MSC-Tools Documentation

Release .12

Adam Chamely

May 07, 2014

Contents:

CHAPTER 1

Introduction

Welcome to the world of Mastercoin.

This guide will walk you through the process of setting up, installing and running the Mastercoin-tools.

1.1 Consensus

A quick note on Consensus.

The official mastercoin state is defined by mastercoin-tools code result.

Periodically checking the consensus on the Masterchain Consensus Report can alert on potential differences among the implimentations.

1.2 Installed Items

Here is a breakdown of everything that is needed/installed to support Mastercoin-tools

- git
- make
- python-simplejson
- python-git
- python-pip
  - ecdsa==0.10
  - pycoin==0.25
  - pybitcointools==1.1
- git
- build-essential
- autoconf
- libtool
- libboost-all-dev
- pkg-config

- libcurl4-openssl-dev
- libleveldb-dev
- libzmq-dev
- libconfig++-dev
- libncurses5-dev
- sx (d9b566e)
- obelisk (4962e2c)
- libbitcoin (4962e2c)
- libwallet (4962e2c)
- mastercoin-tools

CHAPTER 2

Installation/Setup

2.1 Prerequisites

The msc-tools leverage an existing obelisk server. If you wish to know more about obelisk or run your own see the *Obelisk page*. During installation the script will prompt you if you have one. If not you can come back later and update your ~/.sx.cfg file with the correct details.

Need a server? Try checking UN Systems wiki *Note: At present most of the listed servers seem to have issues except obelisk.bysh.me:9091*

2.2 Recommended System Requirements

- 12Gb+ Disk space
- For every Obelisk Worker you plan to run add ~35-40Gb for block chain storage
- 1 Gig+ Ram (Amazon base EC2 instance with 512 will fail to build)
- Use a Tested Environment

2.3 Tested Environments

The installation utility and all components have been tested in the following environments:

- ubuntu-server-13.10 (32 | 64)

2.4 Installing (auto)

An installation script has been provided that automates the installation process. It will prompt for obelisk server details and can be run with the following commands

```
git clone https://github.com/mastercoin-MSC/install-msc.git
cd install-msc
sudo bash install-msc.sh
```

Optionally you can provide the obelisk server details on the cli

```
sudo bash install-msc.sh -os tcp://your.obelisk.server.org:9091
```

2.5 Installing (manual)

If you want to manually install all of the components you can do so with the following commands.

```
#Update the apt-get packages
sudo apt-get update

#install required supporting packages:
sudo apt-get -y install git python-simplejson python-git python-pip
sudo apt-get -y install make
sudo apt-get -y install git build-essential autoconf libtool libboost-all-dev pkg-config libcurl4-ope
sudo pip install -r pip.packages

#Install SX using the modified installation script
#Note, this script installs specific revisions of the sx components known to work with mastercoin-too
sudo bash install-sx.sh

#Download the mastercoin-tools
git clone https://github.com/mastercoin-MSC/mastercoin-tools.git

#copy the scripts and app.sh wrapper for mastercoin tools to the mastercoin-tools directory
cp install-msc/res/app.sh mastercoin-tools/
cp install-msc/scripts/* mastercoin-tools/

#update ~/.sx.cfg with an obelisk server details
# ~/.sx.cfg Sample file.
#service = "tcp://162.243.29.201:9091"

#create the mastercoin tools data directory
mkdir -p /var/lib/mastercoin-tools

#bootstrap files needed to start the parsing engine
tar xzf install-msc/res/bootstrap.tgz -C /var/lib/mastercoin-tools


#Mastercoin-tools directory needs to have permissions set to the user who will run it
sudo chown -R `logname`:`logname` mastercoin-tools
sudo chown -R `logname`:`logname`  /var/lib/mastercoin-tools
```

CHAPTER 3

Running

Included with mastercoin-tools are 2 different methods for downloading/updating the blockchain information.

3.1 app.sh

Continuous running application that you can start in a screen session. It will download/process the entire block chain and then sleep for 60 secs before checking for updates.:

```
screen -R msc
./app.sh
ctrl+a, d  <disconnect screen>

reconnect at anytime with
screen -R msc
```

3.2 msc_cron.sh

Alternatively you can schedule a cron job to execute the msc_cron.sh utility at your predetermined time.

CHAPTER 4

Tools

List of tools included with the installation and how to use them

4.1 msc_createtx.py

4.1.1 Purpose:

Used to create, sign and/or send a Masterprotocol currency transaction

4.1.2 Checks:

Checks from address to make sure it has:

- Enough BTC to create/send the transaction
  - Note: To avoid potential double spends all unspent TX used to create a new TX are tracked/locked for 10 Blocks from use. It is recommended, when offline signing, to make sure you broadcast within this timeframe.
- Balance of the CurrencyID to make sure it has enough to send msc_send_amt
  - Balance is checked using 2 online resources (Masterchest.info and Omniwallet)

4.1.3 Inputs:

Takes json input via STDIN for the following variables:

- transaction_from: The Public Address of the Sender
- transaction_to: The Public address of the Receipiant
- currency_id: Currency ID to send. 1 for MSC, 2 for TMSC
- property_type: 1 for indivisible currency, 2 for divisible (MSC/TMSC are 2, Maidsafecoins are 1)
- send_amt: The amount of the Currency ID to send
- from_private_key: Base58 Private Key of the sender's Public Address *
  - (Note: Should start with the number 5)
- broadcast: Create, Sign and/or Broadcast Tx.

  - 0 - Create the Unsigned TX file only
  - 1 - Create and Sign the TX file
  - 2 - Create, Sign and Broadcast the TX file
- clean: Clean up any of the tx files created.‘‘*‘‘
  - 0 - Keep all Tx files created
  - 1 - Remove only the intersigned Tx files. (Leaves the original unsigned Tx and the signed Tx)
  - 2 - Remove all unsigned Tx files. Leaves only the signed Tx file that can be broadcast.
  - 3 - Remove all Tx files. Signed and unsigned, make sure you have broadcast the Tx before you do this.
- * Only required if you are signing/broadcasting the tx file and can be omitted if just creating unsigned file.*

The json takes the following format:

```
{
  "transaction_from": "{{Public from Address}}",
  "transaction_to": "{{Public to Address}}",
  "currency_id": {{1 for MSC, 2 for TMSC}},
  "send_amt": {{amount to send}},
  "property_type": {{1 for indivisible currency, 2 for divisible (MSC/TMSC are 2, Maidsafecoins are 1
  "broadcast": {{1 to create and broadcast or 0 to just create}},
  "from_private_key": "{{private key for signing}}",
  "clean": {{0 -keep all tx files, 1 -remove intersigned tx, 2 -remove all unsigned, 3 -remove all}}
}
```

Ex:

Note: for security the following was a brand new empty wallet. You should replace it's details with your own applicable info:

```
{
  "transaction_from": "1GGJMZoaxYMS4jsiLwPVbofe5YJyM6ER2i",
  "transaction_to": "19hf8QEkD3GR7NhUrujWXRg6e4gsHUTysp",
  "currency_id": 1,
  "send_amt": 5.1,
  "property_type": 2,
  "from_private_key": "5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhrbGJPgoWb",
  "broadcast": 1,
  "clean": 1
}
```

For reference, here is what the brainwallet.org generator page for the above address looks like. Take note of the ‘Uncompressed/Compressed’ option

4.1.4 Output:

Will return a json formated output. Errors will be returned with json that contains

```
{
   "status": "Status message",
   "error": "error details",
   "fix": "Corrective action to resolve the issue"
}
```

Successful run will return json that contains:

```
{
  "status": "Broadcast/Created/Signed status",
  "valid_check": "Validity check of signed file",
  "hash": "Hash of the tx",
  "st_file": "location/name of the signed tx file"
}
```

4.1.5 Running:

Standalone running/testing can be done by creating a json file (see input details or example_send.json for structure) You can execute/run the program with:

```
cat your_file.json | python msc_createtx.py
```

4.2 msc-sxsend.py

4.2.1 Purpose:

DEPRECIATED, Please see *msc_createtx.py*

Used to create (and/or send) a Mastercoin transaction

4.2.2 Checks:

Checks from address to make sure it has:

- Enough BTC to create/send the transaction
- Balance of the CurrencyID to make sure it has enough to send msc_send_amt
  - Balance is checked using the *msc-balance.py* script

4.2.3 Inputs:

Takes json input via STDIN for the following variables:

- transaction_from: The Public Address of the Sender
- transaction_to: The Public address of the Receipiant
- currency_id: Currency ID to send. 1 for MSC, 2 for TMSC
- msc_send_amt: The amount of the Currency ID to send
- property_type: 1 for indivisible currency, 2 for divisible (MSC/TMSC are 2, Maidsafecoins are 1)
- from_private_key: Base58 Private Key of the sender's Public Address (Note: Should start with 5)
- broadcast: Create and/or Broadcast Tx. 1 to create and broadcast or 0 to just create
- clean: Clean up any of the tx files created.
  - 0 - Keep all Tx files created
  - 1 - Remove only the intersigned Tx files. (Leaves the original unsigned Tx and the signed Tx)

- 2 - Remove all unsigned Tx files. Will leave only the signed Tx file that can be broadcast to the network.
- 3 - Remove all Tx files. Signed and unsigned, make sure you have broadcast the Tx before you do this.

The json takes the following format:

```
{
  "transaction_from": "{{Public from Address}}",
  "transaction_to": "{{Public to Address}}",
  "currency_id": {{1 for MSC, 2 for TMSC}},
  "msc_send_amt": {{amount to send}},
  "property_type": {{1 for indivisible currency, 2 for divisible (MSC/TMSC are 2, Maidsafecoins are 1
  "from_private_key": "{{private key for signing}}",
  "broadcast": {{1 to create and broadcast or 0 to just create}},
  "clean": {{0 -keep all tx files, 1 -remove intersigned tx, 2 -remove all unsigned, 3 -remove all}}
}
```

Ex:

Note: for security the following was a brand new empty wallet. You should replace it's details with your own applicable info:

```
{
  "transaction_from": "1GGJMZoaxYMS4jsiLwPVbofe5YJyM6ER2i",
  "transaction_to": "19hf8QEkD3GR7NhUrujWXRg6e4gsHUTysp",
  "currency_id": 1,
  "msc_send_amt": 5.1,
  "property_type": 2,
  "from_private_key": "5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhrbGJPgoWb",
  "broadcast": 1,
  "clean": 1
}
```

For reference, here is what the brainwallet.org generator page for the above address looks like. Take note of the 'Uncompressed/Compressed' option

4.2.4 Output:

Will return a json formated output. Errors will be returned with json that contains

```
{
   "status": "Status message",
   "error": "error details",
   "fix": "Corrective action to resolve the issue"
}
```

Successful run will return json that contains:

```
{
  "status": "Broadcast/Created status",
  "valid_check": "Validity check of signed file",
  "hash": "Hash of the tx",
  "st_file": "location/name of the signed tx file"
}
```

4.2.5 Running:

Standalone running/testing can be done by creating a json file (see input details or example_send.json for structure) You can execute/run the program with:

```
cat your_file.json | python msc_sxsend.py
```

4.3 msc-txcreate.py

4.3.1 Purpose:

DEPRECIATED, Please see *msc_createtx.py*

Used to create an unsigned Mastercoin transaction

4.3.2 Checks:

Checks from address to make sure it has:

- Enough BTC to create/send the transaction
- Balance of the CurrencyID to make sure it has enough to send msc_send_amt
  - Balance is checked using the *msc-balance.py* script

4.3.3 Inputs:

Takes json input via STDIN for the following variables:

- transaction_from: The Public Address of the Sender
- transaction_to: The Public address of the Receipiant
- currency_id: Currency ID to send. 1 for MSC, 2 for TMSC
- msc_send_amt: The amount of the Currency ID to send
- property_type: 1 for indivisible currency, 2 for divisible (MSC/TMSC are 2, Maidsafecoins are 1)

The json takes the following format:

```
{
  "transaction_from": "{{Public from Address}}",
  "transaction_to": "{{Public to Address}}",
  "currency_id": {{1 for MSC, 2 for TMSC}},
  "msc_send_amt": {{amount to send}},
  "property_type": {{1 for indivisible currency, 2 for divisible (MSC/TMSC are 2, Maidsafecoins are 1
}
```

Ex:

Note: for security the following was a brand new empty wallet. You should replace it's details with your own applicable info:

```
{
  "transaction_from": "1GGJMZoaxYMS4jsiLwPVbofe5YJyM6ER2i",
  "transaction_to": "19hf8QEkD3GR7NhUrujWXRg6e4gsHUTysp",
  "currency_id": 1,
  "msc_send_amt": 5.1
}
```

4.3.4 Output:

Will return a json formated output. Errors will be returned with json that contains

```
{
   "status": "Status message",
   "error": "error details",
   "fix": "Corrective action to resolve the issue"
}
```

Successful run will return json that contains:

```
{
  "status": "Created status",
  "st_file": "location/name of the unsigned tx file"
}
```

4.3.5 Running:

Standalone running/testing can be done by creating a json file (see input details or example_send.json for structure) You can execute/run the program with:

```
cat your_file.json | python msc-txcreate.py
```

4.4 msc-balance.py

4.4.1 Purpose:

Used to get the Mastercoin balance of an address

4.4.2 Requirements:

This script leverages the existing mastercoin tools parsed/validated output. Mastercoin tools should be installed and fully updated with the Mastercoin Data in:

```
/var/lib/mastercoin-tools/mastercoin_verify/addresses/
```

4.4.3 Checks:

Will check/return the date of the parsed date as listed in

```
/var/lib/mastercoin-tools/www/revision.json
```

4.4.4 Inputs:

Takes json input via STDIN for the following variables:

- address: The address you want to check the balance for
- currency_id: The currency you want the balance for
  - 1 - Mastercoin
  - 2 - Test Mastercoins

The json takes the following format:

```
{
  "address": "{{Address to check}}",
  "currency_id": {{1 for MSC, 2 for TMSC}}
}
```

Ex:

```
{
  "address": "1CMauYumpA7YG8i4cPod8FadRLK95HxSob",
  "currency_id": 1
}
```

4.4.5 Output:

Will return a json formated output

Completed run will return json that contains:

```
{
  "address": "Address checked",
  "currency_id": "Currency checked",
  "balance": "Balance or error message",
  "balancetime": "Time in GMT human readable",
  "epochtime": "Balance Timestamp in GMT epoch"
}
```

Note: If the revision file or currency address files are missing the time is omitted and an error message is returned for balance.

4.4.6 Running:

Standalone running/testing can be done by creating a json file (see input details or example_balance.json for structure) You can execute/run the program with:

```
cat your_file.json | python msc-balance.py
```

4.5 getConsensusMSC.py

4.5.1 Purpose:

Used to get the consensus of local installation with Online sites *Note: The final consensus authority is defined by the mastercoin tools code result.* Masterchain Consensus Report

4.5.2 Requirements:

This script leverages the existing mastercoin tools parsed/validated output. Mastercoin tools should be installed and fully updated with the Mastercoin Data in:

```
/var/lib/mastercoin-tools/mastercoin_verify/addresses/
```

4.5.3 Inputs:

Takes json input via STDIN for the sites you wish to validate consensus against: *Note: At present generates consensus output for Currency ID 1 (MSC) only.*

- site: The sites to compare local results against

The json takes the following format:

```
{ "sites":
    [
      "http://masterchain.info/mastercoin_verify/addresses/0",
      "https://masterchest.info/mastercoin_verify/addresses.aspx",
      "http://mymastercoins.com/jaddress.aspx"
    ]
}
```

4.5.4 Output:

Will return a json formated output array of address not in consensus

For each address not in Consensus, completed run will return balance of that address for each site checked in json format:

```
{
  "consensus": Number Representing Consensus Rating,
  "data":[
     [
        {
           "balance": Number Representing Current balance for the site checked,
           "site":"Site/Data Source name",
           "address":"address not in consensus"
        },
        {
          ... data in format of ^ for each site when address is not in consensus
        }
     ],
     [
        ... 2nd address (if exists) not in consensus in format ^^^
     ]
  ]
}
```

4.5.5 Running:

Running by creating a json file (see input details) for sites you wish to check or use the provided getConsensus.json You can execute/run the program with:

```
cat getConsensus.json | python getConsensusMSC.py
```

CHAPTER 5

Obelisk

Some Information and Instructions taken from Libbitcoin Obelisk Quickstart

5.1 What is Obelisk

Obelisk is a scalable blockchain query infrastructure which allows you to maintain your own copies of the blockchain for parsing/data interaction. Mastercoin tools needs/uses an obelisk server to query the blockchain and create/parse Mastercoin Transactions. There are some public obelisk servers available already on the web, however if you wish to run your own server in house this guide will help you get started. For the purposes of this document there are three relevant parts:

- *Server*
- *Workers*
- *Clients*

5.2 Installation

By default Obelisk is installed when you run the Mastercoin-tools installer. It is part of the *sx dependencies/installation package*.

5.3 Configuration

The default Obelisk Configuration files are stored in

```
/etc/obelisk
```

There are two files

- balancer.cfg
- worker.cfg

5.3.1 balancer.cfg

Allows you to configure the port clients and workers will connect to:

- The default port for clients is *9091*.
- The default port for workers is *9092*.

You may modify these to suit your environment or leave them alone.

5.3.2 worker.cfg

This file contains all the information an obelisk workers needs to connect/respond to an obelisk server.

The default settings should work just fine for a normal installation. If you have changed the 'client port' in the *balancer.cfg* or you are running obelisk workers on seperate machines you will need to update the service definition with your updated/relevant details:

```
service = "tcp://localhost:9092"
```

5.4 Server

The obelisk server is what handles the interaction between the client requests and the workers response. It's entire operation is run by a program called: *obbalancer*.

Obbalancer uses the *balancer.cfg* configuration to listen for workers and clients.

There are two methods for running the server: Screen or Daemon.

5.4.1 Screen

You can run the obbalancer in a screen session. This is easy to get started but may not be the most robust method.:

```
screen -S obbalancer
obbalancer
```

Disconnect from the screen session with:

```
CTRL+A  D
```

You can reattach to the screen session with:

```
screen -r obbalancer
```

to check on it's progress/status

5.4.2 Daemon

The obelisk source includes an init.d script you can use. It is located in the *<install-src>/obelisk-git/scripts/init.d/* directory.

On a default installation this should be

```
/usr/local/src/obelisk-git/scripts/init.d/obbalancer
```

You will need to copy the *obbalancer* script to your /etc/init.d/ directory and set its permissions for executing:

```
sudo cp /usr/local/src/obelisk-git/scripts/init.d/obbalancer /etc/init.d/
sudo chmod 755 /etc/init.d/obbalancer
```

The obbalancer init.d script uses the username *ob*.

If it doesn't exist create a limited permissions user with this name or update the script line shown below with the username you wish it to use:

```
DAEMON_USER=ob
```

Once the script is setup you can start it with:

```
/etc/init.d/obbalancer start
```

If you wish the script to start on system startup you can also run:

```
update-rc.d obbalancer defauls
```

5.5 Workers

These are the workhorses of the obelisk server.

Each server leverages one or more connected workers to query the blockchain information they have. You can run multiple workers on the same machine or spread them out and run them from multiple machines for redundancy. Each worker uses/maintains it's own copy of the block chain database.

5.5.1 Initial-Setup

Note: Workers CAN NOT share the same data directories. Each worker needs it's own directory to store it's files/information.

Create and initialize a blockchain database for each worker

```
mkdir worker.1/
cd worker.1/
mkdir blockchain/
sx initchain blockchain/
```

5.5.2 Bootstraping Data

If you have a bitcoind bootstrap.dat, then you can bootstrap a blockchain. See /usr/local/libbitcoin/tools/ (run 'sudo make' and see the bootstrap tool).

Alternatively, once 1 worker is up and running/fully synced, you can:

- Stop that workers 'obworker'
- copy the blockchain/ directory to the new workers directory
- start the original worker and then the new worker.

5.5.3 Running

Once the worker has been setup. You can start it using obworker. It is recommended that workers be run in a screen session for unattendted operation

```
cd worker.1/
screen -S worker.1
obworker
```

You can detach from the screen session with:

```
CTRL+A D
```

You can also reattach to the screen to check on the status with:

```
screen -r worker.1
```

Repeat this process for each worker you wish to start.

Working Notes/Tips:

- Press CTRL-C and wait if you want to stop the worker.
- You can see the output using 'tail -f debug.log' in each workers directory.
- Running multiple workers is good for redundancy in case one crashes or has problems.

5.6 Clients

The client is who/what is actually requesting the information.

In Mastercoin tools the client is the local installation of sx which queries the obelisk server for blockchain information. Clients can connect to an obelisk server on the *configured port*. For proper operation the Obelisk server should be setup, running, and have fully syned workers connected to it.

If you are using a local installation of the obelisk server make sure to update the sx configuration file

```
~/.sx.cfg
```

Run a few test commands with sx to confirm operation

```
sx fetch-last-height     :Returns current height the obelisk server knows
```

or

```
sx balance <btc address>    :Returns balance in satoshis
```

CHAPTER 6

Troubleshooting

Having issues? Things not working as expected?

Here are a few 'Gotchyas' that we've encountered and what to check/how to fix them.

6.1 Permissions

One of the first things to check is folder permissions. The installer tries to figure out what user is running the installer and set the permissions for the folders it creates appropriately. If this does not happen properly the user you run "app.sh" as may not have permission to access the necessary folders.

6.1.1 Items to Check

There are 2 main items that need their permissions checked:

Data directory

```
/var/lib/mastercoin-tools
```

Tools directory

```
~/mastercoin-tools
```

6.1.2 Fix

These need to be owned by the user who is going to run "app.sh":

```
sudo chown -R <youruser>:<youruser> /var/lib/mastercoin-tools
sudo chown -R <youruser>:<youruser> /home/<youruser>/mastercoin-tools
```

6.2 SX Settimgs

One of the other issues we've seen is when sx 'Hangs' or just fails to respond. Also visible if you are watching the system processes (command below) and notice it not moving/changing from the same command

```
watch 'ps aux | grep -i -e sx -e sleep | grep -v grep'
```

6.2.1 Items to Check

- The user running app.sh or calling sx commands needs to have a/the sx config file in the home directory of the user running "app.sh"

```
/home/<youruser>/.sx.cfg
```

- Also check to make sure the sx server is actually responding

```
#should return the block height number of the obelisk server
sx fetch-last-height

#should return the block height number of blockchain.info
sx bci-fetch-last-height
```

6.2.2 Fix

- Make sure you are running "app.sh" as the user who has the sx config file in their home directory
- Try a different sx server. We have had decent experience using: *obelisk.bysh.me:9091*

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*