# **STRATO Mercata Integration Guide for Exchanges**

This document explains how to integrate your system as a Centralized Exchange (CEX) (and can be adapted for other exchanges) with the STRATO Mercata platform. It covers the following topics:

* How to register and log in
* Obtaining an access token via OAuth2 with token caching
* Differences between your contracts (which are not ERC-20/ERC-721) and standard token contracts
* How to transfer ownership of an asset (e.g., asset transfer, token transfer)
* How to resolve and interpret transaction fields via API endpoints
* Request limiting and nonce limits for transactions
* The differences of a PBFT chain compared to Ethereum

**Note:** Since this is a PBFT chain, blocks are only produced when there are transactions. In contrast, Ethereum (a POW or POS chain) creates blocks at regular intervals irrespective of transactions. This has important consequences when checking for transaction finality and when monitoring network performance.

## **1. Account Registration and Login**

Before accessing any of the trading or API endpoints, you must register an account and log in. Follow these steps:

### **Register an Account**

1. Go to:<https://marketplace.mercata.blockapps.net>
2. Click “Register” and fill in your details.
3. Check your email for a verification link and confirm your account.

*Repeat these steps once more:*

* **Register an Account (Detailed):**
  + Visit [marketplace.mercata.blockapps.net](https://marketplace.mercata.blockapps.net)
  + Click **"Register"** and complete the registration form.
  + Verify your account using the email verification link you receive.

### **Log In**

1. After verification, log in at [marketplace.mercata.blockapps.net](https://marketplace.mercata.blockapps.net).

### **Accessing API and Other Links**

1. Once logged in you may visit the marketplace.
2. **Important:** API links cannot be accessed directly from the marketplace application. Instead, use the links separately in your browser (after logging in).
3. **Example API endpoint:** https://marketplace.mercata.blockapps.net/strato-api/eth/v1.2/block?number=1
4. For full API documentation visit:  
    [docs.blockapps.net/api/](https://docs.blockapps.net/api/)

**Remember:** You must be logged in before accessing any of these links, or they will not work.

## **2. Access Tokens and Token Caching**

Access tokens (OAuth2) have a limited lifetime (currently 5 minutes) for security. It is best practice to refresh the token only shortly before it expires.

Here’s an example using a token caching mechanism in your application code:

const TOKEN\_LIFE\_THRESHOLD\_SECONDS = 30;

let CACHED\_DATA = {

token: null,

tokenExpiresAt: null,

};

const getToken = async (req = null) => {

let token = CACHED\_DATA.token;

const expiresAt = CACHED\_DATA.tokenExpiresAt;

if (

!token ||

!expiresAt ||

expiresAt <= Math.floor(Date.now() / 1000) + TOKEN\_LIFE\_THRESHOLD\_SECONDS

) {

const tokenObj = await your\_get\_token\_function\_here();

token = tokenObj.token['access\_token'];

CACHED\_DATA.token = token;

CACHED\_DATA.tokenExpiresAt = Math.floor(tokenObj.token.expires\_at / 1000);

}

return token;

};

Call getToken() whenever a valid access token is needed in your code.

## **3. Contract Differences: Not ERC‑20 or ERC‑721**

Unlike standard token contracts on Ethereum (ERC‑20/ERC‑721), your contracts follow a custom design. For example, the **Asset contract** (which is the base) can be seen here:

* **Asset Contract (Base):** [Asset.sol](https://github.com/blockapps/strato-mercata-opensource/blob/main/marketplace/backend/dapp/mercata-base-contracts/Templates/Assets/Asset.sol)

A contract such as **Metals.sol** (for Gold) ultimately inherits from Asset:

* **Metals Contract (Gold):** [Metals.sol](https://github.com/blockapps/strato-mercata-opensource/blob/main/marketplace/backend/dapp/items/contracts/Metals.sol)

Because your contracts are not built to the ERC-20 or ERC-721 standards, interactions with them differ; each transfer generates a new UTXO-style contract instance for the token.

## **4. How to Interact with the Contracts (Examples)**

### **4.1 Transferring an Asset**

Below is an example script to transfer an asset using the custom “automaticTransfer” method:

bash

USERNAME=""

PASSWORD=""

ASSET\_ADDRESS=""

SIMPLE\_RESERVE\_ADDRESS="758834d4c9d52949ee4d04423eceb222e171388e"

USDST\_QUANTITY="1000000000000000000000000000000"

# Get access token

ACCESS\_TOKEN=$(curl -L -X POST "https://keycloak.blockapps.net/auth/realms/mercata-testnet2/protocol/openid-connect/token" \

-H 'Content-Type: application/x-www-form-urlencoded' \

-H 'Authorization: Basic ' \

--data-urlencode 'grant\_type=password' \

--data-urlencode "username=$USERNAME" \

--data-urlencode "password=$PASSWORD" \

-s | jq -r '.access\_token')

echo "Access token: $ACCESS\_TOKEN"

# Generate a random transfer number

TRANSFER\_NUMBER=$(shuf -i 1000-9999 -n 1)

# Use the access token to call purchaseTransfer for Asset

NEW\_ASSET\_RESULT=$(curl -X POST "https://marketplace.mercata.blockapps.net/bloc/v2.2/transaction?resolve=true" \

-H 'Content-Type: application/on' \

-H "Authorization: Bearer $ACCESS\_TOKEN" \

-d '{"txs":[{"payload":{"contractAddress":"'"$ASSET\_ADDRESS"'","method":"automaticTransfer","args":{"\_newOwner":"'"$SIMPLE\_RESERVE\_ADDRESS"'","\_quantity":'"$USDST\_QUANTITY"',"\_transferNumber":'"$TRANSFER\_NUMBER"',"\_price":0.1}},"type":"FUNCTION"}],"txParams":{"gasLimit":10000000000,"gasPrice":1}}')

echo "NEW\_ASSET\_RESULT: $NEW\_ASSET\_RESULT"

NEW\_ASSET\_ADDRESS=$(echo $NEW\_ASSET\_RESULT | jq -r '.[0].txResult.contractsCreated')

### **4.2 Transferring a Token**

Below is a script to transfer tokens (where the token is treated as a UTXO):

bash

# Initialize variables for the source user's credentials

FROM\_USERNAME=""

FROM\_PASSWORD=""

# Initialize variable for the target username

# The username is associated with an X.509 certificate, which contains the user's blockchain address.

TO\_USERNAME=""

# Set the quantity of tokens to transfer

TOKEN\_QUANTITY="100"

# Set the address of the token contract

TOKEN\_ADDRESS="e7d32bc37325377b6e530643438c8e60d62d201e"

# Obtain an access token from the authentication server using the source user's credentials

ACCESS\_TOKEN=$(curl -L -X POST "https://keycloak.blockapps.net/auth/realms/mercata/protocol/openid-connect/token" \

-H 'Content-Type: application/x-www-form-urlencoded' \

-H 'Authorization: Basic Y29pbnN0b3JlLmNvbTpiNjMxN2M1Yy1jNzk2LTQ2ZWMtOTQwYy0zOWUzNzU2YmE0ZTE=' \

--data-urlencode 'grant\_type=password' \

--data-urlencode "username=$FROM\_USERNAME" \

--data-urlencode "password=$FROM\_PASSWORD" \

-s | jq -r '.access\_token')

echo "Access token: $ACCESS\_TOKEN"

# Fetch the user address associated with the target username from the API

RAW\_RESPONSE=$(curl -s "https://marketplace.mercata.blockapps.net/cirrus/search/Certificate?commonName=eq.$TO\_USERNAME&select=userAddress" \

-H 'Accept: application/on' \

-H 'Content-Type: application/on' \

-H "Authorization: Bearer $ACCESS\_TOKEN")

echo "Raw API Response: $RAW\_RESPONSE"

# Extract the user address from the API response

TO\_ADDRESS=$(echo "$RAW\_RESPONSE" | jq -r '.[0].userAddress')

if [ -z "$TO\_ADDRESS" ]; then

echo "Error: TO\_ADDRESS is empty. Please check the username or the API response."

exit 1

fi

echo "TO\_ADDRESS: $TO\_ADDRESS"

# Generate a random transfer number to uniquely identify the transaction

TRANSFER\_NUMBER=$(shuf -i 1000-9999 -n 1)

# Use the access token to initiate a token transfer transaction on the blockchain

NEW\_TOKEN\_RESULT=$(curl -X POST "https://marketplace.mercata.blockapps.net/bloc/v2.2/transaction?resolve=true" \

-H 'Content-Type: application/on' \

-H "Authorization: Bearer $ACCESS\_TOKEN" \

-d '{"txs":[{"payload":{"contractAddress":"'"$TOKEN\_ADDRESS"'","method":"automaticTransfer","args":{"\_newOwner":"'"$TO\_ADDRESS"'","\_quantity":'"$TOKEN\_QUANTITY"',"\_transferNumber":'"$TRANSFER\_NUMBER"',"\_price":0.1}},"type":"FUNCTION"}],"txParams":{"gasLimit":10000000000,"gasPrice":1}}')

if ! echo "$NEW\_TOKEN\_RESULT" | jq empty; then

echo "Error: NEW\_TOKEN\_RESULT is not a valid ON. Response: $NEW\_TOKEN\_RESULT"

exit 1

fi

echo "NEW\_TOKEN\_RESULT: $NEW\_TOKEN\_RESULT"

NEW\_TOKEN\_ADDRESS=$(echo $NEW\_TOKEN\_RESULT | jq -r '.[0].txResult.contractsCreated[0]')

echo "NEW\_TOKEN\_ADDRESS: $NEW\_TOKEN\_ADDRESS"

NEW\_TOKEN\_DATA=$(curl -s "https://marketplace.mercata.blockapps.net/cirrus/search/BlockApps-Mercata-Asset?address=eq.$NEW\_TOKEN\_ADDRESS" \

-H 'Accept: application/on' \

-H 'Content-Type: application/on' \

-H "Authorization: Bearer $ACCESS\_TOKEN")

echo "NEW\_TOKEN\_DATA: $NEW\_TOKEN\_DATA"

**Explanation:** This script demonstrates the connection between the x509 certificate‑based authentication system, the username, and the user address. In this context, transferring ownership of an asset (or UTXO) involves generating a new contract instance on the blockchain.

**Keep in mind:** Since your chain uses PBFT, every valid transaction result means that once the transaction is successful it will be included in a new block (see further details below).

## **5. How to Resolve Transaction Fields**

When you post a transaction you receive a response with the transaction hash and status. You can use the API endpoints to query and resolve details of the transaction.

### **Primary Transaction Data**

1. **Transaction Hash (txhash):** Found directly in the "hash" field inside "receiptTransactions", for example:  
    https://marketplace.mercata.blockapps.net/strato-api/eth/v1.2/block?hash=32e42447e42a767d6beb0b8d1d6261b58db69c36d564fb02890caefd18becc6e  
     
    You can also query the tx hash after receiving it:  
    https://marketplace.mercata.blockapps.net/bloc/v2.2/transactions/328a7b57dbc7d40b68d7b0130553c358d4e15e7d028905cde82622f707482a8c/result
2. **Sender User Address (from):** Query the *ItemTransfers* API using the transaction hash:  
    https://marketplace.mercata.blockapps.net/cirrus/search/BlockApps-Mercata-Asset-ItemTransfers?transaction\_hash=eq.<tx\_hash>  
    Look for the "transaction\_sender" field.
3. **Receipt Address:** Found in the "newOwner" field from the *ItemTransfers* API response.
4. **Quantity (Amount of Token Transferred):** Found in the "quantity" field in the *ItemTransfers* API response.

### **UTXO and Token Details**

1. **Transaction Status:** When you post a transaction, the API response includes the transaction hash as well as its success/failure status. Since this is a PBFT chain, once you see a “tx success” status, the transaction will be included in the next block.  
     
    You can verify the status by polling:  
    https://marketplace.mercata.blockapps.net/strato-api/eth/v1.2/transactionResult/{txHash}  
     
    Or checking via:  
    https://marketplace.mercata.blockapps.net/bloc/v2.2/transactions/{txhash}/result  
     
    *Note:* Repeatedly mention that since this is a PBFT chain, block production happens only when there are transactions—this is different from Ethereum’s POW/POS where blocks are produced continuously.

## **6. Request Limits and Nonce Limitations**

### **Request Limit**

Your interface has several limits:

* **Rate Limit:** The rate limit is 1.25 API calls per millisecond. If you exceed this rate:  
  + For the first 100 calls, delays are added.
  + Further requests are blocked until the rate decreases.
* **Nonce Limit:** Each account (for example, the “glant” account) can only submit up to 4,000 transactions before reaching the nonce cap.

### **Unique Identifiers**

* Each transaction is unique and can be identified by the transaction hash.
* You can also use the **userAddress** (which is unique) as an identifier.

### **API Queries Examples**

Here are some example queries:

1. **Check Certificate by User Address:** https://marketplace.mercata.blockapps.net/cirrus/search/Certificate?userAddress=eq.08044a8717cc942b91d1ae31aad006c93d96cb7a
2. **Query Item Transfers (for Asset GOLDST, example):** https://marketplace.mercata.blockapps.net/cirrus/search/BlockApps-Mercata-Asset-ItemTransfers?assetName=eq.GOLDST&oldOwnerCommonName=eq.David%20Nallapu&newOwner=eq.08044a8717cc942b91d1ae31aad006c93d96cb7a
3. **Query Asset Data:** https://marketplace.mercata.blockapps.net/cirrus/search/BlockApps-Mercata-Asset?name=eq.GOLDST&owner=eq.08044a8717cc942b91d1ae31aad006c93d96cb7a&select=address,name,quantity

## **7. Additional Notes on Integration**

1. **Preventing Spamming:** Some endpoints were put in place to prevent spamming (there are no transaction fees at the moment). More details may be provided later.
2. **Transfer Number Generation:** Using a random number generator for the transfer number is acceptable and is mainly for bookkeeping purposes. You may use a similar approach in your interface.
3. **Transaction Consensus:** Though most transactions are fast, consensus time can sometimes take 15–20 seconds depending on the transaction type and node health.
4. **Block Production in PBFT vs Ethereum:**
   * **PBFT Chain:** Blocks are only produced when there are pending transactions. For example, while the average new block time is 3 seconds, there can be larger gaps if there are no transactions.
   * **Ethereum Chain (POW/POS):** Blocks are produced continuously (every ~15 seconds for POW, or as determined by POS), regardless of transaction volume.
5. **Implication:** When checking transaction status or block explorer data (for example, at<https://marketplace.mercata.blockapps.net/dashboard/#/blocks>), note that the absence of a new block on your PBFT chain simply means there were no transactions to process rather than a network slowdown.
6. **Validating a Transaction:** To check if a transaction is valid, call:  
    https://marketplace.mercata.blockapps.net/strato-api/eth/v1.2/transactionResult/<txhash>
7. **Certificate Data Query:** To check for a user certificate (for example, for user address "abc"):  
    https://marketplace.mercata.blockapps.net/cirrus/search/Certificate?userAddress=eq.abc  
    This query will return an empty array if no certificate exists.
8. **Endpoints Overview:**
   * **strato-api:** Used for querying transaction data, block data, and blockchain-specific information.
   * **cirrus:** Used for querying indexed data stored in Postgres (such as asset/UTXO data and certificate data).
9. Therefore, when looking for asset or UTXO data, use the cirrus endpoints.

## **8. Summary and Final Remarks**

* **Account Setup:** Create an account and complete email verification before attempting any transaction or API calls.
* **Token Management:** Ensure your application uses token caching and automatically refreshes the token based on its lifetime.
* **Contract Interaction:** Your contract architecture is different from ERC-20/ERC-721, so transfers create new UTXO-like contract instances. Be sure to refer to the Asset and Metals contract examples provided in the GitHub links.
* **Transaction Monitoring:** Use the provided endpoints to retrieve transaction hashes, check sender and receiver addresses, verify transferred amounts, and determine transaction status. Always bear in mind the differences in consensus—on a PBFT chain, transactions are confirmed only when there are pending transactions, unlike Ethereum’s block production mechanism.
* **Request & Nonce Limits:** Mind your API call rate and nonce limits (maximum 4,000 transactions per account), as these directly impact transaction submission.
* **Integration with PBFT:** The platform uses PBFT consensus which can result in variable block times (averaging 3 seconds, but with possible larger gaps). This feature sets your implementation apart from typical Ethereum-based systems.