Mastercoin Documentation

Release .12

Adam Chamely

July 01, 2014

Contents

Contents:

CHAPTER 1

Mastercore

Contents:

1.1 Introduction

Welcome to the world of Mastercoin.

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

1.1.1 Consensus

A quick note on Consensus.

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

This will soon be updated to the Mastercore project.

Until that time Periodically checking the consensus on the Masterchest Consensus Report can alert on potential differences among the implimentations.

1.1.2 Installed Items

Here is a breakdown of everything that is needed/installed to support Mastercore

CHAPTER 2

Mastercoin-Tools

Contents:

2.1 Introduction

Welcome to the world of Mastercoin.

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

2.1.1 Consensus

A quick note on Consensus.

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

This will soon be updated to the Mastercore project.

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

2.1.2 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

2.2 Installation/Setup

2.2.1 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*

2.2.2 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

2.2.3 Tested Environments

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

- ubuntu-server-13.10 (32 | 64)

2.2.4 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
```

2.2.5 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
```

2.3 Running

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

2.3.1 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
```

2.3.2 msc_cron.sh

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

2.4 Tools

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

- *Create Asset Issuance (generateTX50_SP.py)*
- *Create Crowdsale (generateTX51_SP.py)*
- *Close Crowdsale early (generateTX53_SP.py)*
- *Create/Send Tx (msc_createtx.py)*

2.4.1 generateTX50_SP.py

Back to Top

Purpose:

Create a New Smart Property

Creates the raw transacation that when broadcasts will create a New Smart property. Located in mastercoin-tools/scripts

Requirements:

- Python 2.7.6
- Fully synced *Bitcoind* node (can be local or remote)
- Private key of the issuing address in bitcoind

Inputs:

Takes json input via STDIN for the following variables:

- transacation_type: type int - representing the tx type (50)
- ecosystem: type int - 1 For production deployment, 2 for test deployments

- property_type: type int - 1 for indivisible currency, 2 for divisible (MSC/TMSC are 2, Maidsafecoins are 1)
- previous_property_id: type int - If you are replacing a previous Smart property enter the currency ID here. Otherwise enter 0
- property_category: type string - Main category for your property (Suggested *Categories*)
- property_subcategory: type string - Sub category for your property (See listing on category)
- property_name: type string - Name of your Coin/Token/Property
- property_url: type string - Short url users can go to for more information about the Coin/Token/Property you are creating.
- property_data: type string - Brief description about what your Coin/Token/Property is for
- number_properties: type int - The number of Coins/Tokens/Properties you wish to issue/create.
- transacation_from: type base58 - Your sending address
- from_private_key: type base58 - Private Key of the sending address *
  - (Note: Should start with the number 5)

The json takes the following format:

```
{
          "transaction_type": 'type: int, ex: 50',
          "ecosystem": 'type: int, ex: 1',
          "property_type": 'type: int, ex: 1',
          "previous_property_id": 'type: int, ex: 3',
          "property_category": "type: string, ex: Testing",
          "property_subcategory": "type: string, ex: Testing Smart Property",
          "property_name": "type: string, ex: Test Property 1",
          "property_url": "type: string, ex: mastercoinfoundation.org",
          "property_data": "type: string, ex: Test Data",
          "number_properties": 'type: int, ex: 10',
          "transaction_from": "type: base58",
          "from_private_key": "type: base58"
}
```

Ex:

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

```
{
        "transaction_type": 50,
        "ecosystem": 2,
        "property_type": 2,
        "previous_property_id": 0,
        "property_category": "Testing",
        "property_subcategory": "Smart Property Test Sequence 1",
        "property_name": "Doubloons",
        "property_url": "http://tinyurl/dubloons",
        "property_data": "Test Issuing a new Currency",
        "number_properties": 1000,
        "transaction_from": "1GGJMZoaxYMS4jsiLwPVbofe5YJyM6ER2i",
        "from_private_key": "5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhrbGJPgoWb"
}
```

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

Output and Running:

You can execute/run the program with:

```
cat your_file.json | python generateTX50_SP.py
```

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

```
{
       "status": "NOT OK", "fix": "bitcoind importprivkey 5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhr
       "error": "Couldn't find address in wallet, please run 'fix' on the machine"
}
```

In this case you need to import the private key into bitcoind and then run again.

> bitcoind importprivkey 5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhrbGJPgoWb imported_1397503463 false

Successful run will return json that contains the raw hex:

```
{
     "rawtransaction": {
     "hex": "0100000001e604......90b53ae00000000",
     "complete": true
     }
}
```

Once you have the completed successful raw hex send the transaction by copying and pasting that hex string (without its quotes) as an argument to bitcoind sendrawtransaction:

```
bitcoind sendrawtransaction 0100000001e604......90b53ae00000000
```

The output will be the transaction hash ID. Check http://blockchain.info to see the status of the transation. You can also find the transaction via blockchain.info's page for the issuing address.

2.4.2 generateTX51_SP.py

Back to Top

Purpose:

Create a New Crowdsale

Creates the raw transacation that when broadcasts will create a New Crowdsale. Located in mastercoin-tools/scripts

Requirements:

- Python 2.7.6
- Fully synced *Bitcoind* node (can be local or remote)
- Private key of the issuing address in bitcoind

Inputs:

Takes json input via STDIN for the following variables:

- transacation_type: type int - representing the tx type (51)
- ecosystem: type int - 1 For production deployment, 2 for test deployments
- property_type: type int - 1 for indivisible currency, 2 for divisible (MSC/TMSC are 2, Maidsafecoins are 1)
- previous_property_id: type int - If you are replacing a previous Smart property enter the currency ID here. Otherwise enter 0
- property_category: type string - Main category for your property (Suggested *Categories*)
- property_subcategory: type string - Sub category for your property (See listing on category)
- property_name: type string - Name of your Coin/Token/Property
- property_url: type string - Short url users can go to for more information about the Coin/Token/Property you are creating.
- property_data: type string - Brief description about what your Coin/Token/Property is for
- currency_identifier: type int - The currency ID to accept for the crowdsale (what coin investors have to send) ex: 2 (Test MSC)
- number_properties: type int - The number of Coins/Tokens/Properties you wish to issue/create.
- deadline: type int - Time in UTC the Crowdsale should finish/close/stop.
- earlybird_bonus: type int - Percent extra/week investor gets when investing before the deadline.
- percentage_for_issuer: type int - Percent credited to the issuer for every investment. (You get this percent per token generate for investors)
- transacation_from: type base58 - Your sending address
- from_private_key: type base58 - Private Key of the sending address *
  - (Note: Should start with the number 5)

The json takes the following format:

```
{
          "transaction_type": 'type: int, ex: 51',
          "ecosystem": 'type: int, ex: 1',
          "property_type": 'type: int, ex: 1',
          "previous_property_id": 'type: int, ex: 3',
          "property_category": "type: string, ex: Testing",
          "property_subcategory": "type: string, ex: Testing Smart Property",
          "property_name": "type: string, ex: Test Property 1",
          "property_url": "type: string, ex: mastercoinfoundation.org",
          "property_data": "type: string, ex: Test Data",
          "currency_identifier_desired": 'type int, ex: 2',
          "number_properties": 'type: int, ex: 10',
          "deadline": 'type: int, ex: 7731439200',
          "earlybird_bonus": 'type: int, ex: 10',
          "percentage_for_issuer": 'type: int, ex: 12',
          "transaction_from": "type: base58",
          "from_private_key": "type: base58"
}
```

Ex:

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

```
{
        "transaction_type": 51,
        "ecosystem": 2,
        "property_type": 2,
        "previous_property_id": 0,
        "property_category": "Testing",
        "property_subcategory": "Smart Property Test Sequence 1",
        "property_name": "Doubloons-Sale",
        "property_url": "http://tinyurl/dubloons",
        "property_data": "Test Issuing a new Currency",
        "currency_identifier_desired": 2,
        "number_properties": 1000,
        "deadline": 1397869200,
        "earlybird_bonus": 0,
        "percentage_for_issuer": 0,
        "transaction_from": "1GGJMZoaxYMS4jsiLwPVbofe5YJyM6ER2i",
        "from_private_key": "5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhrbGJPgoWb"
}
```

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

Output and Running:

You can execute/run the program with:

```
cat your_file.json | python generateTX50_SP.py
```

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

```
{
       "status": "NOT OK", "fix": "bitcoind importprivkey 5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjh
       "error": "Couldn't find address in wallet, please run 'fix' on the machine"
}
```

In this case you need to import the private key into bitcoind and then run again.

> bitcoind importprivkey 5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhrbGJPgoWb imported_1397503463 false

Successful run will return json that contains the raw hex:

```
{
     "rawtransaction": {
     "hex": "0100000001e604......90b53ae00000000",
     "complete": true
     }
}
```

Once you have the completed successful raw hex send the transaction by copying and pasting that hex string (without its quotes) as an argument to bitcoind sendrawtransaction:

```
bitcoind sendrawtransaction 0100000001e604......90b53ae00000000
```

The output will be the transaction hash ID. Check http://blockchain.info to see the status of the transation. You can also find the transaction via blockchain.info's page for the issuing address.

2.4.3 generateTX53_SP.py

Back to Top

Purpose:

Close an existing Crowdsale early.

Creates the raw transacation that when broadcasts will close the current crowdsale Crowdsale immediately. Located in mastercoin-tools/scripts

Requirements:

- Python 2.7.6
- Fully synced *Bitcoind* node (can be local or remote)
- Private key of the issuing address in bitcoind

Inputs:

Takes json input via STDIN for the following variables:

- transacation_type: type int - representing the tx type (53)
- previous_property_id: type int - If you are replacing a previous Smart property enter the currency ID here. Otherwise enter 0
- transacation_from: type base58 - Your sending address
- from_private_key: type base58 - Private Key of the sending address *
  - (Note: Should start with the number 5)

The json takes the following format:

```
{
          "transaction_type": 'type: int, ex: 53',
          "property_type": 'type: int, ex: 1',
          "transaction_from": "type: base58",
          "from_private_key": "type: base58"
}
```

Ex:

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

```
{
        "transaction_type": 53,
        "property_type": 4,
        "transaction_from": "1GGJMZoaxYMS4jsiLwPVbofe5YJyM6ER2i",
        "from_private_key": "5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhrbGJPgoWb"
}
```

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

Output and Running:

You can execute/run the program with:

```
cat your_file.json | python generateTX53_SP.py
```

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

```
{
  "status": "NOT OK", "fix": "bitcoind importprivkey 5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhrbGJP
  "error": "Couldn't find address in wallet, please run 'fix' on the machine"
}
```

In this case you need to import the private key into bitcoind and then run again.

> bitcoind importprivkey 5JXxd7qecXrzd9hJGdJsBnwkfJauHxVqbqRmBqQUjhrbGJPgoWb imported_1397503463 false

Successful run will return json that contains the raw hex:

```
{
    "rawtransaction": {
    "hex": "0100000001e604......90b53ae00000000",
    "complete": true
     }
}
```

Once you have the completed successful raw hex send the transaction by copying and pasting that hex string (without its quotes) as an argument to bitcoind sendrawtransaction:

```
bitcoind sendrawtransaction 0100000001e604......90b53ae00000000
```

The output will be the transaction hash ID. Check http://blockchain.info to see the status of the transation. You can also find the transaction via blockchain.info's page for the issuing address.

2.4.4 msc_createtx.py

Back to Top

Purpose:

Used to create, sign and/or send a Masterprotocol currency transaction. Located in mastercoin-tools/scripts

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)

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

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"
}
```

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
```

2.4.5 msc-sxsend.py

Back to Top

Purpose:

DEPRECIATED, Please see *msc_createtx.py*

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

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

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

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"
}
```

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
```

2.4.6 msc-txcreate.py

Back to Top

Purpose:

DEPRECIATED, Please see *msc_createtx.py*

Used to create an unsigned Mastercoin transaction

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

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
}
```

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"
}
```

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
```

2.4.7 msc-balance.py

Back to Top

Purpose:

Used to get the Mastercoin balance of an address

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/`

Checks:

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

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

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
}
```

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.

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
```

2.4.8 getConsensusMSC.py

Back to Top

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

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/
```

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"
    ]
}
```

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 ^^^
     ]
  ]
}
```

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
```

2.5 Obelisk

Some Information and Instructions taken from Libbitcoin Obelisk Quickstart

2.5.1 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*

2.5.2 Installation

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

2.5.3 Configuration

The default Obelisk Configuration files are stored in

```
/etc/obelisk
```

There are two files

- balancer.cfg
- worker.cfg

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.

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"
```

2.5.4 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.

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

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
```

2.5.5 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.

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/
```

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.

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.

2.5.6 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
```

2.6 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.

2.6.1 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.

Items to Check

There are 2 main items that need their permissions checked:

Data directory

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

Tools directory

```
~/mastercoin-tools
```

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
```

2.6.2 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'
```

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
```

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*

2.6.3 Bitcoind

Bitcoind is used by the Smart Property Scripts.

- *Asset Issuance (generateTX50_SP.py)*
- *Crowdsale (generateTX51_SP.py)*
- *Close Crowdsale (generateTX53_SP.py)*

It can either be run locally or on a remote machine. The scripts will first attempt to connect to a locally running instance of bitcoind, if that fails then they will look for and and try to connect to a remote instance. It will load connection information from

```
home/<username>/.bitcoin/bitcoin.conf
```

bitcoin.conf is a 4 line file with the following values (these should be taken straigt from the configuration of your running bitcoind):

- rpcuser - The username defined in your bitcoin.conf
- rpcpassword - The password defined in your bitcoin.conf
- rpcconnect - The ip adress of the remote bitcoind machine
- rpcport - (optional) The port its running on. (If not specified defaults to 8332)

2.7 Categories

List of suggested categories and subcategories for use with *Crowdsales* and *Asset Issuances*:

```
{
   "Accommodation and food service activities": [
       "Accommodation",
       "Food and beverage service activities",
       "Other"
   ],
   "Activities of extraterritorial organizations and bodies": [
       "Activities of extraterritorial organizations and bodies",
       "Other"
   ],
   "Activities of households as employers; undifferentiated goods- and services-producing activities
       "Activities of households as employers of domestic personnel",
       "Undifferentiated goods- and services-producing activities of private households for own use",
       "Other"
   ],
   "Administrative and support service activities": [
       "Rental and leasing activities",
       "Employment activities",
       "Travel agency, tour operator, reservation service and related activities",
       "Security and investigation activities",
       "Services to buildings and landscape activities",
       "Office administrative, office support and other business support activities",
       "Other"
   ],
   "Agriculture, forestry and fishing": [
       "Crop and animal production, hunting and related service activities",
       "Forestry and logging",
       "Fishing and aquaculture",
```

```
        "Other"
    ],
    "Arts, entertainment and recreation": [
        "Creative, arts and entertainment activities",
        "Libraries, archives, museums and other cultural activities",
        "Gambling and betting activities",
        "Sports activities and amusement and recreation activities",
        "Other"
    ],
    "Construction": [
        "Construction of buildings",
        "Civil engineering",
        "Specialized construction activities",
        "Other"
    ],
    "Education": [
        "Education",
        "Other"
    ],
    "Electricity, gas, steam and air conditioning supply": [
        "Electricity, gas, steam and air conditioning supply",
        "Other"
    ],
    "Financial and insurance activities": [
        "Financial service activities, except insurance and pension funding",
        "Insurance, reinsurance and pension funding, except compulsory social security",
        "Activities auxiliary to financial service and insurance activities",
        "Other"
    ],
    "Human health and social work activities": [
        "Human health activities",
        "Residential care activities",
        "Social work activities without accommodation",
        "Other"
    ],
    "Information and communication": [
        "Publishing activities",
        "Motion picture, video and television programme production, sound recording and music publishi
        "Programming and broadcasting activities",
        "Telecommunications",
        "Computer programming, consultancy and related activities",
        "Information service activities",
        "Other"
    ],
    "Manufacturing": [
        "Manufacture of food products",
        "Manufacture of beverages",
        "Manufacture of tobacco products",
        "Manufacture of textiles",
        "Manufacture of wearing apparel",
        "Manufacture of leather and related products",
        "Manufacture of wood and of products of wood and cork, except furniture; manufacture of articl
        "Manufacture of paper and paper products",
        "Printing and reproduction of recorded media",
        "Manufacture of coke and refined petroleum products",
        "Manufacture of chemicals and chemical products",
        "Manufacture of basic pharmaceutical products and pharmaceutical preparations",
        "Manufacture of rubber and plastics products",
```

```
        "Manufacture of other non-metallic mineral products",
        "Manufacture of basic metals",
        "Manufacture of fabricated metal products, except machinery and equipment",
        "Manufacture of computer, electronic and optical products",
        "Manufacture of electrical equipment",
        "Manufacture of machinery and equipment n.e.c.",
        "Manufacture of motor vehicles, trailers and semi-trailers",
        "Manufacture of other transport equipment",
        "Manufacture of furniture",
        "Other manufacturing",
        "Repair and installation of machinery and equipment",
        "Other"
    ],
    "Mining and quarrying": [
        "Mining of coal and lignite",
        "Extraction of crude petroleum and natural gas",
        "Mining of metal ores",
        "Other mining and quarrying",
        "Mining support service activities",
        "Other"
    ],
    "Other service activities": [
        "Activities of membership organizations",
        "Repair of computers and personal and household goods",
        "Other personal service activities",
        "Other"
    ],
    "Professional, scientific and technical activities": [
        "Legal and accounting activities",
        "Activities of head offices; management consultancy activities",
        "Architectural and engineering activities; technical testing and analysis",
        "Scientific research and development",
        "Advertising and market research",
        "Other professional, scientific and technical activities",
        "Veterinary activities",
        "Other"
    ],
    "Public administration and defence; compulsory social security": [
        "Public administration and defence; compulsory social security",
        "Other"
    ],
    "Real estate activities": [
        "Real estate activities",
        "Other"
    ],
    "Transportation and storage": [
        "Land transport and transport via pipelines",
        "Water transport",
        "Air transport",
        "Warehousing and support activities for transportation",
        "Postal and courier activities",
        "Other"
    ],
    "Water supply; sewerage, waste management and remediation activities": [
        "Water collection, treatment and supply",
        "Sewerage",
        "Waste collection, treatment and disposal activities; materials recovery",
        "Remediation activities and other waste management services",
```

```
        "Other"
    ],
    "Wholesale and retail trade; repair of motor vehicles and motorcycles": [
        "Wholesale and retail trade and repair of motor vehicles and motorcycles",
        "Wholesale trade, except of motor vehicles and motorcycles",
        "Retail trade, except of motor vehicles and motorcycles",
        "Other"
    ],
    "Other": [
        "Other"
    ]
}
```

CHAPTER 3

Omniwallet

Contents:

3.1 Introduction

Welcome to the world of Mastercoin.

This guide will walk you through the process of working with Omniwallet

3.1.1 Consensus

A quick note on Consensus.

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

This will soon be updated to the Mastercore project.

Until then periodically checking the consensus on the Masterchest Consensus Report can alert on potential differences among the implimentations.

3.1.2 Installed Items

Here is a breakdown of everything that is needed/installed to support Omniwallet

CHAPTER 4

Indices and tables

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