approval process. The settings and their default values are also outlined.

Dexible Contract

The Dexible contract has multisig settings plus the following items:

Setting	Description	Initial Value

revshareSplitRatio	The amount of bps fee going to the Community Vault expressed as whole-number percentage	50
stdBpsRate	The standard BPS fee applied to traded volume	8
minBpsRate	The minimum fee applied to any trade regardless of discounts	4
communityVault	The Community Vault contract address	<set at deployment>
treasury	The Dexible Treasury address where other revshare split and gas fees go	<set at deployment>
dxblToken	Address of the DXBL token contract	<set at deployment>
adminMultiSig	Address of the Gnosis Safe that will administer the contract	<set at deployment>
minFeeUSD	Potential minimum fee per txn to mitigate infra costs exceeding revenue per order.	0
initialRelays	Whitelisted wallets that can submit orders on traders behalf	<set at deployment>

Community Vault

The pool contract has all of the multisig settings plus:

Setting	Description	Initial Value
wrappedNativeToken	The address of the wrapped native token for the network (WETH, WAVAX, etc)	<set at deployment>
baseMintThreshold	The starting mint rate	$100
rateRanges	An array of mint rate range ranges (in millions) and their increased % per million in volume	See litepaper for details
feeTokenConfig	A set of tokens and their chainlink price feed addresses	See litepaper for details
discountRateBps	Applied on the DXBL token contract, which determines how much of a discount to apply per token (expressed in bps)	5
dxblToken	Sets the DXBL token address	<set after deploying DXBL token>
dexible	Dexible contract address since it is the only one allowed to request DXBL rewards for traders	<set after Dexible deployed>
timelockSeconds	Grace period to apply migrations expressed in seconds	7-days

adminMultiSig	The Gnosis Safe used to administer the contract.	<set at deployment>

DXBL Token

These settings are not directly changeable but are set at deployment. The only one that can change is the rewards per token setting, which is adjusted via the Community Pool contract.

Setting	Description	Initial Value
discountRateBps	The amount of discount per token owned expressed in bps	5
minter	The contract address that can make mint/burn requests	<set to Community Pool address at deployment>

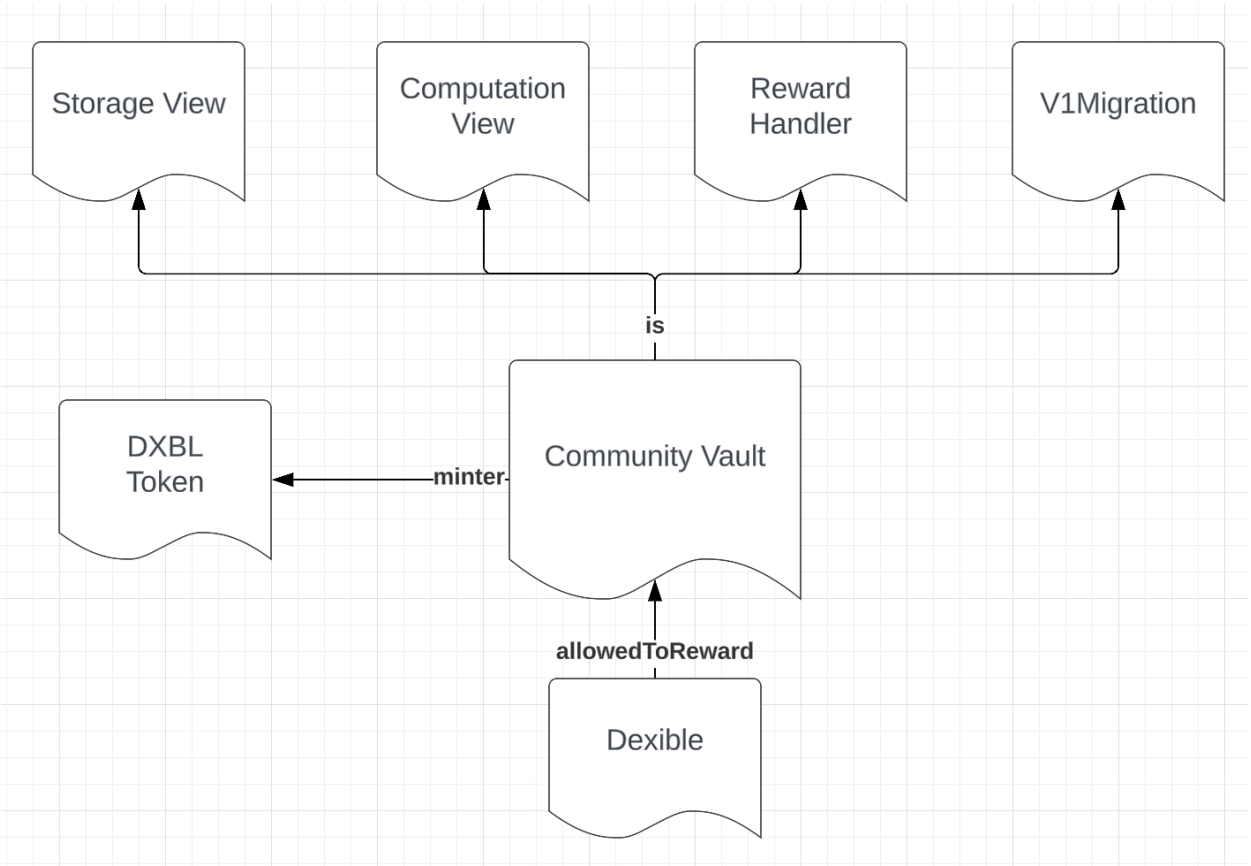
Community Vault Contract

What rules and logic guide the Community Vault

The **Community Vault** oversees the administration of **Approved Vault Token assets** and the supply of DXBL tokens. Following the Proof of Trade ("PoT") protocol contracts, the Design Principles outlined on that page must be upheld.

See the bullet points under the headers titled Security, Trading, Gas Fees, and Community Vault.

This diagram outlines the relationship between several contracts.



At the center is the **CommunityVault**. This hosts the primary configuration, redemption, and pausing features of the vault. It extends several base contracts that focus on accessing stored properties, computing something based on stored properties, rewarding traders, and migrating to new versions of the vault.

StorageView provides access to the vault's configuration and storage properties. It's a read-only contract.

ComputationView uses stored information to compute things like AUM, mint rates, redemption estimates, etc. It too is a read-only contract.

RewardHandler handles rewarding traders with tokens and updating volume metrics to adjust mint rates, etc.

V1Migration is a mechanism used to migrate the vault to a new version if the protocol and community decide a transition is needed.

The following table outlines how each requirement is satisfied by these contracts.

Requirement	Satisfied By	Description
1,2,4	Gnosis Safe	All approvals for changes are managed by multi sig Safe
5-8,10	ComputationView and CommunityVault	Logic to manage asset values and mint rates
9	Dexible/Vault	Dexible is allowed to request rewards for traders and the pool computes the rewards
3	None	This requires general community involvement and audits

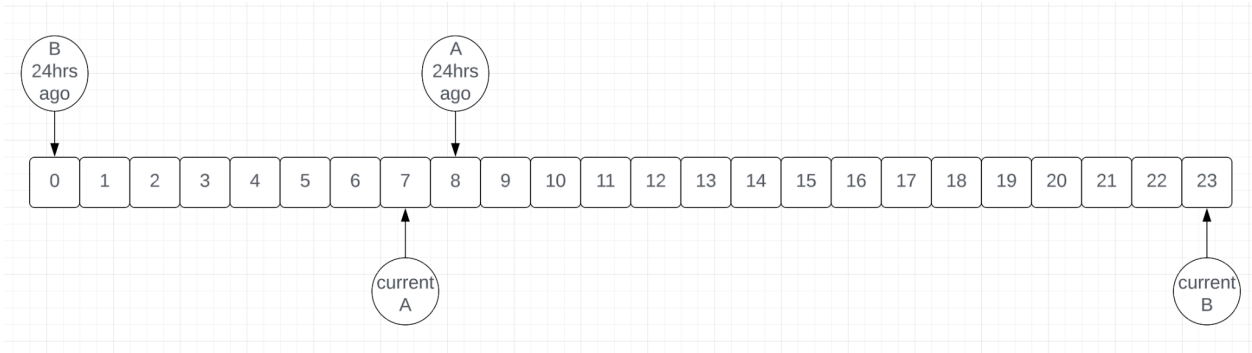
To maintain the **24-hour volume window**, the contract logic works like this:

- Hourly buckets of volume totals are maintained
- On each trade, the USD value of the trade's approved vault token is computed using the Chainlink oracle price feeds

A vault token must be the input or output token in the swap

- The current hour's bucket is incremented with the USD value
- The bucket of the previous 24-hour slot is used to reduce the total 24-hour volume and then reset to 0 to avoid multiple deductions

The following image depicts this a little more clearly:

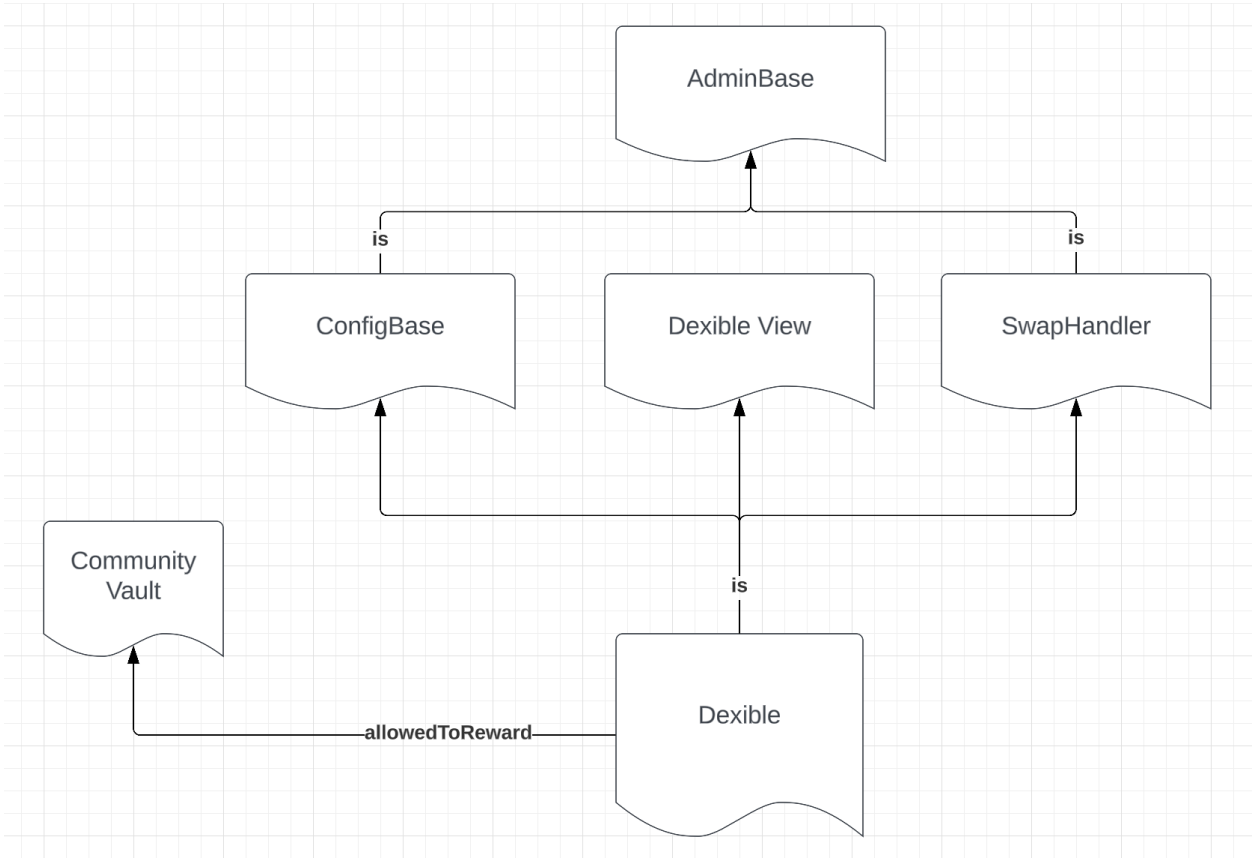


The rolling 24-hour volume window is maintained as a circular list of buckets. 24-hour reductions are simply the next bucket with wraparound logic to account for 24-hour clock.

The Dexible Contract

The Dexible contract is responsible for satisfying execution requests. Today it only supports **Swap requests**. But future versions will open this up to more **execution types** such as lending, staking, and other primitives.

Here is the relationships for the Dexible contract:



Like the Community Vault, there are base contracts that provide configuration, storage property views, and swap functionality for protocol. Dexible is configured with whitelisted relay wallets. The relays are used to enforce the rule that only relays can submit sensitive execution requests where certain fields of the request require more control, like affiliate fees for example. Here is a breakdown of how these contracts satisfy the requirements:

Requirement	Satisfied By	Description
1,2,4	Gnosis Safe	Safe multi sig ensures multiple parties approve changes
5,6	Dexible	Restricts private swap requests to only those addresses with the "relay" role. And Dexible offers both relay and public swap submissions.
7-8	CommunityVault	By leveraging its relationship with the vault, Dexible can ask the vault for current prices of fee tokens to convert the gas fees to fee-token units. Dexible also uses the vault to determine if a fee token is allowed.

DXBL Token Contract

The $DXBL token contract is a standard ERC20 contract with some additional functions for minting and fee discount calculation.

Recall the requirements stated in Design Principles under the headers Security and Token.

The relationships to the $DXBL token contract have already been laid out in previous diagrams. And those relationships are what satisfy the requirements.

Requirement	Satisfied By	Description
2,3	CommunityVault	Pool has minter role on DXBL contract so it can request tokens to be minted and burned
4	DXBL Tokens	Has functions for computing discounts according to its configured discount bps rate.

1	None	Required community involvement and audits

Arbitrum Gas

There is a special case when computing gas costs for the Arbitrum network. In Solidity, the normal mechanisms for computing the amount of gas usage do not yet accurately reflect an amount that accounts for the **L1 batch gas fees**. Future network upgrades are supposed to fix this, but as of writing, the gas usage cannot be estimated using only the gasleft solidity function.

To work around this issue, Dexible deployed an Arbitrum gas price oracle smart contract. This oracle uses the arbOs precompiles and a **multiplier** to determine the actual gas cost for a transaction. The math is pretty simple:

$$(u * p2) + ((m * p1 * s)/1e18)$$

- *u* is the current L2 gas usage estimate (using gasleft() function)
- *p1* is the current L1 gas price from the arbOs precompiles
- *p2* is the L2 gas price for the transaction
- *m* is the multiplier, expressed in 18 decimals
- *s* is the size of the transaction's calldata

The multiplier can and likely will be adjusted according to historical gas estimations and especially if the network nodes start accurately reporting gas usage using gasleft() function. As of this writing, the multiplier is 2e17.

This whole thing is a **stopgap** to protect relays when submitting transactions for traders until nodes accurately report gas usage.