Logistichain functional tests

Thank you for showing your interest in Logistichain. This document helps you understand the basic functionalities of the blockchain as well as their behaviour. Feel free to try and break some stuff by running a private network yourself. You run a private network by disconnecting with existing Logistichain nodes or by changing the network identifier in the blockchain code.

https://github.com/logistichain

Test setup for all testcases

All testcases start with this setup, unless described otherwise. Go to the next page to start testing and skip this page. When a testcase refers to these steps, then execute them.

- Run two instances of the software on one computer. The instances run on port 10101 and 20202 and are called *instance 1* and *instance 2* respectively. Make sure both instances are running from **different directories** and **not connected to other external nodes** so they both use their own local blockchain file (named blockchain-testnet.json, located in the same directory as the executable) in a private network.
- Run instance 1. Make sure all existing network connections are disconnected
- Run instance 2 and change the port by running these commands:
    - `Networking setport` *[enter]*
    - `<specify the port, like 20202>` *[enter]*
    - `Networking restart` *[enter]*
- The mining difficulty mechanism needs to adjust itself to the mining power, so the first 30-50 blocks will be created quickly. To prevent message spam, first create around 50 blocks. Run these commands on instance 1:
    - `Startmining` *[enter]*
    - And after 50+ blocks: `stopmining` *[enter]*
- Connect instance 1 with instance 2 by executing with the following commands on instance 1:
    - `Networking connect` *[enter]*
    - `127.0.0.1:20202` *[enter]*
- Confirm that the second instance is fully synchronized by receiving the debug message “[INF] Successfully synced with remote node” te ontvangen.

This setup also covers the networking functionality of the software. In some cases a race condition may appear when you mine with multiple instances at the same time, causing both instances to reject each other’s blocks. When this happens, stop the mining on both instances and run `resetblockchain` on both instances. Shut the instances down and restart by following the steps above.

Earn tokens

This tests the implicit operation of creating a valid block and making a transaction. According to the protocol, the first transaction must be a 'coinbase' transaction, where the miner claims new tokens as a reward for making a valid block.

Testcase identifier	1. Earn tokens
Test items	Creating and publishing a valid block and check the token balance.
Begin situation	The test setup as described on page 2, except the mining part. That will be done here.
Event flow	● After starting the first instance, memorize the public key ● Run the command `startmining` ● Wait until one or more blocks have been created ● Stop mining by running the `stopmining` command ● Validate the token balance by running the `accounts` command, look up your public key
Expected result	One or more blocks have been created in which the public key gets credited 5000 tokens for each new block. The total balance is 5000 multiplied by the amount of created blocks.
Actual result	
Analysis	
Dependencies on other testcases	*None*

Transfer tokens

Transferring tokens tests the implicit operation of publishing transactions.

Testcase identifier	2. Transfer tokens
Test items	Transferring tokens to another public key and checking if this process was successful.
Begin situation	The test setup as described on page 2.
Event flow	● Copy the public key after starting the instance ● Execute the command `transfertokens` and follow these steps: ● At “Enter the sender’s public key”, fill in the public key ● At “Enter the receiver’s public key”, fill in “montapacking” ● At “Enter the amount to send”, fill in “1” ● At “Enter optional data[]”, fill in “test” ● Confirm that the transaction was submitted to the transactionpool successfully ● Add the transaction to the next block by executing the command `startmining` ● Once a block has been found, stop mining by using the command `stopmining` ● Verify the balances with the command `accounts`
Expected result	The transaction was added to the transactionpool successfully. The transaction was included in a block. The balance of the public key is (new amount of blocks * 5000) – 11 because 1 token was sent and the action costed 10 tokens. The account “montapacking” is in the list of accounts and has a balance of 1 TK.
Actual result	
Analysis	
Dependencies on other testcases	-

1.1 Create SKU

This action registers an SKU (product information) as a blueprint in the blockchain. This functionality also enables the user to create supply for the product immediately. The supply will be added to the creator's public key balance.

Testcase identifier	3. Create SKU
Test items	The creation of product information (Stock Keeping Unit) and assigning initial supply to the public key of the SKU creator.
Begin situation	The test setup as described on page 2.
Event flow	● Copy the public key after starting the instance ● Write down your current token balance by executing the command `accounts` and find the balance for your public key ● Execute `createsku` and follow the next steps: ● At "Enter the sender's public key", fill in your public key ● At "Enter the SKU name", fill in "Bicycle" ● At "Enter EAN Code", fill in "1234567890" ● At "Provide a description", fill in "TREK black" ● At "Specify the initial supply", fill in "100" ● Confirm that the transaction was submitted to the transactionpool successfully ● Add the transaction to the next block by executing the command `startmining` ● Once a block has been found, stop mining by using the command `stopmining` ● Verify that the action costed 100 tokens with command `accounts` ● Verify the product information with the command `skus`
Expected result	The transaction was added to the transactionpool successfully. The transaction was included in a block. The balance of the public key is (new amount of blocks * 5000) – 100 because the action costed 100 tokens. The correct product information is shown after using the `skus` command.
Actual result	
Analysis	
Dependencies on other testcases	-

1.2 Create supply for SKU

This action enables an SKU creator to create new supply for the existing Stock Keeping Unit. The receiver of the new stock can be specified immediately.

Testcase identifier	4. Create supply for SKU
Test items	The creation of new supply for an existing product.
Begin situation	The test setup as described on page 2.
Event flow	● Copy the public key after starting the instance ● Write down your current token balance by executing the command `accounts` and find the balance for your public key ● Execute `createsupply` and follow the next steps: ● At "Enter block hash where the SKU was created", fill in the 'block hash' value, found in the results of the `skus` command from testcase 3. ● At "Enter the index of the transaction where the SKU resides" fill in "1" ● At "Specify the new amount of supply to create", fill in "1" ● At "Enter optional data", press enter ● Confirm that the transaction was submitted to the transactionpool successfully ● Add the transaction to the next block by executing the command `startmining` ● Once a block has been found, stop mining by using the command `stopmining` ● Verify that the action costed 100 tokens with command `accounts` ● Verify that the global supply increased with 1 by using the command `skus` (check the last result in this list)
Expected result	The transaction was added to the transactionpool successfully. The transaction was included in a block. The balance of the public key is (new amount of blocks * 5000) – 100 because the action costed 100 tokens. The global supply is shown as 101 after using the `skus` command.
Actual result	
Analysis	
Dependencies on other testcases	Testcase 3

1.3 Destroy supply for SKU

The last step in the process is set when the goods arrive at the consumer. Because the consumer doesn't bother to create a public and private key, the supply will be destroyed from the blockchain by adding another transaction. This functionality is also used when supply goes lost or stolen.

Testcase identifier	5. Destroy supply for SKU
Test items	Deleting existing supply for an existing Stock Keeping Unit.
Begin situation	The test setup as described on page 2.
Event flow	● Copy the public key after starting the instance ● Write down your current token balance by executing the command `accounts` and find the balance for your public key ● Execute `destroysupply` and follow the next steps: ● At "Enter block hash where the SKU was created", fill in the 'block hash' value, found in the results of the `skus` command from testcase 3. ● At "Enter the index of the transaction where the SKU resides" fill in "1" ● At "Specify the amount of supply to destroy", fill in "1" ● At "Enter optional data", press enter ● Confirm that the transaction was submitted to the transactionpool successfully ● Add the transaction to the next block by executing the command `startmining` ● Once a block has been found, stop mining by using the command `stopmining` ● Verify that the action costed 1 token with command `accounts` ● Verify that the global supply decreased with 1 by using the command `skus` (check the last result in this list)
Expected result	The transaction was added to the transactionpool successfully. The transaction was included in a block. The balance of the public key is (new amount of blocks * 5000) – 1 because the action costed 1 token. After using the `skus` command, the last transaction shows the total global supply decreased with 1 in comparison with the transaction before the last one.
Actual result	
Analysis	
Dependencies on other testcases	Testcase 3

1.4 Transfer supply for SKU

This core functionality handles the transfer of supply between companies or other entities that represent a public key.

Testcase identifier	6. Transfer supply for SKU
Test items	Transferring existing supply for an existing SKU to another public key.
Begin situation	The test setup as described on page 2.
Event flow	● Copy the public key after starting the instance ● Write down your current token balance by executing the command `accounts` and find the balance for your public key ● Execute `transfersupply` and follow the next steps: ● At "Enter block hash where the SKU was created", fill in the 'block hash' value, found in the results of the `skus` command from testcase 3. ● At "Enter the index of the transaction where the SKU resides" fill in "1" ● At "Enter the receiver's public key", fill in "warehouse" ● At "Specify the amount to transfer", fill in "3" ● At "Enter optional data", press enter ● Confirm that the transaction was submitted to the transactionpool successfully ● Add the transaction to the next block by executing the command `startmining` ● Once a block has been found, stop mining by using the command `stopmining` ● Verify that the action costed 1 token with command `accounts` ● Verify that three bicycles were transferred by using the `skus` command
Expected result	The transaction was added to the transactionpool successfully. The transaction was included in a block. The balance of the public key is (new amount of blocks * 5000) – 1 because the action costed 1 token. After using the `skus` command, confirm that three bicycles were transferred to "warehouse".
Actual result	
Analysis	
Dependencies on other testcases	Testcase 3