# **Mercata Protocol Guide**

This document provides a detailed overview of the integration process for Strato—a PBFT layer 1 chain. It covers the RST API endpoints, secure OAuth flows via SMD, and smart contract deployment/interaction workflows including specific instructions for asset management and liquidity operations.

## **1. Introduction**

This integration guide is intended to assist users in:

* Querying the blockchain for current state and transaction details.
* Securely interacting with smart contracts via the SMD portal.
* Deploying new contracts

## **2. SMD Portal Login**

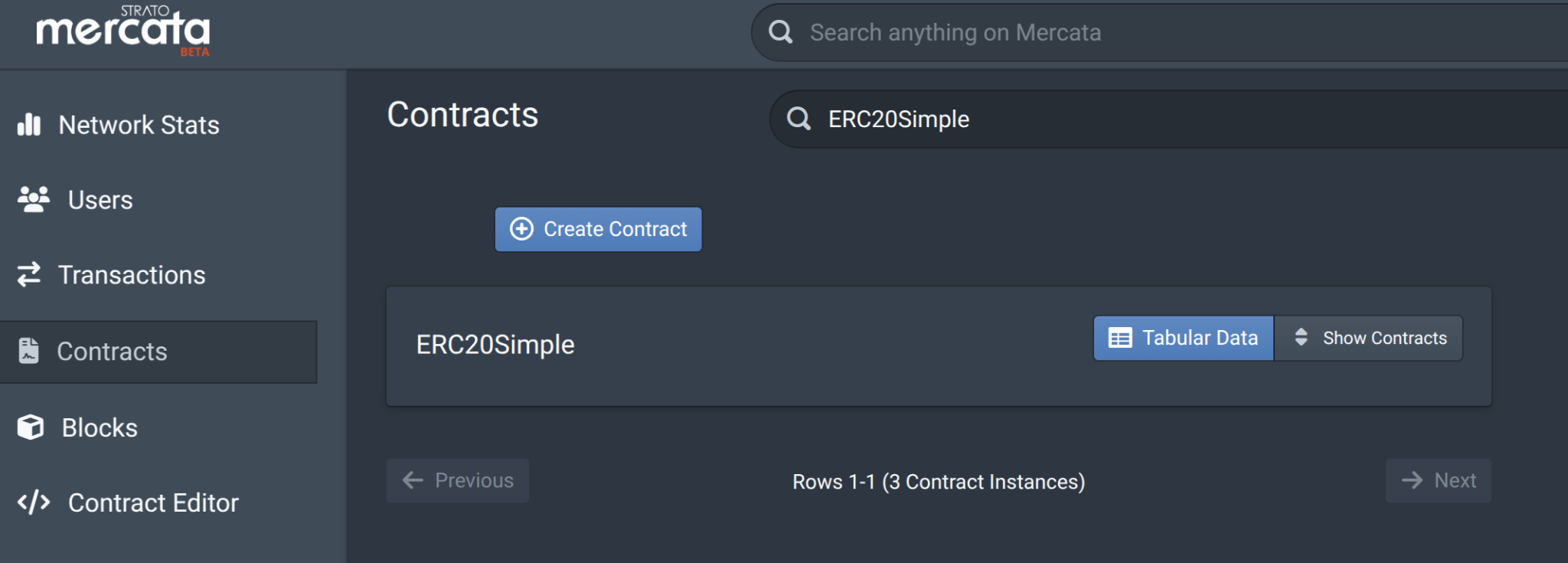
Accessing and interacting with smart contracts requires authentication via OAuth. Here are the key steps:

1. **Login to SMD:**
   * **Portal URL:** [SMD Dashboard](https://node1.mercata-testnet2.blockapps.net/dashboard)
   * **Credentials:** Use your provided username and password.
   * SMD acts as the central portal to create and interact with already deployed contracts.
2. **Getting Started:**
   * **Login Securely:** Access the platform using your username and password.
   * **Manage and View Contracts:** Easily view details of deployed contracts, including balances, transaction history, and metadata.
   * **Interact with Contracts:** Call functions (such as transferring tokens or swapping assets) directly from the user interface.
   * **Deploy New Contracts:** Use SMD to compile and deploy your own smart contracts.

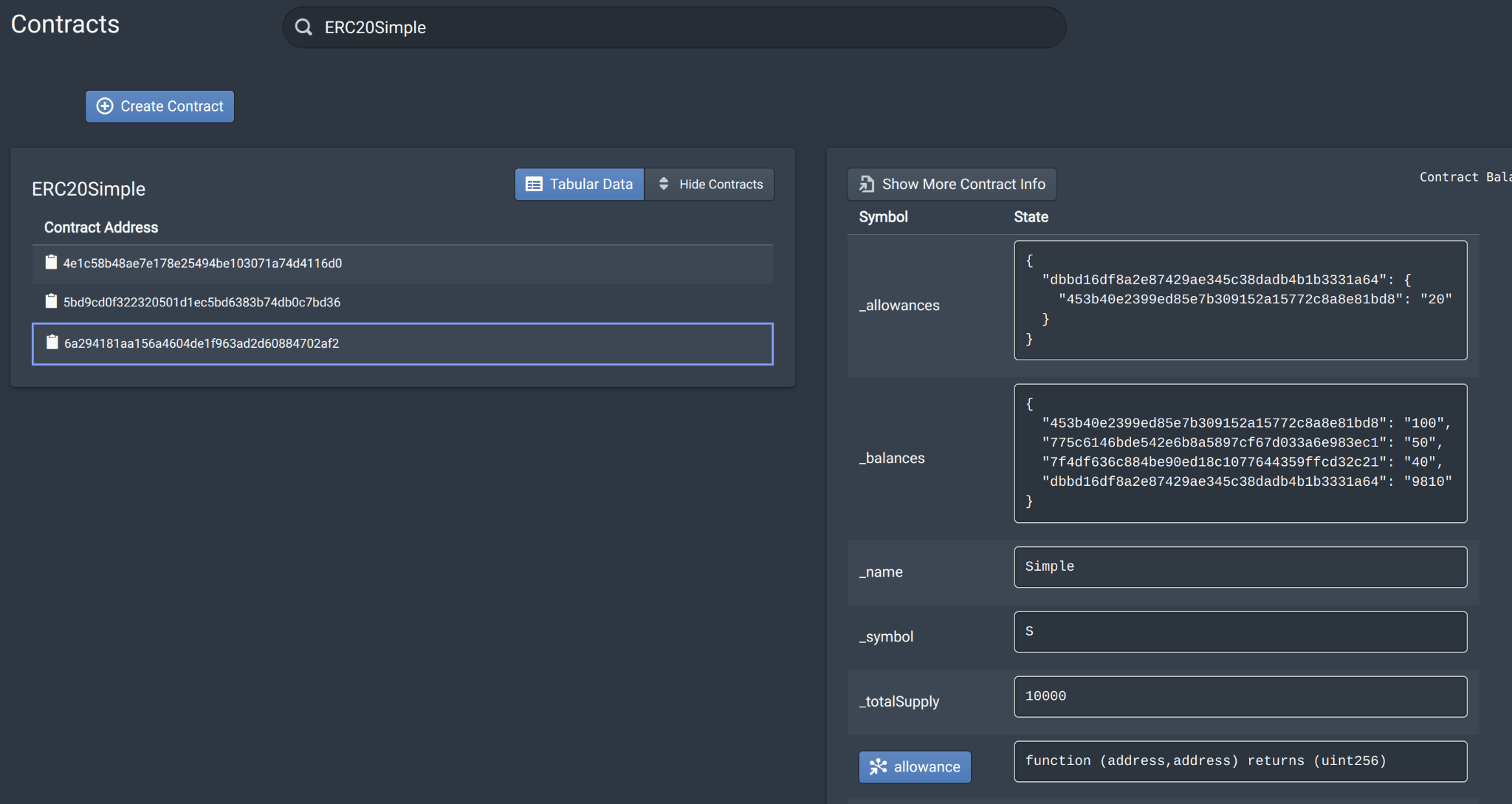
## **3. Interacting with Deployed Contracts on SMD (Calls)**

Once logged in, you can use SMD as your GUI for smart contract interactions. The typical flow is:

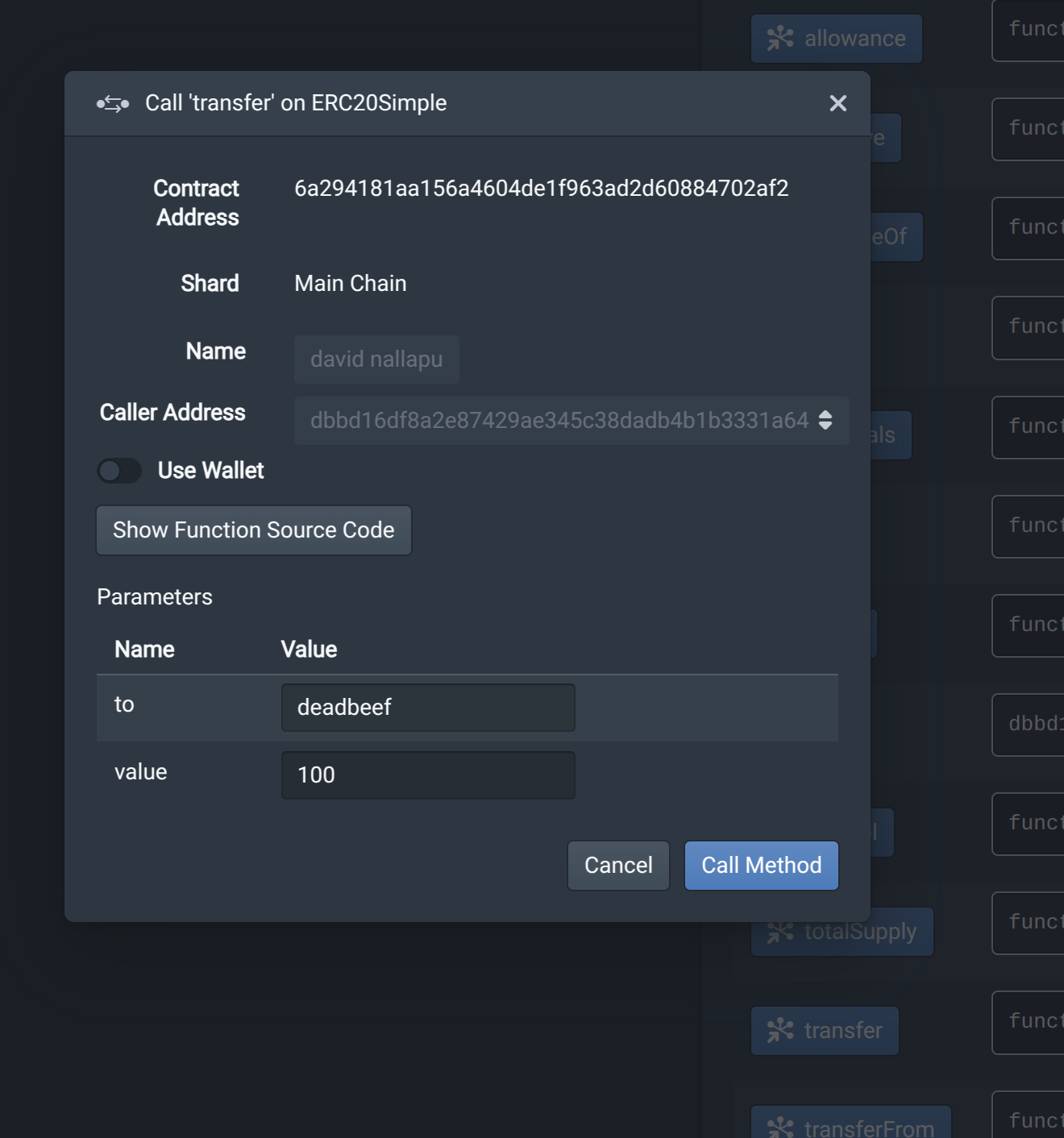
1. **Search & Select Contract:**
   * Enter the contract address or name(for instance, an ERC20 token or a Liquidity Pool contract) in the search field.



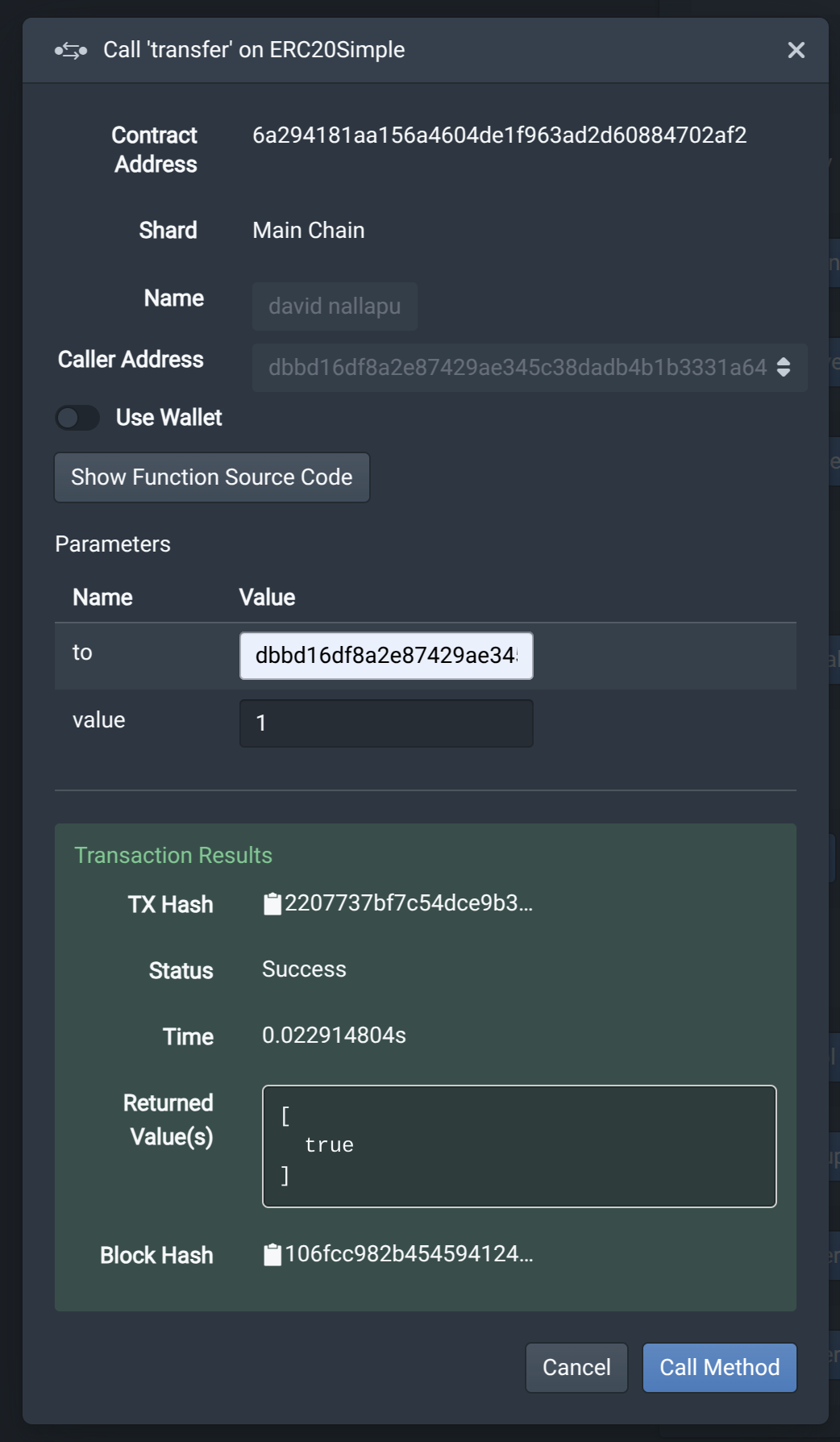
* + Click on the address and then you will have access to detailed views similar to Etherscan—where you can inspect transactions, view metadata, and execute function calls.



1. **Function Calls:**
   * **Token Transfers:** Use the inherent ERC20 or Pool methods.
     1. If you click on any of the functions, a dialog box will open up where you can fill in the params of the function and click on Call method to execute



* + **After Execution:** 
    1. After execution you get the
       1. **TX Hash (Transaction Hash):** A unique identifier for the transaction on the blockchain. It's like a receipt number for the transaction.
       2. **Status:** Indicates whether the transaction was successful or failed.
       3. **Time:** The timestamp of when the transaction was processed and included in a block.
       4. **Returned Value(s):** The output or result of the function call. For example, if you transferred tokens, this might show the amount transferred.
       5. **Block Hash:** The unique identifier of the block that contains the transaction. It links the transaction to a specific block in the blockchain.

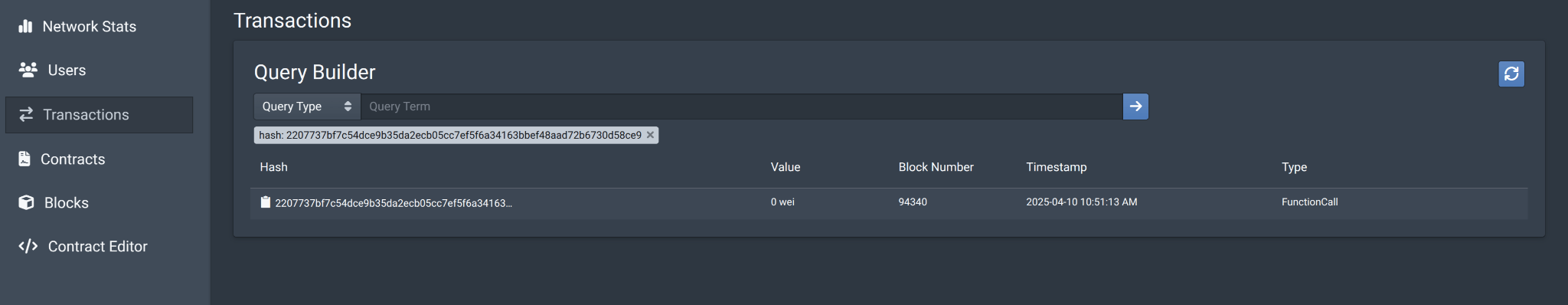
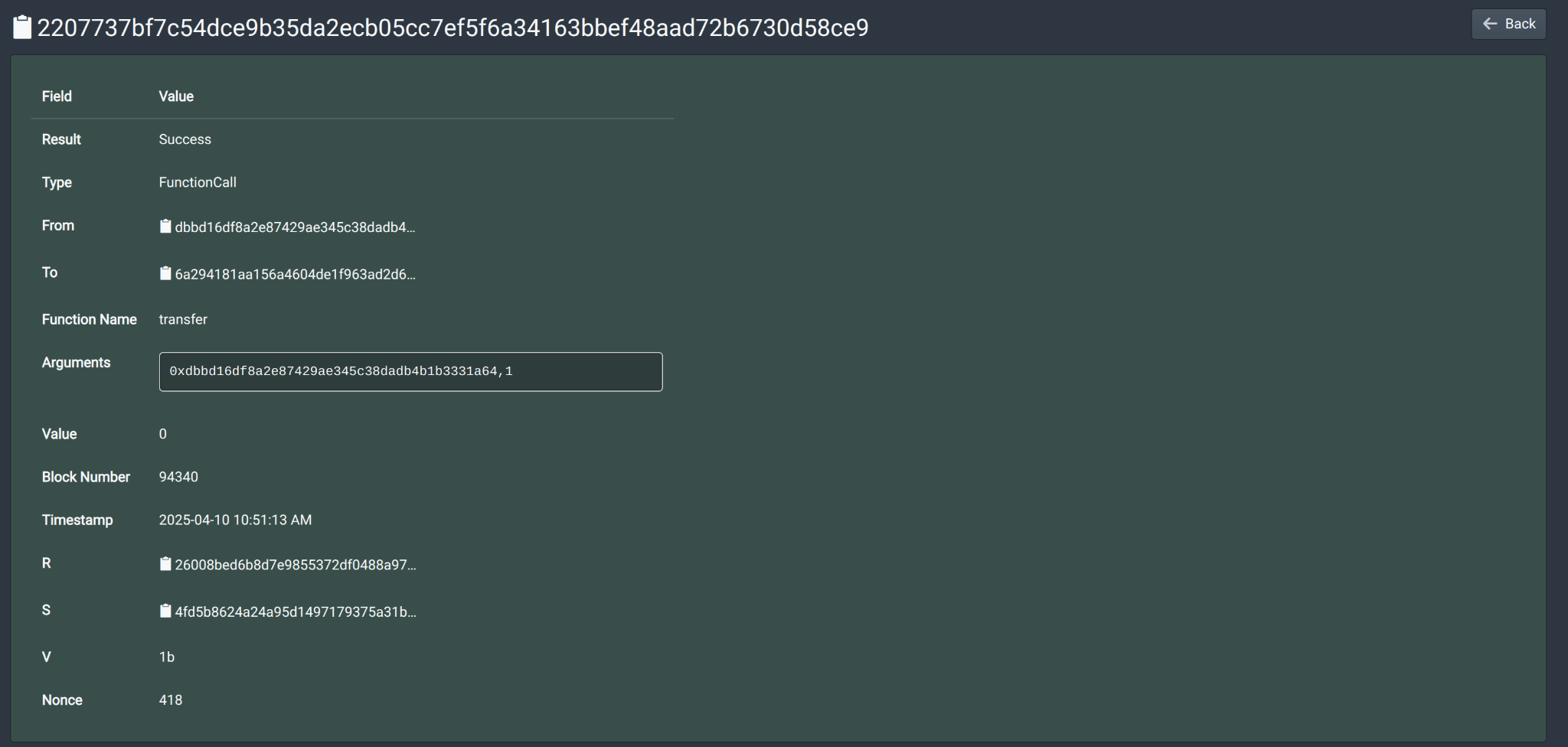


## **4. Querying the results on SMD**

There are three options to see the results:

1. Through the Transactions tab in SMD
2. Through the strato-api REST API endpoint
3. Through the cirrus POSTGREST REST API endpoint

Through the Transactions tab in SMD

* + Click on the Transactions tab
  + You will see a Query Builder Box
  + Choose Hash as the Query Type and add the TX Hash
    - Click on the result and you will see all the details
    - **Note:** You can go directly to this tab by adding the hash here:  
      <https://node1.mercata-testnet2.blockapps.net/dashboard/#/transactions/2207737bf7c54dce9b35da2ecb05cc7ef5f6a34163bbef48aad72b6730d58ce9>

Through the strato-api REST API endpoint:

* <NODE\_URL>/strato-api/eth/v1.2/transactionResult/<TXHASH>  
  + Example: <https://node1.mercata-testnet2.blockapps.net/strato-api/eth/v1.2/transactionResult/2207737bf7c54dce9b35da2ecb05cc7ef5f6a34163bbef48aad72b6730d58ce9>

Through the cirrus POSTGREST REST API endpoint:

* This depends on what you are trying to do and how it affects the contract state. For a transfer, an event is emitted so you would see the results in:  
  + <NODE\_URL>/cirrus/search/<TableName>/Postgrestquery

Example: <https://node1.mercata-testnet2.blockapps.net/cirrus/search/BlockApps-Mercata-ERC20-Transfer?transaction_hash=eq.2207737bf7c54dce9b35da2ecb05cc7ef5f6a34163bbef48aad72b6730d58ce9>

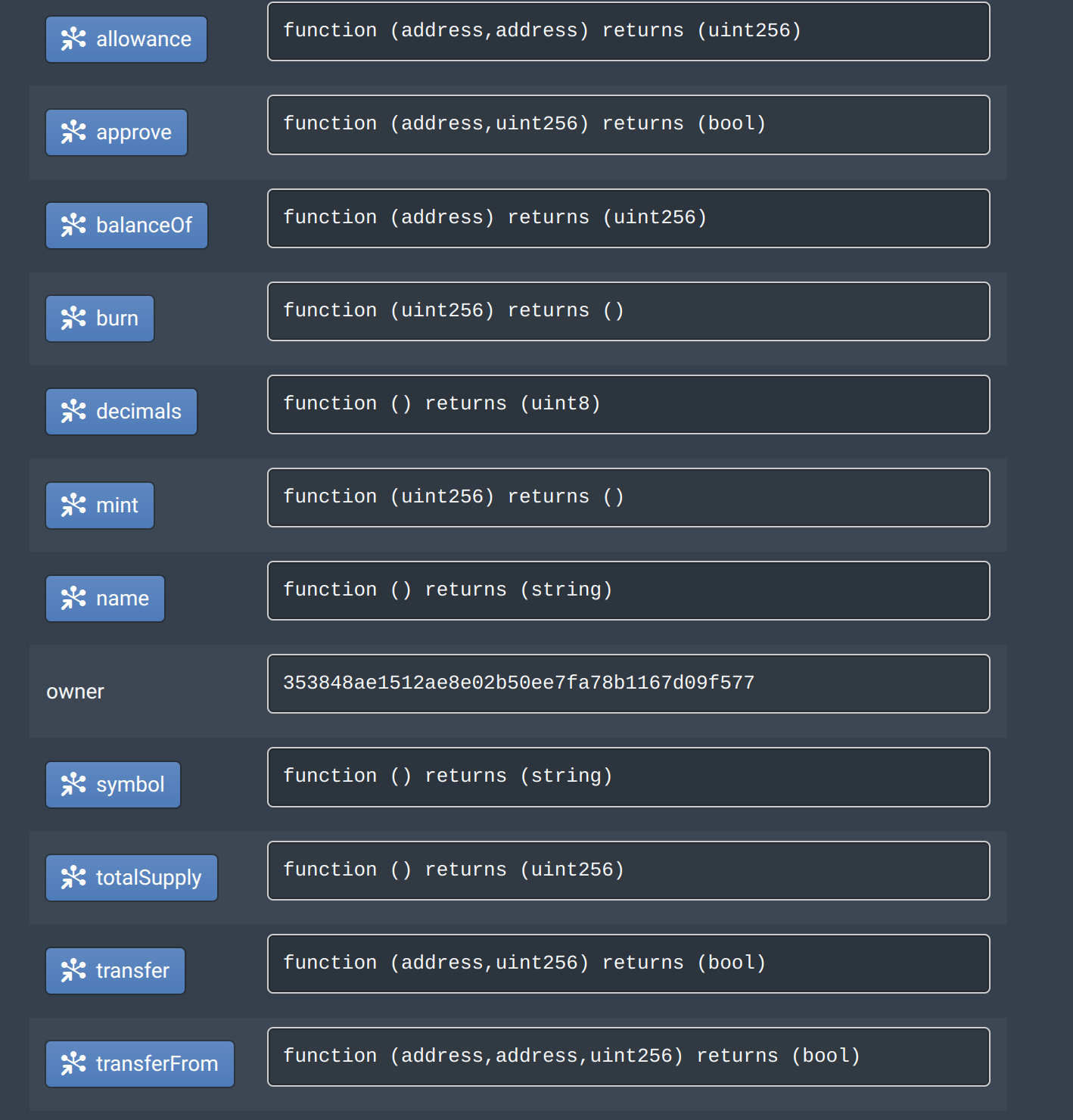
## **4. How to interact with Assets in Mercata?**

In the Mercata platform, as evidenced by the provided code, assets are represented as **ERC20 tokens** with additional metadata management.

* **ERC20 Compliance**: The core representation is based on the ERC20 standard, which defines how tokens are transferred and how information about them is accessed. This ensures compatibility with other systems and wallets that support ERC20 tokens.
* **Metadata Management**: The `MercataMetadata` contract is used to associate each asset (token) with additional information, such as name, description, images, files, and creation date. This metadata provides context and enriches the asset's representation.
* **Ownership and Control**: The `Ownable` contract is integrated, granting specific control to the owner of the asset. This includes the ability to mint new tokens, as demonstrated by the `mint` function, which is restricted to the owner.
* **Extensibility**: By being an abstract contract, the `Asset` contract provides a blueprint for more specific asset types. This allows for the creation of various asset representations with tailored functionalities while maintaining core features.

Overall, Mercata represents assets as a combination of standardized token functionality (ERC20) and customizable metadata, providing a flexible framework for managing diverse assets on a blockchain.

To interact with an ERC20 contract on SMD, first locate the contract by searching its name or address. Then, utilize the available ERC20 functions (like `transfer`) by inputting the necessary parameters and executing the call. Afterwards, review the transaction details, including the status and returned values.



| **Function Name** | **Parameters** | **Description** |
| --- | --- | --- |
| `totalSupply()` | None | Returns the total number of tokens in existence. |
| `balanceOf(address tokenOwner)` | `tokenOwner` (address): The address of the token holder. | Returns the number of tokens owned by the specified address. |
| `transfer(address to, uint256 value)` | `to` (address): The address to send tokens to. `value` (uint256): The number of tokens to send. | Transfers a specified number of tokens from the sender's address to the specified address. |
| `allowance(address owner, address spender)` | `owner` (address): The address of the token owner. `spender` (address): The address authorized to spend tokens. | Returns the number of tokens that the `spender` is allowed to spend on behalf of the `owner`. |
| `approve(address spender, uint256 value)` | `spender` (address): The address to authorize to spend tokens. `value` (uint256): The number of tokens to allow the `spender` to spend. | Approves the `spender` to spend a specified number of tokens on behalf of the sender. |
| `transferFrom(address from, address to, uint256 value)` | `from` (address): The address to transfer tokens from. `to` (address): The address to send tokens to. `value` (uint256): The number of tokens to send. | Transfers a specified number of tokens from one address to another, but only if the sender has approved the caller to spend on their behalf. |
| `mint(address to, uint256 value)` | `to` (address): The address to receive the new tokens. `value` (uint256): The number of tokens to create. | Creates new tokens and assigns them to the specified address. |
| `burn(uint256 value)` | `value` (uint256): The number of tokens to destroy. | Destroys a specified number of tokens from the sender's address. |

You can also view the state through Cirrus by visiting the following link:

`<NODE\_URL>/cirrus/search/BlockApps-Mercata-ERC20?address=eq.<CONTRACT\_ADDRESS>

For example:

<https://node1.mercata-testnet2.blockapps.net/cirrus/search/BlockApps-Mercata-ERC20?address=eq.4a5cf225d09e44c2c13fddce82673f41e69047bc>

You can then see the results:



To view ERC20 token balances in Mercata, use the following query structure:

<NODE\_URL>/cirrus/search/BlockApps-Mercata-ERC20-\_balances?address=eq.<CONTRACT\_ADDRESS>

#### ****Query Parameters

| Parameter | Description |
| --- | --- |
| `<NODE\_URL>` | Replace with the appropriate node URL. |
| `<CONTRACT\_ADDRESS>` | Replace with the ERC20 contract address. |
| `<OWNER\_ADDRESS>` | Replace with the address of the token holder whose balance you want to check (optional, but if not specified it will show all balances). |

#### Example Query

https://node1.mercata-testnet2.blockapps.net/cirrus/search/BlockApps-Mercata-ERC20-\_balances?address=eq.4a5cf225d09e44c2c13fddce82673f41e69047bc



This example query retrieves the balances for the ERC20 contract located at address `4a5cf225d09e44c2c13fddce82673f41e69047bc` on the node at `https://node1.mercata-testnet2.blockapps.net`.

## **5. How to interact with Liquidity Pools in Mercata?**

Liquidity pools in Mercata allow users to easily swap tokens, provide funds (liquidity), and withdraw funds—all through a simple graphical interface in the SMD portal. A liquidity pool consists of two assets (usually an ERC20 token paired with a stablecoin) that you can deposit or swap according to your needs.

### **1. Overview of Liquidity Pools**

* **Purpose:** Liquidity pools enable decentralized trading without a traditional order book. Users deposit an equal value of two tokens into the pool and, in return, receive liquidity pool (LP) tokens, which represent their share of the pool. These LP tokens give you the right to withdraw your deposited funds plus any earned fees.
* **Main Actions:**
  + **Adding Liquidity:** Deposit tokens and stablecoins into the pool.
  + **Removing Liquidity:** Withdraw your share of tokens and stablecoins from the pool.
  + **Swapping Tokens:** Exchange one asset for another (for example, stablecoin to token or token to stablecoin).

### **2. Using SMD to Interact with Liquidity Pools**

Once you’ve logged into SMD, you can interact with liquidity pools with just a few clicks. The SMD interface simplifies calling contract functions by providing dialog boxes to fill in required parameters.

#### **a. Adding Liquidity**

* **What It Means:** You deposit a certain amount of stablecoins and tokens into the pool. In return, the pool issues LP tokens to represent your share in the pool.
* **How to Do It in SMD:**
  1. **Navigate to the Liquidity Pool Contract:** Find the liquidity pool contract by searching its name or address in SMD.
  2. **Select the “Add Liquidity” Function:** Click on this function. A dialog box will appear.
  3. **Enter the Required Parameters:**
     + **Stable Amount:** The amount of stablecoins you wish to deposit.
     + **Maximum Token Amount:** The maximum number of tokens you are willing to deposit to maintain the pool ratio.
  4. **Call the Function:** When you click “Call Method,” the system calculates the exact token amount based on the current reserves.
  5. **Review Results:** After execution, you will get a TX Hash, status, and a confirmation that LP tokens have been issued to your account.

#### **b. Removing Liquidity**

* **What It Means:** You redeem your LP tokens to withdraw your share of the pool. The returned amount consists of both stablecoins and tokens based on the pool’s current balance.
* **How to Do It in SMD:**
  1. **Navigate to the Liquidity Pool Contract:** As before, locate the liquidity pool contract via SMD.
  2. **Select the “Remove Liquidity” Function:** Click on the function to open the input dialog.
  3. **Enter the Required Parameters:**
     + **LP Token Amount:** The number of LP tokens you want to redeem.
     + **Minimum Stablecoins:** The minimum stablecoin amount you expect to receive.
     + **Minimum Tokens:** The minimum token amount you expect to receive.
  4. **Call the Function:** Executing the function will burn the specified LP tokens and transfer your share of assets back to your wallet.
  5. **Review Results:** You will receive a transaction receipt showing the amounts returned, the TX Hash, and the confirmation status.

#### **c. Swapping Tokens**

Liquidity pools facilitate asset swaps. For example, you might exchange stablecoins for tokens, or vice versa. There are two main functions:

* **Stable to Token Swap (stableToToken):**
  + **Steps in SMD:**
    1. Go to the liquidity pool contract and select the “Stable to Token” function.
    2. Enter:  
       - **Stable Sold:** The amount of stablecoins you wish to exchange.
       - **Minimum Tokens Expected:** A threshold to protect against slippage (ensuring you receive at least this many tokens).
    3. Execute the function. SMD will display the calculated token amount based on current reserves.
    4. Review the transaction details (TX Hash, status, returned value).
* **Token to Stable Swap (tokenToStable):**
  + **Steps in SMD:**
    1. Find the “Token to Stable” function in the liquidity pool contract.
    2. Enter:  
       - **Tokens Sold:** The amount of tokens you want to exchange.
       - **Minimum Stablecoins Expected:** A threshold to ensure you receive a satisfactory amount of stablecoins.
    3. Execute the function. SMD shows how many stablecoins you will get.
    4. Check the transaction details afterward.

## **Key Function Summary for Liquidity Pools**

| Function Name | Parameters | Description |
| --- | --- | --- |
| Add Liquidity | Stable Amount, Maximum Token Amount | Deposits stablecoins and tokens into the pool; issues LP tokens to represent share. |
| Remove Liquidity | LP Token Amount, Minimum Stablecoins, Minimum Tokens | Redeems LP tokens to withdraw share of stablecoins and tokens. |
| Stable to Token Swap | Stable Sold, Minimum Tokens Expected | Exchanges stablecoins for tokens based on current pool reserves. |
| Token to Stable Swap | Tokens Sold, Minimum Stablecoins Expected | Exchanges tokens for stablecoins based on current pool reserves. |

### **4. Monitoring and Verifying Transactions**

There are three options to see the transaction results:

1. Through the Transactions tab in SMD
2. Through the strato-api REST API endpoint
3. Through the cirrus POSTGREST REST API endpoint

#### **Through the Transactions tab in SMD**

* Click on the Transactions tab
* You will see a Query Builder Box
* Choose Hash as the Query Type and add the TX Hash
  + Click on the result and you will see all the details
  + **Note:** You can go directly to this tab by adding the hash here: <https://node1.mercata-testnet2.blockapps.net/dashboard/#/transactions/2207737bf7c54dce9b35da2ecb05cc7ef5f6a34163bbef48aad72b6730d58ce9>

#### **Through the strato-api REST API endpoint**

* \<NODE\\_URL\>/strato-api/eth/v1.2/transactionResult/\<TXHASH\>
  + Example: <https://node1.mercata-testnet2.blockapps.net/strato-api/eth/v1.2/transactionResult/2207737bf7c54dce9b35da2ecb05cc7ef5f6a34163bbef48aad72b6730d58ce9>

#### **Through the cirrus POSTGREST REST API endpoint**

* This depends on what you are trying to do and how it affects the contract state. For a transfer, an event is emitted so you would see the results in:
  + \<NODE\\_URL\>/cirrus/search/\<TableName\>/Postgrestquery

Example: [https://node1.mercata-testnet2.blockapps.net/cirrus/search/BlockApps-Mercata-Pool?transaction\_hash=eq.2207737bf7c54dce9b35da2ecb05cc7ef5f6a34163bbef48aad72b6730d58ce9](https://node1.mercata-testnet2.blockapps.net/cirrus/search/BlockApps-Mercata-ERC20-Transfer?transaction_hash=eq.2207737bf7c54dce9b35da2ecb05cc7ef5f6a34163bbef48aad72b6730d58ce9)

## **6. Example Scenario**

Imagine the current liquidity pool has the following reserves:

* **Stablecoin reserve:** 1,000 units
* **Token reserve:** 500 tokens

From this, the current pool ratio is roughly 0.5 tokens per stablecoin (500 ÷ 1,000 = 0.5). The pool uses this ratio to ensure deposits maintain balance.

### 1. Adding Liquidity

### **Function: addLiquidity(stable\_amount, max\_tokens)**

**Process Explanation:**

* **Goal:** You deposit a specified amount of stablecoins along with a proportional number of tokens into the pool.
* **stable\_amount:** The exact number of stablecoins you want to deposit.
* **max\_tokens:** The maximum number of tokens you are willing to deposit to meet the pool’s required ratio. This protects you by preventing you from depositing more tokens than expected if the pool ratio changes unexpectedly.

**Example:**

* **stable\_amount:** 200 stablecoins
* **Calculating the Required Token Amount:** Using the current ratio, the required token amount is determined roughly by:  
   required tokens = (stable\_amount \* token\_reserve / stable\_reserve) + 1  
   Plugging in our numbers:  
   200×5001000=100200 \times \frac{500}{1000} = 100200×1000500​=100 tokens, plus a buffer of 1 equals **101 tokens.**
* **Setting max\_tokens:** To be safe, you set max\_tokens to a value slightly above 101; for example, **105 tokens.**

**In SApprove Stablecoin (TokenB):**

* **Navigate: Go to Contracts → Stablecoin.**
* **Select: The approve function.**
* **Enter Parameters:**
  + **spender: Paste the Pool contract address.**
  + **value: 300  
     *(Calculation: You plan to deposit 200 stablecoins; add a buffer (e.g., 100 extra) to ensure there’s enough allowance.)***
* **Action: Click Call Method and wait for confirmation.**

**Approve Asset Token (TokenA):**

* **Navigate: Go to Contracts → TokenA.**
* **Select: The approve function.**
* **Enter Parameters:**
  + **spender: Paste the Pool contract address.**
  + **value: 300  
     *(Calculation: With the pool ratio at 0.5 tokens per stablecoin, depositing 200 stablecoins requires approximately 100 tokens. Adding a generous buffer ensures you cover any fluctuations.)***
* **Action: Click Call Method and wait for confirmation.**

**Add Liquidity:**

* **Navigate: Go to Contracts → Pool.**
* **Select: The addLiquidity function.**
* **Enter Parameters:**
  + **stable\_amount: 200**
  + **max\_tokens: 105  
     *(Calculation: Based on the pool ratio, required tokens = (200 × 0.5) + 1 = 101 tokens; setting max\_tokens to 105 provides a small safety margin.)***
* **Action: Click Call Method.**
* **Result: The pool contract automatically withdraws 200 stablecoins and approximately 101 tokens (as long as the token amount is below your approved max\_tokens value) and then issues LP tokens representing your share.**

### 2. Removing Liquidity

### **Function: removeLiquidity(amount, min\_stable, min\_tokens)**

**Process Explanation:**

* **Goal:** Redeem your LP tokens to withdraw your share of both stablecoins and tokens from the pool.
* **amount:** The number of LP tokens you want to redeem.
* **min\_stable and min\_tokens:** These are the minimum amounts of stablecoins and tokens you expect to receive. They protect you from major unfavorable swings during the transaction.

**Example:**

* Suppose you hold **50 LP tokens**.
* The total LP token supply in the pool is **1,000 LP tokens**.
* **Calculation of Withdrawn Assets:**
  + **Stablecoins:** 50×1,000 stablecoins1,000 LP tokens=5050 \times \frac{1,000 \text{ stablecoins}}{1,000 \text{ LP tokens}} = 5050×1,000 LP tokens1,000 stablecoins​=50 stablecoins.
  + **Tokens:** 50×500 tokens1,000 LP tokens=2550 \times \frac{500 \text{ tokens}}{1,000 \text{ LP tokens}} = 2550×1,000 LP tokens500 tokens​=25 tokens.
* **Setting Minimums:** To account for possible minor fluctuations, you might set min\_stable to **48** and min\_tokens to **24.**

**In SMD:**

1. Open the liquidity pool contract.
2. Click on the removeLiquidity function.
3. Enter **50** for the amount (LP tokens), **48** for min\_stable, and **24** for min\_tokens.
4. Click “Call Method.”
5. The contract redeems your 50 LP tokens, calculates your share (50 stablecoins and 25 tokens), and if the values meet your minimums, completes the transaction.
6. You then see the transaction receipt with the amounts received.

### 3. Swapping Tokens

### **A. Stable-to-Token Swap**

### **Function: stableToToken(stable\_sold, min\_tokens)**

**Process Explanation:**

* **Goal:** Exchange a certain amount of stablecoins for tokens.
* **stable\_sold:** The amount of stablecoins you want to swap.
* **min\_tokens:** The minimum number of tokens you expect to receive in exchange, safeguarding you against slippage.

**Example:**

* **stable\_sold:** 100 stablecoins.
* Suppose the current pool conditions suggest you should receive around **45 tokens** based on the pricing function.
* To ensure you’re protected against small price fluctuations, you set min\_tokens to **44 tokens.**

**In SMD: Complete Flow for Stable→Token Swap**

1. **Approve TokenB (Stablecoin)**
   * **Navigate:** Contracts → **TokenB (Stablecoin)**
   * **Select:** approve
   * **Params:**
     + **spender:** Pool contract address
     + **value:** 150  
       - *Calculation:* You plan to swap 100 TokenB; add a 50‑unit buffer → 100 + 50 = 150.\*
   * **Action:** Click **Call Method** and wait for confirmation.
2. **Open the Pool Contract**
   * Navigate to **Contracts → Pool** and click to open the liquidity pool contract page.
3. **Select stableToToken**
   * Scroll to find and click the **stableToToken** function.
4. **Enter Parameters**
   * **stable\_sold (TokenB):** 100
   * **min\_tokens (TokenA):** 44  
     + *Calculation:* Expected tokens ≈45; setting min\_tokens to 44 protects against minor price fluctuations.
5. **Click “Call Method”**
   * Execute the swap.
6. **Result**
   * The pool deducts 100 TokenB and transfers at least 44 TokenA to your address.
   * Check the **Transactions** tab to confirm the transaction status and exact amount received.

### **B. Token-to-Stable Swap**

### **Function: tokenToStable(tokens\_sold, min\_stable)**

**Process Explanation:**

* **Goal:** Exchange a certain number of tokens for stablecoins.
* **tokens\_sold:** The number of tokens you wish to swap.
* **min\_stable:** The minimum number of stablecoins you expect to receive in return.

**Example:**

* **tokens\_sold:** 50 tokens.
* Imagine the expected stablecoin return, based on current pool conditions, is around **90 stablecoins**.
* To cover potential minor slippage, set min\_stable to **88 stablecoins.**

**In SMD: Complete Flow for Token→Stable Swap**

1. **Approve TokenA (Asset Token)**
   * **Navigate:** Contracts → **TokenA (Your Asset)**
   * **Select:** approve
   * **Params:**
     + **spender:** Pool contract address
     + **value:** 60  
       - *Calculation:* You plan to sell 50 TokenA; add a 10‑token buffer → 50 + 10 = 60.\*
   * **Action:** Click **Call Method** and wait for confirmation.
2. **Open the Pool Contract**
   * Navigate to **Contracts → Pool** and click to open the liquidity pool contract page.
3. **Select tokenToStable**
   * Scroll to find and click the **tokenToStable** function.
4. **Enter Parameters**
   * **tokens\_sold (TokenA):** 50
   * **min\_stable (TokenB):** 88  
     + *Calculation:* Expected stablecoins ≈90; setting min\_stable slightly lower for safety (88).
5. **Click “Call Method”**
   * Execute the swap.
6. **Result**
   * The pool deducts 50 TokenA and transfers at least 88 TokenB (stablecoins) to your address.
   * Check the **Transactions** tab to confirm the transaction status and cirrus for exact amount received.

## **Summary of the Process**

1. **Preparation:** Understand the current reserves and pool ratio. This gives you an idea of how many tokens will be required when adding liquidity or what you might receive during swaps.
2. **Adding Liquidity:**
   * You choose how many stablecoins to add.
   * The interface calculates the required tokens based on the pool ratio.
   * You enter a “max\_tokens” value to cap the token amount you deposit.
   * Confirm the transaction and receive LP tokens representing your pool share.
3. **Removing Liquidity:**
   * You specify how many LP tokens to redeem.
   * You set minimum acceptable values for stablecoins and tokens to avoid receiving less than expected.
   * Upon confirmation, the corresponding assets are transferred back to your account, and your LP tokens are burned.
4. **Swapping:**
   * For **stable-to-token swaps**, decide the amount of stablecoins you want to exchange and the minimum number of tokens you expect.
   * For **token-to-stable swaps**, determine the number of tokens to swap and the minimum stablecoins you expect in return.
   * Execute the swaps, and then verify the transaction outcomes in SMD.

## **6. Flow for Redeeming a Token through SMD**

This guide describes the complete step-by-step process to redeem assets represented as tokens in Mercata’s platform through the **PhysicalRedemptionService** (for metals or other physical assets) and **CryptoRedemptionService** (for crypto-assets) contracts.

### **Redeeming a Metal Token (Physical Asset):**

This flow uses PhysicalRedemptionService.sol.

**Preconditions:**

* You have Metal tokens in your account.
* You've logged into SMD.

**Step-by-Step Flow:**

1. **Approve Token Spending (Metal Token Contract):**
   * **Navigate:** Contracts → Your Metal Token Contract (e.g., "Gold").
   * **Click:** approve
   * **Params:**
     + spender: (Address of the PhysicalRedemptionService contract)
     + value: Amount to redeem
   * **Action:** Click **Call Method** and confirm transaction.
2. **Redeem Tokens at Spot Price:**
   * **Navigate:** Contracts → **PhysicalRedemptionService**
   * **Click:** redeemAtSpot
   * **Params:**
     + tokenAmount: Amount you wish to redeem (e.g., 50)
     + baseAddress: *(not used for physical assets, can enter any placeholder or leave empty)*
   * **Action:** Click **Call Method**
   * **Result:**
     + Your tokens are transferred to the redemption contract, burned (sent to zero address), and redemption initiated.
     + Check **Transactions** tab for confirmation and details.
3. **Optional (Sell Tokens for USDST at Spot):** If you want USDST instead of physical redemption:  
   * **Navigate:** Contracts → **PhysicalRedemptionService**
   * **Click:** sellForSpot
   * **Params:**
     + tokenAmount: Amount of tokens you wish to sell
   * **Action:** Click **Call Method**
   * **Result:**
     + Your tokens transfer to the redemption contract, and you receive USDST tokens.
     + Check **Transactions** tab for details.

### **Redeeming a Crypto Token:**

This flow uses CryptoRedemptionService.sol.

**Preconditions:**

* You hold Crypto tokens in your account.
* You've logged into SMD.
* You have an external crypto wallet address ready for receiving redeemed native crypto assets.

**Step-by-Step Flow:**

1. **Approve Token Spending (Crypto Token Contract):**
   * **Navigate:** Contracts → Your Crypto Token Contract
   * **Click:** approve
   * **Params:**
     + spender: (Address of the CryptoRedemptionService contract)
     + value: Amount to redeem
   * **Action:** Click **Call Method** and confirm transaction.
2. **Redeem Tokens at Spot Price:**
   * **Navigate:** Contracts → **CryptoRedemptionService**
   * **Click:** redeemAtSpot
   * **Params:**
     + tokenAmount: Amount of tokens you wish to redeem (e.g., 100)
     + baseAddress: External crypto wallet address to receive native crypto asset
   * **Action:** Click **Call Method**
   * **Result:**
     + Your tokens transfer to the redemption contract.
     + The contract initiates redemption via a blockchain bridge. *(Note: Bridge ERC20 integration requires completion.)*
     + Check **Transactions** tab to confirm redemption initiation and monitor transaction status.

## **Additional Transaction Verification:**

After initiating redemption transactions, confirm their statuses via:

* **SMD → Transactions Tab**: Search by transaction hash to confirm success and retrieve details.
* Or use API endpoints:  
  + https://marketplace.mercata.blockapps.net/bloc/v2.2/transactions/{hash}/result

## **7. Flow for Creating a New Contract via Mercata**

1. **Deploy Mercata Contracts Locally**

**Terminal:  
  
Go to the strato-platform directory:**

cd mercata/contracts

yarn deploy

**Output Example:**  
Contract deployed successfully!

Contract address: 6ce48262a934015585375b5d0a98bb87a69be50e

====== Deployment Successful ======

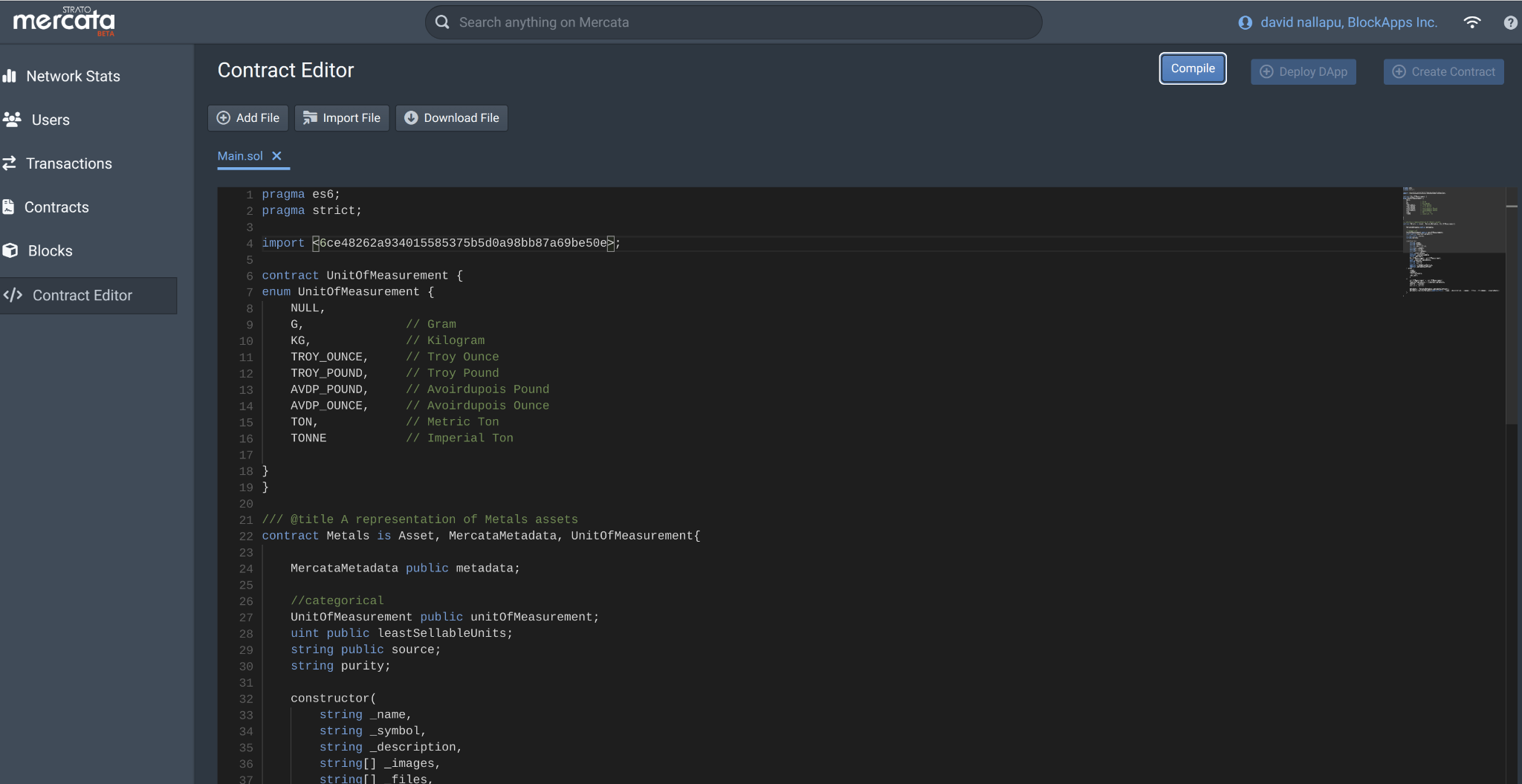
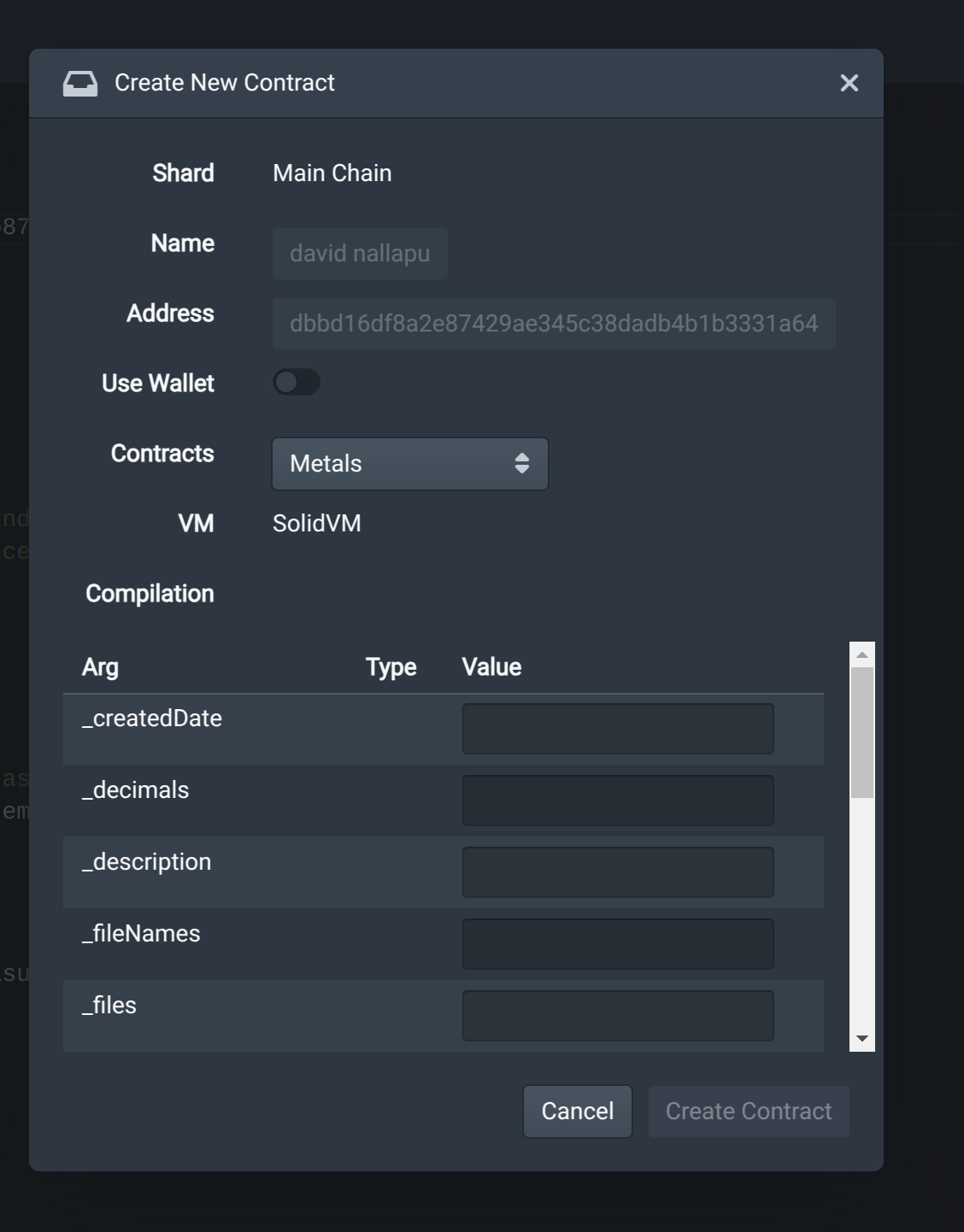
Contract: Mercata

Address: 6ce48262a934015585375b5d0a98bb87a69be50e

===================================

BASE\_CODE\_COLLECTION=6ce48262a934015585375b5d0a98bb87a69be50e

* + **Note:** This address (6ce48262a934015585375b5d0a98bb87a69be50e) is your **Base Code Collection**—the library/blueprint for Mercata contracts.

1. **Open SMD’s Contract Editor**
   * In SMD’s side menu, click **Contract Editor**.
2. **Paste Your Contract Code**
   * Copy the Solidity source of the contract you wish to deploy (e.g., Metals.sol or another Mercata-based contract).
3. In the editor, **replace** the placeholder <BASE\_CODE\_COLLECTION> with your deployed Mercata address (6ce48262a934015585375b5d0a98bb87a69be50e).
4. **Compile the Contract**
   * Click the **Compile** button.
   * If there are any errors, SMD will highlight them—make any necessary code adjustments and recompile.
5. **Fill in Constructor Parameters**
   * After successful compilation, SMD displays a form with your contract’s constructor inputs. For example, a Metals contract might ask for:  
     + name (string)
     + symbol (string)
     + description (string)
     + images (string[])
     + files (string[])
     + fileNames (string[])
     + createdDate (uint)
     + initialSupply (uint256)
     + decimals (uint8)
     + unitOfMeasurement (enum)
     + leastSellableUnits (uint)
     + source (string)
     + purity (string)
     + redemptionService (address)
     + metadataContract (address)
6. **Create the Contract**
   * After filling in all required fields, click **Create Contract**.
   * SMD will broadcast the deployment transaction to Strato.
7. **View Deployment Results**
   * SMD will display:  
     + **New Contract Address** (e.g., 0xabc123…)
     + **Transaction Hash**
   * **Where to Find Your Contract:**
     + **Contracts Tab:** Your new contract appears in the list.
     + **Transactions Tab:** The deployment transaction is logged—search by its hash to view details.

You’ve now successfully deployed a new Mercata-based contract using SMD’s Contract Editor. If you need to interact with this contract (e.g., mint tokens, register metadata), simply navigate to it under the **Contracts** tab and use the provided function calls.