# SGB FTSO Contracts

Chuck Benedict

Aug 09, 2022

# USAGE

A small Python library to quickly instantiate Flare Time Series Oracle (FTSO) contracts on the Songbird network.

# CHAPTER ONE

# INSTALLATION

```
pip install sgb-ftso-contracts
```

CHAPTER

# TWO

# GET STARTED

How to get prices of crypto assets tracked by the Songbird network:

```
from sgb_ftso_contracts import Ftso
from web3 import Web3

# Songbird network RPC endpoint
# This is a free, rate-limited API node.
rpcurl = "https://songbird-api.flare.network/ext/bc/C/rpc"

# Init web3 with REST HTTP provider.
web3 = Web3(Web3.HTTPProvider(rpcurl))

# Create an FTSO contract instance with factory library.
btcFtso = Ftso("BTC").contract(web3)

# Fetch the latest price for Bitcoin from the FTSO.
btcDecimals = btcFtso.functions.ASSET_PRICE_USD_DECIMALS().call()
btcPriceData = btcFtso.functions.getCurrentPrice().call()

# Prices are recorded as integers. Convert to decimal format.
print(btcPriceData[0] / pow(10, btcDecimals))
```

## 2.1 API

### 2.1.1 FactoryBase

**class** sgb_ftso_contracts.**FactoryBase**(*apiurl='https://songbird-explorer.flare.network/api'*)

This is a class common to all contract factory operations.

self.**apiurl**

Location of Blockscout api endpoint for Songbird.

**Type**
string

**__init__**(*apiurl='https://songbird-explorer.flare.network/api'*)

The constructor for FactoryBase class.

**Parameters**
`apiurl` (*string*) – Location of Blockscout api endpoint for Songbird. Used for fetching ABI automatically.

`contract`(*web3: Web3*)

A method to instantiate a Web3 Contract set at self.address.

**Parameters**
`web3` (*Web3*) – An instance of the Web3 class, wired up to a valid provider.

**Returns**
An instantied contract at self.address.

**Return type**
Contract

`getABI`(*address*)

A method to fetch the application binary interface (ABI) for the contract at the specified address.

**Parameters**
`address` (*string*) – Contract address prefixed with 0x.

**Returns**
The ABI of the contract.

**Return type**
string

### 2.1.2 `Ftso`

`class sgb_ftso_contracts.Ftso`(*symbol*, *apiurl=None*)

A factory to instantiate the Ftso contract. Note that this link is the FTSO for BTC. FTSOs are all the same contract; one instance per asset.

`self.symbol`

The asset symbol of the FTSO.

**Type**
string

`__init__`(*symbol*, *apiurl=None*)

The constructor for Ftso class.

**Parameters**

- `symbol` (*string*) – The asset symbol of the FTSO.
- `apiurl` (*string*) – Location of Blockscout api endpoint for Songbird. Used for fetching ABI automatically.

`contract`(*web3: Web3*)

A method to instantiate a Web3 Contract set at self.address.

**Parameters**
`web3` (*Web3*) – An instance of the Web3 class, wired up to a valid provider.

**Returns**
An instantied contract at self.address for the FTSO identified by self.symbol.

**Return type**
Contract

### 2.1.3 `FtsoManager`

`class sgb_ftso_contracts.`**`FtsoManager`**`(`*address='0xbfA12e4E1411B62EdA8B035d71735667422A6A9e', apiurl=None*`)`

A factory to instantiate the FtsoManager contract. This contract manages FTSO voting operations and coordinates price and reward epoch execution.

`self.`**`address`**

The address of the contract starting with 0x.

**Type**
string

**`__init__`**`(`*address='0xbfA12e4E1411B62EdA8B035d71735667422A6A9e', apiurl=None*`)`

The constructor for FtsoManager class.

**Parameters**

- **`address`** (*`string`*) – Address of the contract starting with 0x.
- **`apiurl`** (*`string`*) – Location of Blockscout api endpoint for Songbird. Used for fetching ABI automatically.

### 2.1.4 `FtsoRegistry`

`class sgb_ftso_contracts.`**`FtsoRegistry`**`(`*address='0x6D222fb4544ba230d4b90BA1BfC0A01A94E6cB23', apiurl=None*`)`

A factory to instantiate the FtsoRegistry contract. This provides Centralized access to all FTSOs from one contract. This contract is handy to get FTSO current price info from, as one does not need to go after the FTSO directly.

`self.`**`address`**

The address of the contract starting with 0x.

**Type**
string

**`__init__`**`(`*address='0x6D222fb4544ba230d4b90BA1BfC0A01A94E6cB23', apiurl=None*`)`

The constructor for FtsoRegistry class.

**Parameters**

- **`address`** (*`string`*) – Address of the contract starting with 0x.
- **`apiurl`** (*`string`*) – Location of Blockscout api endpoint for Songbird. Used for fetching ABI automatically.

### 2.1.5 `FtsoRewardManager`

`class sgb_ftso_contracts.`**`FtsoRewardManager`**(*address='0xc5738334b972745067fFa666040fdeADc66Cb925'*, *apiurl=None*)

A factory to instantiate the FtsoRewardManager contract. Vote power delegators can use this contract to claim rewards and check reward status.

`self.`**`address`**

The address of the contract starting with 0x.

**Type**
string

**`__init__`**(*address='0xc5738334b972745067fFa666040fdeADc66Cb925'*, *apiurl=None*)

The constructor for FtsoRewardManager class.

**Parameters**

- **`address`** (*`string`*) – Address of the contract starting with 0x.
- **`apiurl`** (*`string`*) – Location of Blockscout api endpoint for Songbird. Used for fetching ABI automatically.

### 2.1.6 `VoterWhiteLister`

`class sgb_ftso_contracts.`**`VoterWhitelister`**(*address='0xa76906EfBA6dFAe155FfC4c0eb36cDF0A28ae24D'*, *apiurl=None*)

A factory to instantiate the VoterWhitelister contract. Price providers must be whitelisted in order to submit prices to the FTSOs. This contract is how they do it.

`self.`**`address`**

The address of the contract starting with 0x.

**Type**
string

**`__init__`**(*address='0xa76906EfBA6dFAe155FfC4c0eb36cDF0A28ae24D'*, *apiurl=None*)

The constructor for VoterWhitelister class.

**Parameters**

- **`address`** (*`string`*) – Address of the contract starting with 0x.
- **`apiurl`** (*`string`*) – Location of Blockscout api endpoint for Songbird. Used for fetching ABI automatically.

### 2.1.7 `WNAT`

`class sgb_ftso_contracts.`**`WNAT`**(*address='0x02f0826ef6aD107Cfc861152B32B52fD11BaB9ED'*, *apiurl=None*)

A factory to instantiate the WNAT contract. Holders of the native chain token (for the Songbird network, SGB) use this contract to wrap their Songbird in order to delegate vote power to FTSOs. WNAT = wrapped native self.**address**

The address of the contract starting with 0x.

**Type**

string

**__init__**(*address='0x02f0826ef6aD107Cfc861152B32B52fD11BaB9ED'*, *apiurl=None*)

The constructor for WNAT class.

**Parameters**

- **address** (*string*) – Address of the contract starting with 0x.
- **apiurl** (*string*) – Location of Blockscout api endpoint for Songbird. Used for fetching ABI automatically.

## 2.2 Claim

### 2.2.1 Claim Unclaimed Rewards

To use this example, create a local file containing an encrypted private key for the account to claim rewards for. Never store private keys in the clear. This example prompts you to enter the password from the command line and decrypts the private key for one-time use.

```python
from sgb_ftso_contracts import *
from web3 import Web3
from eth_account import Account
import getpass

rpcurl = "https://songbird-api.flare.network/ext/bc/C/rpc"

# Init web3
web3 = Web3(Web3.HTTPProvider(rpcurl))

# Init the FtsoRewardManager contract
ftsoRewardManagerFactory = FtsoRewardManager()
ftsoRewardManagerContract = ftsoRewardManagerFactory.contract(web3)

# Setup default web3 transaction parameters. Note chain id 19 is the Songbird chain.
tx_parms = {
  'chainId': 19,
  'gas': 500000,
  'gasPrice': web3.toWei('50', 'gwei'),
  'nonce': 0
}

# Get the password to unencrypt key
password = getpass.getpass()

with open("an encrypted private key file reference goes here", 'r') as f:
  # Set up account to work with by decrypting the private key.
  encrypted = json.loads(f.read())
  privatekey = Account.decrypt(encrypted, password)
```

(continues on next page)

(continued from previous page)

```
  account = Account.from_key(privatekey)
  web3.eth.default_account = account.address

  # See if there are unclaimed rewards by reward epoch for account.
  unclaimedEpochs = ftsoRewardManagerContract.functions.
↪getEpochsWithUnclaimedRewards(account.address).call()

  if len(unclaimedEpochs) > 0:

    # Build the undelegate transaction
    tx_parms["nonce"] = web3.eth.getTransactionCount(account.address)
    # Build transaction to claim rewards for all unclaimed reward epochs
    tx = ftsoRewardManagerContract.functions.claimReward(account.address,␣
↪unclaimedEpochs).buildTransaction(tx_parms)
    # Sign transaction
    signed_tx = account.sign_transaction(tx)
    # Execute the transaction
    tx_hash = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
    tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
    print(f'Claimed with tx_hash {tx_receipt["transactionHash"].hex()}')
```

## 2.3 Delegation

### 2.3.1 Undelegate All Votepower to Ftsos

To use this example, create a local file containing an encrypted private key for the account to undelegate from. Never store private keys in the clear. This example prompts you to enter the password from the command line and decrypts the private key for one-time use.

```
from sgb_ftso_contracts import *
from web3 import Web3
from eth_account import Account
import getpass

rpcurl = "https://songbird-api.flare.network/ext/bc/C/rpc"

# Init web3
web3 = Web3(Web3.HTTPProvider(rpcurl))

# Init the WNAT contract
wNatFactory = WNAT()
wNat = wNatFactory.contract(web3)

# Setup default web3 transaction parameters. Note chain id 19 is the Songbird chain.
tx_parms = {
  'chainId': 19,
  'gas': 500000,
  'gasPrice': web3.toWei('50', 'gwei'),
  'nonce': 0
}
```

(continues on next page)

(continued from previous page)

```python
# Get the password to unencrypt key
password = getpass.getpass()

with open("an encrypted private key file reference goes here", 'r') as f:
  # Set up account to work with by decrypting the private key.
  encrypted = json.loads(f.read())
  privatekey = Account.decrypt(encrypted, password)
  account = Account.from_key(privatekey)
  web3.eth.default_account = account.address

  # Build the undelegate transaction
  tx_parms["nonce"] = web3.eth.getTransactionCount(account.address)
  tx = wNat.functions.undelegateAll().buildTransaction(tx_parms)
  signed_tx = account.sign_transaction(tx)

  # Execute the transaction
  tx_hash = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
  tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
  print(f'Undelegated with tx_hash {tx_receipt["transactionHash"].hex()}')
```

### 2.3.2 Delegate Votepower By Percentage to Ftsos

To use this example, create a local file containing an encrypted private key for the account to undelegate from. Never store private keys in the clear. This example prompts you to enter the password from the command line and decrypts the private key for one-time use.

```python
from sgb_ftso_contracts import *
from web3 import Web3
from eth_account import Account
import getpass

rpcurl = "https://songbird-api.flare.network/ext/bc/C/rpc"

# Init web3
web3 = Web3(Web3.HTTPProvider(rpcurl))

# Init the WNAT contract
wNatFactory = WNAT()
wNat = wNatFactory.contract(web3)

# Setup default web3 transaction parameters. Note chain id 19 is the Songbird chain.
tx_parms = {
  'chainId': 19,
  'gas': 500000,
  'gasPrice': web3.toWei('50', 'gwei'),
  'nonce': 0
}

# Get the password to unencrypt key
password = getpass.getpass()
```

(continues on next page)

(continued from previous page)

```
with open("an encrypted private key file reference goes here", 'r') as f:
  # Set up account to work with by decrypting the private key.
  encrypted = json.loads(f.read())
  privatekey = Account.decrypt(encrypted, password)
  account = Account.from_key(privatekey)
  web3.eth.default_account = account.address

  # Build the undelegate transaction
  tx_parms["nonce"] = web3.eth.getTransactionCount(account.address)
  # Percentage in basis points = % * 100
  tx = wNat.functions.delegate(web3.toChecksumAddress("delegator address goes here"),␣
↪10000).buildTransaction(tx_parms)
  signed_tx = account.sign_transaction(tx)

  # Execute the transaction
  tx_hash = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
  tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
  print(f'Undelegated with tx_hash {tx_receipt["transactionHash"].hex()}')
```

## 2.4 Encryption

### 2.4.1 Encrypt Private Key

To use the included examples that execute transactions on the Songbird blockchain, you will need access to your private key (or a wallet and an API). Hardware or software wallets are out of scope for these examples. Private keys should never be stored in the clear, so this utility provides a means to encrypt your key with a password.

```
from eth_account import Account
from getopt import getopt, GetoptError
from pprint import pprint
import json
import sys
import getpass

def main(argv):
  '''Encrypt a wallet private key using a passphrase and store in a
  local directory.

  The encrypted key file is defined with the -o switch on the command line.
  '''
  outputfile = ''

  # Get command line args
  try:
    opts, args = getopt(argv, "ho:", ["outputfile="])
  except GetoptError:
    printinvoke()
    exit(2)
```

(continues on next page)

(continued from previous page)

```
  # Private key and password could be passed on command line for automation
  # purposes, but is not secure since command history can be stored.
  for opt, arg in opts:
    if opt == "-h":
      printinvoke()
      exit()
    elif opt in ("-o", "--outputfile"):
      outputfile = arg

  # Get private key and password...do not echo to terminal.
  key = getpass.getpass("Private key: ")
  password = getpass.getpass()

  # Encrypt...
  encrypted = Account.encrypt(key, password)
  pprint(encrypted)

  # Write encrypted key to output file defined on command line.
  with open(outputfile, 'w') as f:
    f.write(json.dumps(encrypted))

def printinvoke():
  print("usage:")
  print("  encryptkey.py -o <outputfile>")

if __name__ == "__main__":
  main(sys.argv[1:])
```

## 2.5 Events

The Songbird smart contract collection fires many events that can give you insight on Ftso operations as they happened. Any event can be fetched, so long as you know what block range to look for. It is unwise to start a block 0, scanning all blocks in the chain. Most API providers will not allow this operation. So one should build and keep a mapping of block to something like price epoch or reward epoch off-chain, filtering over small block ranges shortly after they happen. This can be done with RPC or streaming over web sockets. Web sockets are more performant but less reliable (IMHO). The examples below use RPC.

### 2.5.1 PriceHashesSubmitted

Retrieve logs from blocks containing the PriceHashesSubmitted event from the PriceSubmitter contract.

```
from sgb_ftso_contracts import PriceSubmitter
from web3 import Web3

# Define the blockchain endpoint to use
rpcurl = "https://songbird-api.flare.network/ext/bc/C/rpc"

# Init web3
web3 = Web3(Web3.HTTPProvider(rpcurl))
```

(continues on next page)

(continued from previous page)

```python

# Get the PriceSubmitter contract
priceSubmitterFactory = PriceSubmitter()
priceSubmitter = priceSubmitterFactory.contract(web3)

# This search happens over a block range, so it is necessary to define that range.
endblock = 20481300
submitfilter = priceSubmitter.events.PriceHashesSubmitted.
↪createFilter(fromBlock=endblock - 30, toBlock=endblock)
events = submitfilter.get_all_entries()
for event in events:
  print(event["args"]["submitter"])
  print(event["args"]["epochId"])
  print(event["args"]["ftsos"])
  print(event["args"]["timestamp"])
```

### 2.5.2 PricesRevealed

Retrieve logs from blocks containing the PricesRevealed event from the PriceSubmitter contract. This example will print the revealed prices for ADA for a given epoch.

```python
from sgb_ftso_contracts import Ftso, PriceSubmitter
from web3 import Web3

# Define the blockchain endpoint to use
rpcurl = "https://songbird-api.flare.network/ext/bc/C/rpc"

# Init web3
web3 = Web3(Web3.HTTPProvider(rpcurl))

# Init the ftsoManager factory
ftsoFactory = Ftso("ADA")
ftsoADA = ftsoFactory.contract(web3)

# Get the PriceSubmitter contract
priceSubmitterFactory = PriceSubmitter()
priceSubmitter = priceSubmitterFactory.contract(web3)

# This search happens over a block range, so it is necessary to define that range.
endblock = 20481300

# Create an event filter
revealfilter = priceSubmitter.events.PricesRevealed.createFilter(fromBlock=endblock - 15,
↪ toBlock=endblock + 15)
events = revealfilter.get_all_entries()
# Iterate over the events found
for event in events:
  pos = 0
  print(event["args"]["epochId"])
  print(event["args"]["voter"])
  for ftso in event["args"]["ftsos"]:
```

(continues on next page)

(continued from previous page)

```
    if ftso.lower() == ftsoADA.address.lower():
      print(event["args"]["prices"][pos])
    else:
      pos += 1
```

### 2.5.3 PriceEpochInitializedOnFtso

Retrieve logs from blocks containing the PriceEpochInitializedOnFtso event from the ADA Ftso contract. Any Ftso would work. Iterating over ranges of blocks, one can map blocks to price epochs, which then enable fetching other events within the same price epoch. Knowing what blocks to go after is the key.

```
from sgb_ftso_contracts import *
from web3 import Web3

rpcurl = "https://songbird-api.flare.network/ext/bc/C/rpc"

# Init web3
web3 = Web3(Web3.HTTPProvider(rpcurl))

# Init an ftso factory
ftsoFactory = Ftso("ADA")
ftsoADA = ftsoFactory.contract(web3)

# Some block that has already occured
endblock = 20481240

# Discern when price epochs occur in the context of chain blocks.
# Blocks are not produced at a constant rate, so this mapping cannot be determined by␣
↪formula.
epochInitFilter = ftsoADA.events.PriceEpochInitializedOnFtso.
↪createFilter(fromBlock=endblock - 30, toBlock=endblock)
events = epochInitFilter.get_all_entries()
for event in events:
  print(f"Over block range: {endblock - 30} to {endblock}")
  print(f"At block: {event['blockNumber']}")
  print(f"The price epoch was: {event['args']['epochId']}")
  print(f"And the price epoch ends at Unix epoch timestamp: {event['args']['endTime']}")
```

CHAPTER

# THREE

## INDICES AND TABLES

- genindex
- modindex
- search

# INDEX