shadowlands Documentation

Cath Thomas

Jun 16, 2019

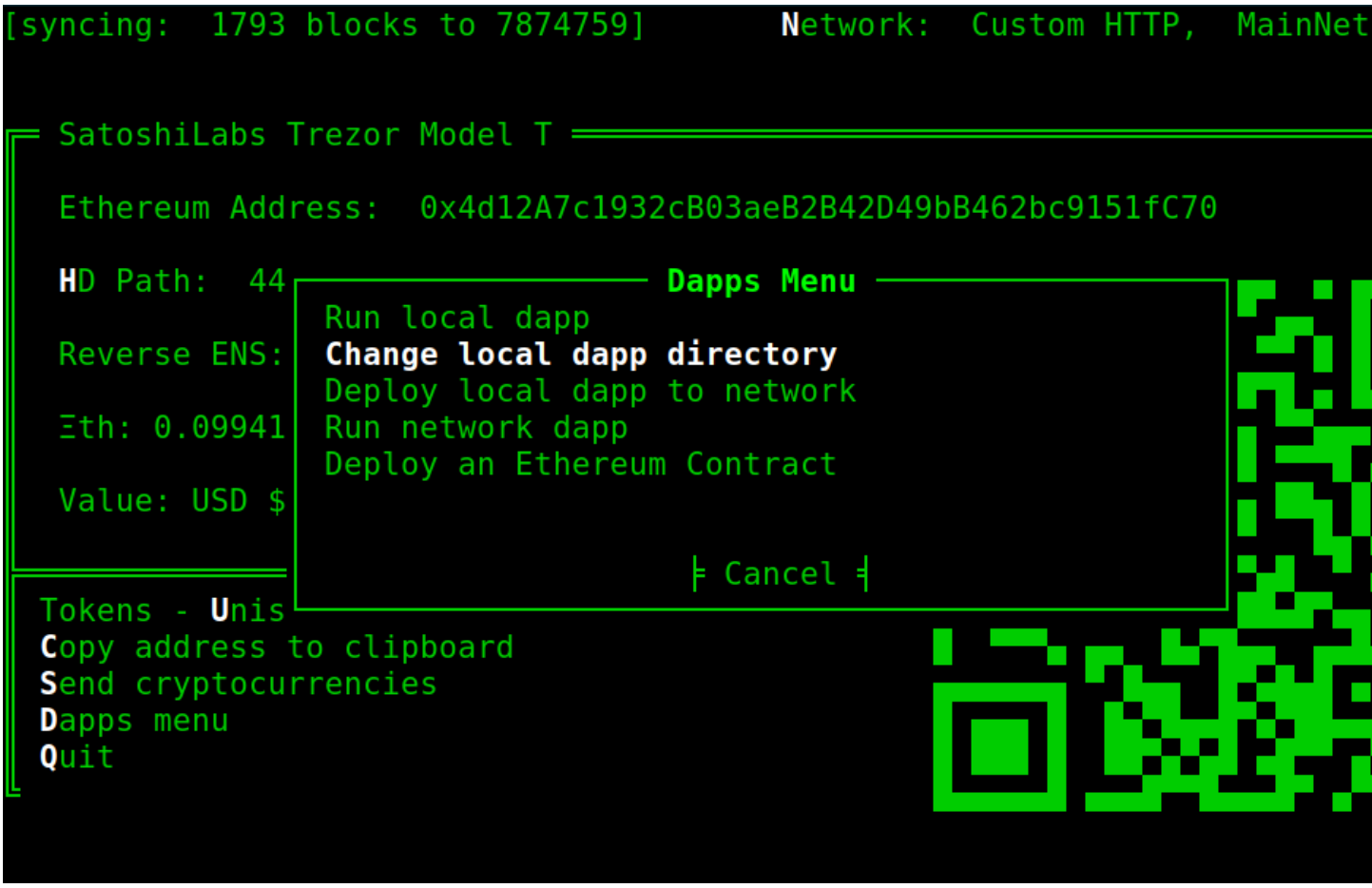
Contents

Shadowlands is a python-based platform for writing Ethereum Dapps without a web browser.

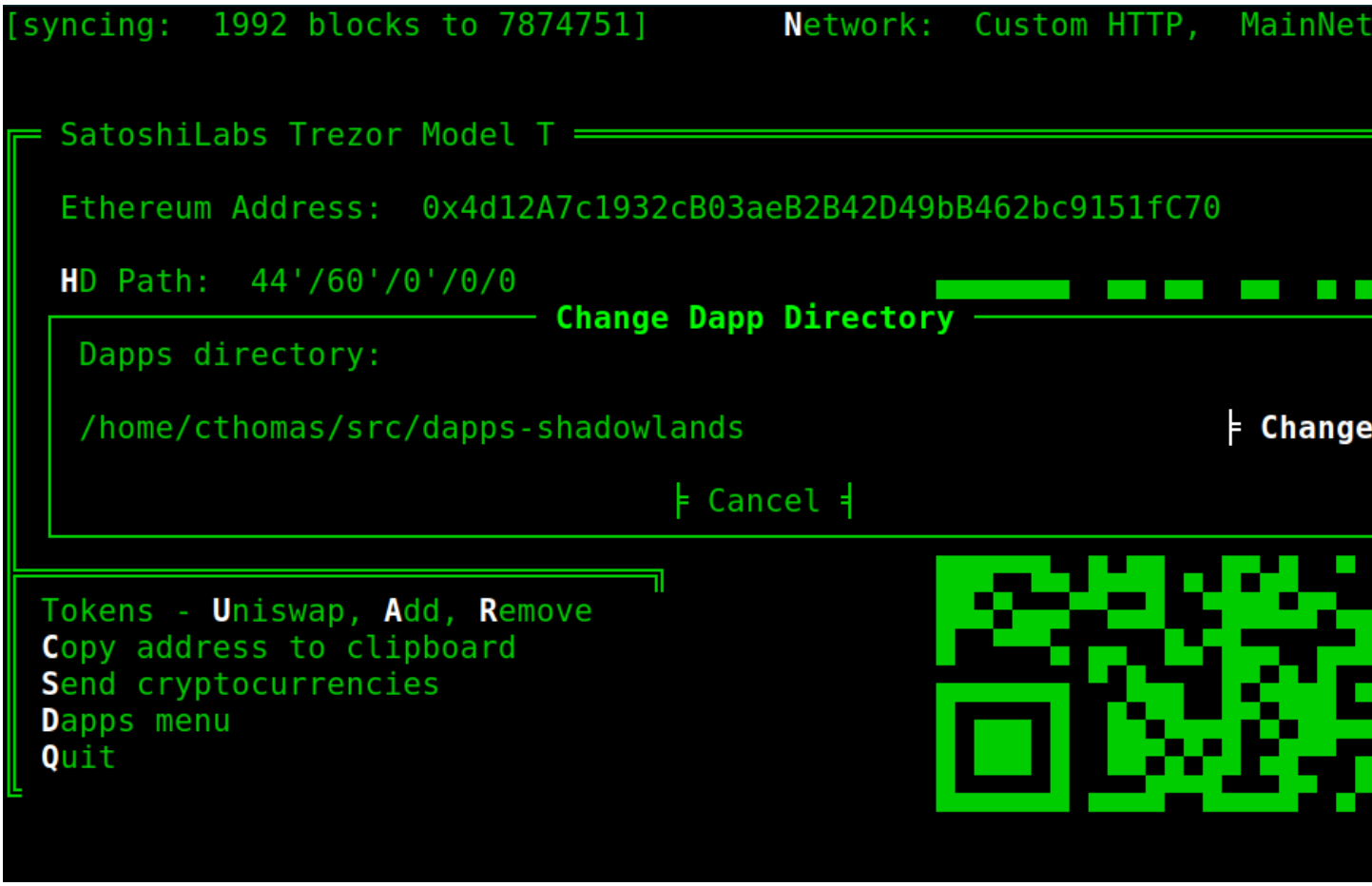
CHAPTER 1

Tutorial

To start your own Dapp, open the dapps menu by pressing ‘d’ or ‘D’ within the shadowlands application.



Next, set the local dapp directory to your desired location.



Inside your chosen local dapp directory, create a new subdirectory with the name of your dapp. Inside, create a file called `__init__.py`.

```
(.shadowlands) cthomas@soykaf:~/src/dapps-shadowlands$ mkdir trogdor
(.shadowlands) cthomas@soykaf:~/src/dapps-shadowlands$ cd trogdor
(.shadowlands) cthomas@soykaf:~/src/dapps-shadowlands/trogdor$ touch __i
(.shadowlands) cthomas@soykaf:~/src/dapps-shadowlands/trogdor$
```

1.1 Your first SLDapp

For our example, let's become Trogdor the Burninator, the wingaling dragon. We will burninate peasants in the kingdom of peasantry. In our case, peasants will be BRNT (Burninator tokens) visible at token.burninator.eth)

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame
from shadowlands.sl_contract import SLContract

class Dapp(SLDapp):
    def initialize(self):
        # Define any variables that will be useful to you, such as contracts.
        # Any other setup steps go here
```

(continues on next page)

(continued from previous page)

```
        # add a frame to begin the user interface
        self.add_sl_frame(MyMenuFrame(self, height=5, width=40, title=
↪"Trogdooooor!"))

class MyMenuFrame(SLFrame):
    def initialize(self):
        self.add_divider()
        self.add_button(self.close, "Close")
```

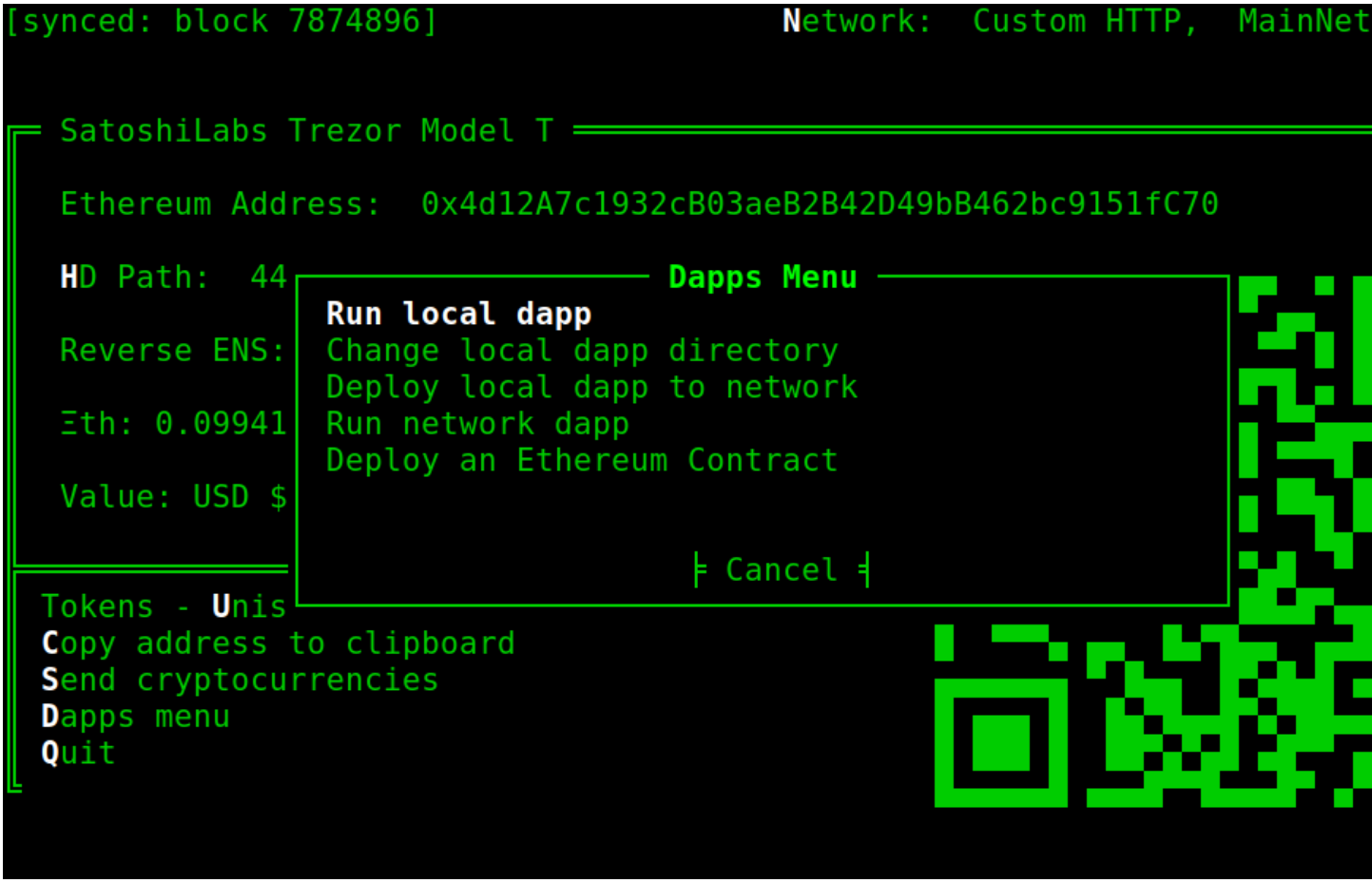
Import *SLDapp* at the top of your `__init__.py` file in your dapp's subdirectory. We'll also import *SLFrame* and *SLContract*.

Create a class named `Dapp` that subclasses *SLDapp*. The class must be named `Dapp` in order for the shadowlands plugin system to detect your dapp. Override the *SLDapp.initialize()* method, and do any necessary preperation within. Then, add an *SLFrame* subclass (which you need to provide) with *SLDapp.add_sl_frame()*. This step begins the user interface.

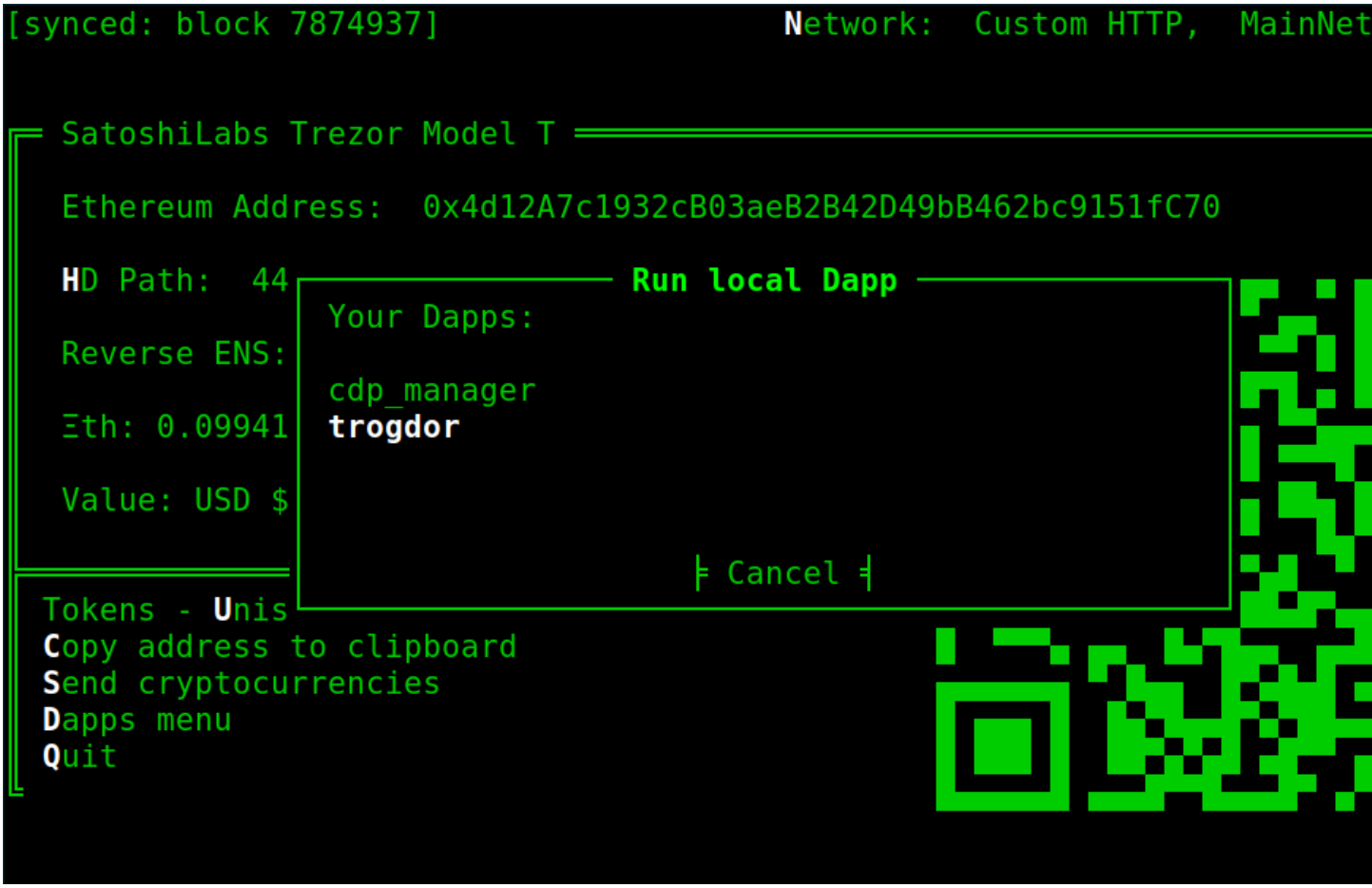
The line `self.add_sl_frame(MyMenuFrame(self, height=5, width=40, title="Trogdooooor!"))`, referenced from `initialize()`, will load an *SLFrame* instance with the listed parameters when the dapp loads.

Like *SLDapp* instances, *SLFrame* instances execute `initialize()` when they are created, and you must implement this abstract method. Our *SLFrame* will add a one-line divider with `self.add_divider()` and then add a close button with `self.add_button(self.close, "Close")`. The first parameter to `self.add_button` is a function to be executed upon the button press action, in this case `self.close()`.

Now, let's run our first dapp. Open the dapps menu and choose to run a local dapp:



Now, choose your dapp from the list:



And this is the output:

1.2 Debugging

Now, let's make a few changes.

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame
from shadowlands.sl_contract.erc20 import Erc20
from shadowlands.tui.debug import debug, end_debug
import pdb

class Dapp(SLDapp):
    def initialize(self):
        # Define any variables that will be useful to you, such as contracts.
        # Any other setup steps go here
        debug(); pdb.set_trace()

        PEASANT_ADDRESS = '0x8B654789353b0B622667F105eAEF9E97d3C33F44'
        peasant_contract = Erc20(self.node, address=PEASANT_ADDRESS)
        self.add_sl_frame(MyMenuFrame(self, height=5, width=40, title="Trogdooooor!"))

        # add a frame to begin the user interface

class MyMenuFrame(SLFrame):
```

(continues on next page)

(continued from previous page)

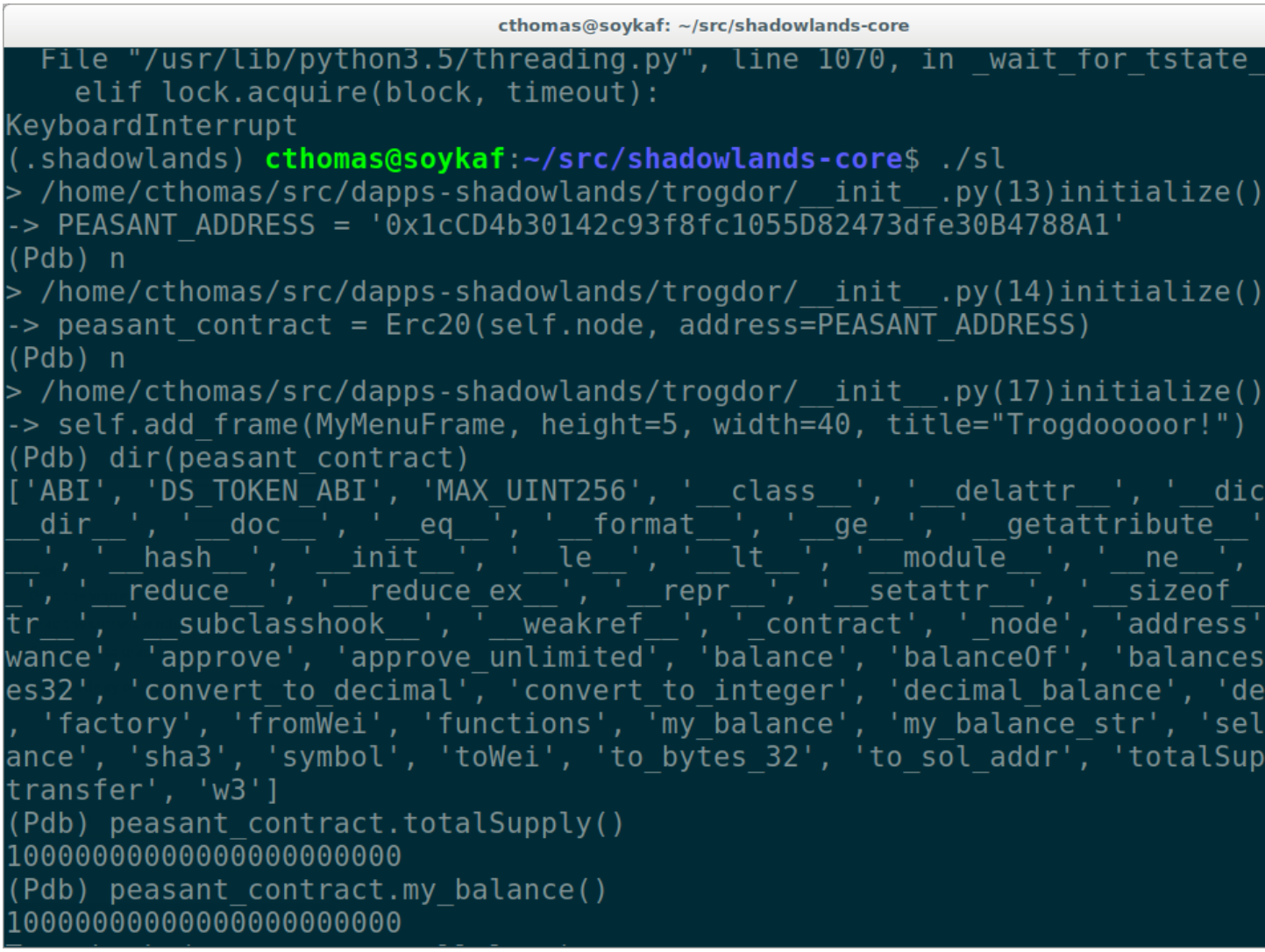
```
    def initialize(self):
        self.add_divider()
        self.add_button(self.close, "Close")
```

Here you can see we've set up some debugging tools with a few import statements. The functions debug() and end_debug() will give us a way to escape from the curses library that's controlling the screen and let pdb work.

You can also see I defined `PEASANT_ADDRESS` which is the ethereum mainnet address of a simple ERC20 contract. We load the contract with the `Erc20(self.node, address=PEASANT_ADDRESS)` constuctor. `self.node` is a reference to the `Node` object that the Dapp object has access to.

The important line `debug(); pdb.set_trace()` is something you should become familiar with when writing a shadowlands app. Running pdb without escaping from the user interface will be a maddening experience, so don't forget to run `debug()` before you get pdb running.

Now, when you run your dapp, you'll see:



Here you can see some of the methods that the *Erc20* class provides. You can also access the underlying web3.py contract object by accessing `peasant_contract._contract`.

To escape from the debug session and get back to your app, type `end_debug();; continue`. This incantation

will restore control of the screen to the curses library and end the session.

1.3 Requirements File

You can include a `requirements.txt` file in your dapp directory to import modules that you might need. They will be installed into Shadowlands' python virtual environment at `~/.shadowlands` when the dapp runs on the host system.

There's no library dependancy in this tutorial, I just wanted to mention it.

1.4 Handling user input

Let's get some user input and do something, er... useful?

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame
from shadowlands.sl_contract.erc20 import Erc20
from decimal import Decimal
from shadowlands.tui.debug import debug, end_debug
import pdb

class Dapp(SLDapp):
    def initialize(self):
        PEASANT_ADDRESS = '0x8B654789353b0B622667F105eAEF9E97d3C33F44'
        self.peasant_contract = Erc20(self.node, address=PEASANT_ADDRESS)
        self.peasants = self.peasant_contract.my_balance() / Decimal(10**18)
        self.add_sl_frame(MyMenuFrame(self, height=10, width=70, title="Trogdooooor!
↪"))

class MyMenuFrame(SLFrame):
    def initialize(self):
        self.add_label("Trogdor the wingaling dragon intends to burninate peasants.")
        self.add_label("Trogdor has {} peasants in need of burnination.".format(self.
↪peasants_str))
        self.text_value = self.add_textbox("How many?")
        self.add_divider()
        self.add_button_row([
            ("Burninate!", self.burninate, 0),
            ("Close", self.close, 1)
        ])

    @property
    def peasants_str(self):
        return "{:f}".format(self.dapp.peasants)[:8]

    def burninate(self):
        try:
            peasants_to_burninate = Decimal(self.text_value())
        except:
            self.dapp.add_message_dialog("That number of peasants doesn't make sense.
↪")
            return

        if peasants_to_burninate > self.dapp.peasants:
```

(continues on next page)

(continued from previous page)

```
            self.dapp.add_message_dialog("You don't even *have* that many peasants!")
            return
        elif peasants_to_burninate < 0.5:
            self.dapp.add_message_dialog("This will not satisfy Trogdor.")
            return
```

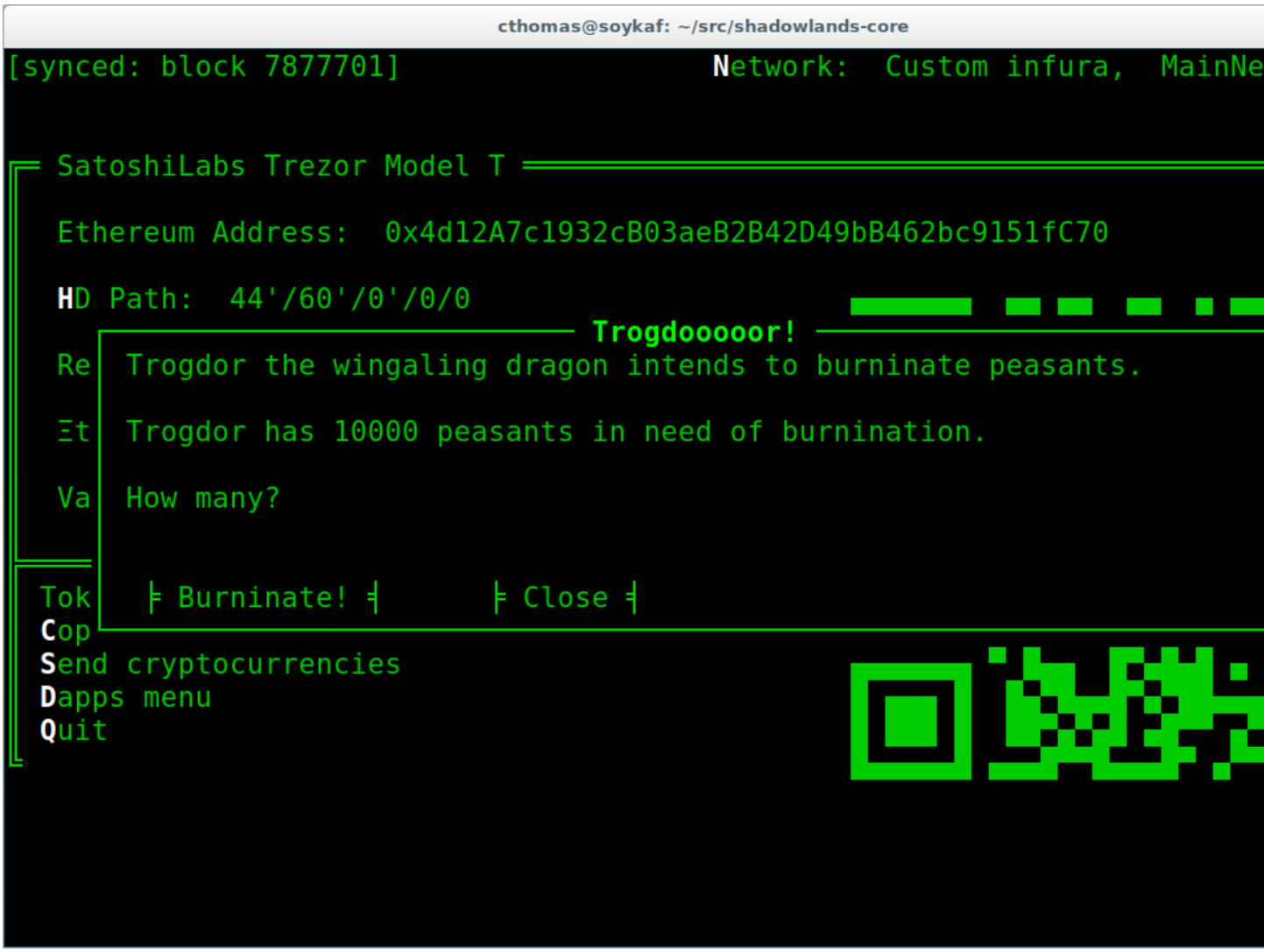
We've add some height and width to our *SLFrame* on line 13, added labels and a textbox on lines 17 - 19, and traded in our simple button for `add_button_row()` on line 21. All of the widgets available to display are documented on the *SLFrame* page.

On line 12, we divide the number of peasantcoins by (10 ** 18) to account for the 18 decimal places of precision of this coin.

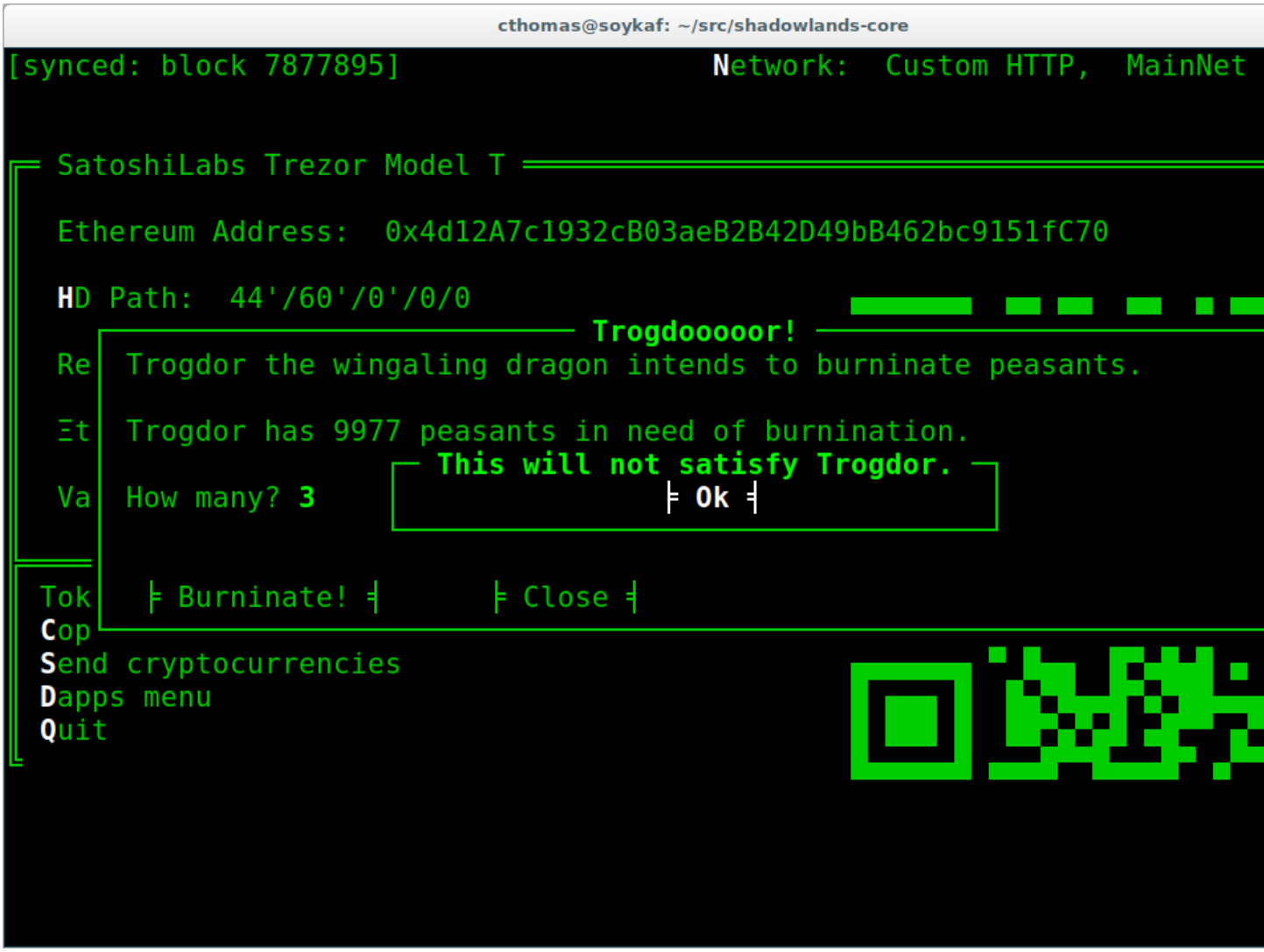
We're doing some simple input sanitization here, as well as some restrictions as to how many peasants can be burninated in one go.

Note that `add_message_dialog()` is a method belonging to Dapp, which is always accessible from an *SLFrame* instance via `self.dapp`.

So, let's see how we did.



Below we see the result of failing the input validation.



1.5 Transactions

Let's get on to burninating some peasants.

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame
from shadowlands.sl_contract.erc20 import Erc20
from decimal import Decimal
from shadowlands.tui.debug import debug, end_debug
import pdb

class Dapp(SLDapp):
    def initialize(self):
        PEASANT_ADDRESS = '0x8B654789353b0B622667F105eAEF9E97d3C33F44'
        self.peasant_contract = Erc20(self.node, address=PEASANT_ADDRESS)
        self.peasants = Decimal(self.peasant_contract.my_balance() / (10 ** 18))
```

(continues on next page)

(continued from previous page)

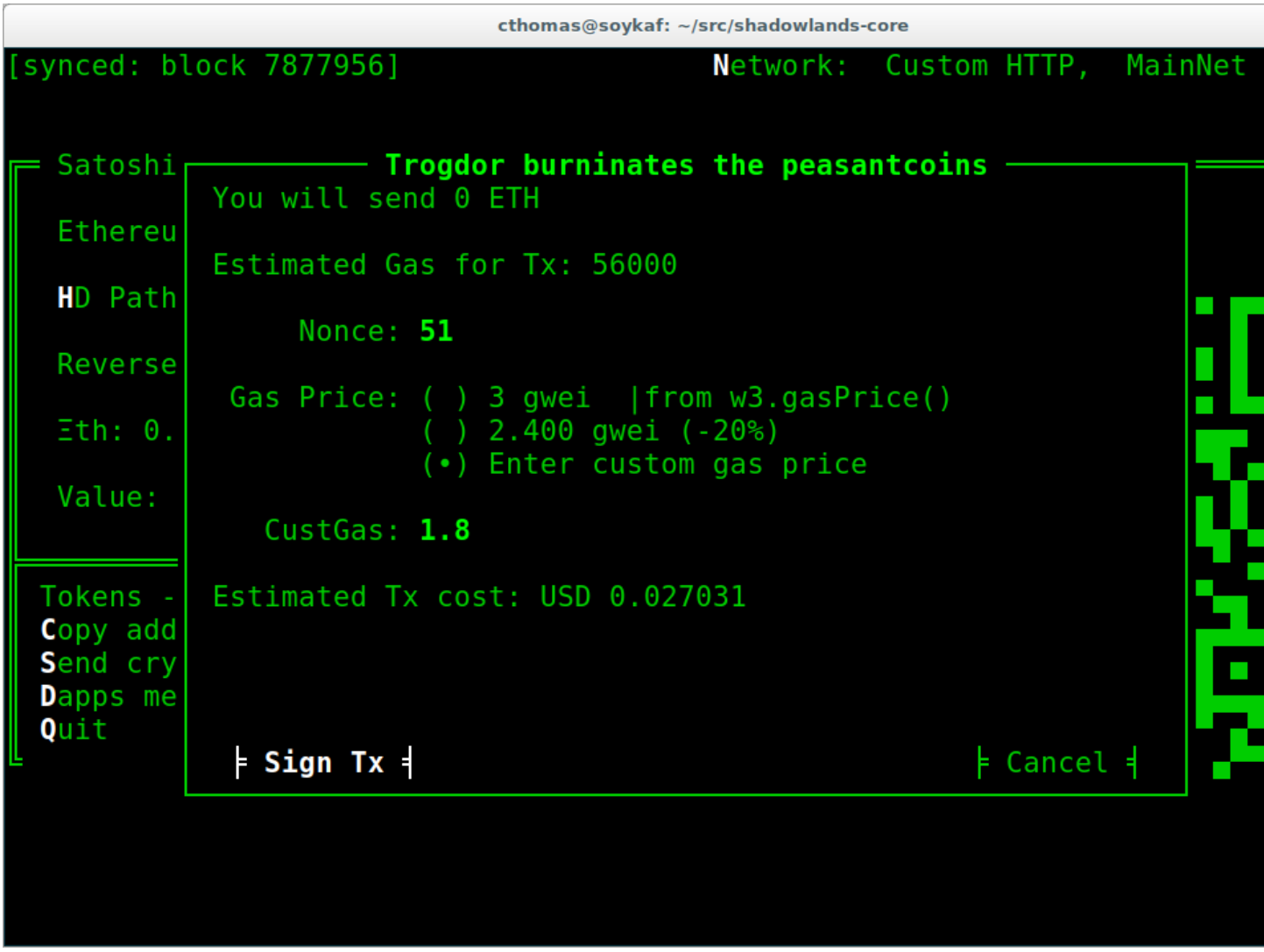
```
        self.add_sl_frame(MyMenuFrame(self, height=10, width=70, title="Trogdooooor!
↪"))

class MyMenuFrame(SLFrame):
    def initialize(self):
        self.add_label("Trogdor the wingaling dragon intends to burninate peasants.")
        self.add_label("Trogdor has {} peasants in need of burnination.".format(self.
↪peasants_str))
        self.text_value = self.add_textbox("How many?")
        self.add_divider()
        self.add_button_row([
            ("Burninate!", self.burninate, 0),
            ("Close", self.close, 1)
        ])

    @property
    def peasants_str(self):
        return "{:f}".format(self.dapp.peasants)[:8]

    def peasants_validated(self):
        try:
            self.peasants_to_burninate = Decimal(self.text_value())
        except:
            self.dapp.add_message_dialog("That number of peasants doesn't make sense.
↪")
            return False

        if self.peasants_to_burninate > self.dapp.peasants:
            self.dapp.add_message_dialog("You don't even *have* that many peasants!")
            return False
        elif self.peasants_to_burninate < 0.5:
            self.dapp.add_message_dialog("This will not satisfy Trogdor.")
            return False

        return True


    def burninate(self):
        if not self.peasants_validated():
            return

        peasantcoins_to_burninate = self.peasants_to_burninate * Decimal(10 ** 18)

        burn_fn = self.dapp.peasant_contract.transfer(
            '0x000000000000000000000000000000DeaDBeef',
            int(peasantcoins_to_burninate)
        )

        self.dapp.add_transaction_dialog(
            burn_fn,
            title="Trogdor burninates the peasantcoins",
            gas_limit=56000
        )

        self.close()
```

I've refactored our input validation to the method `peasants_validated` on line 30.

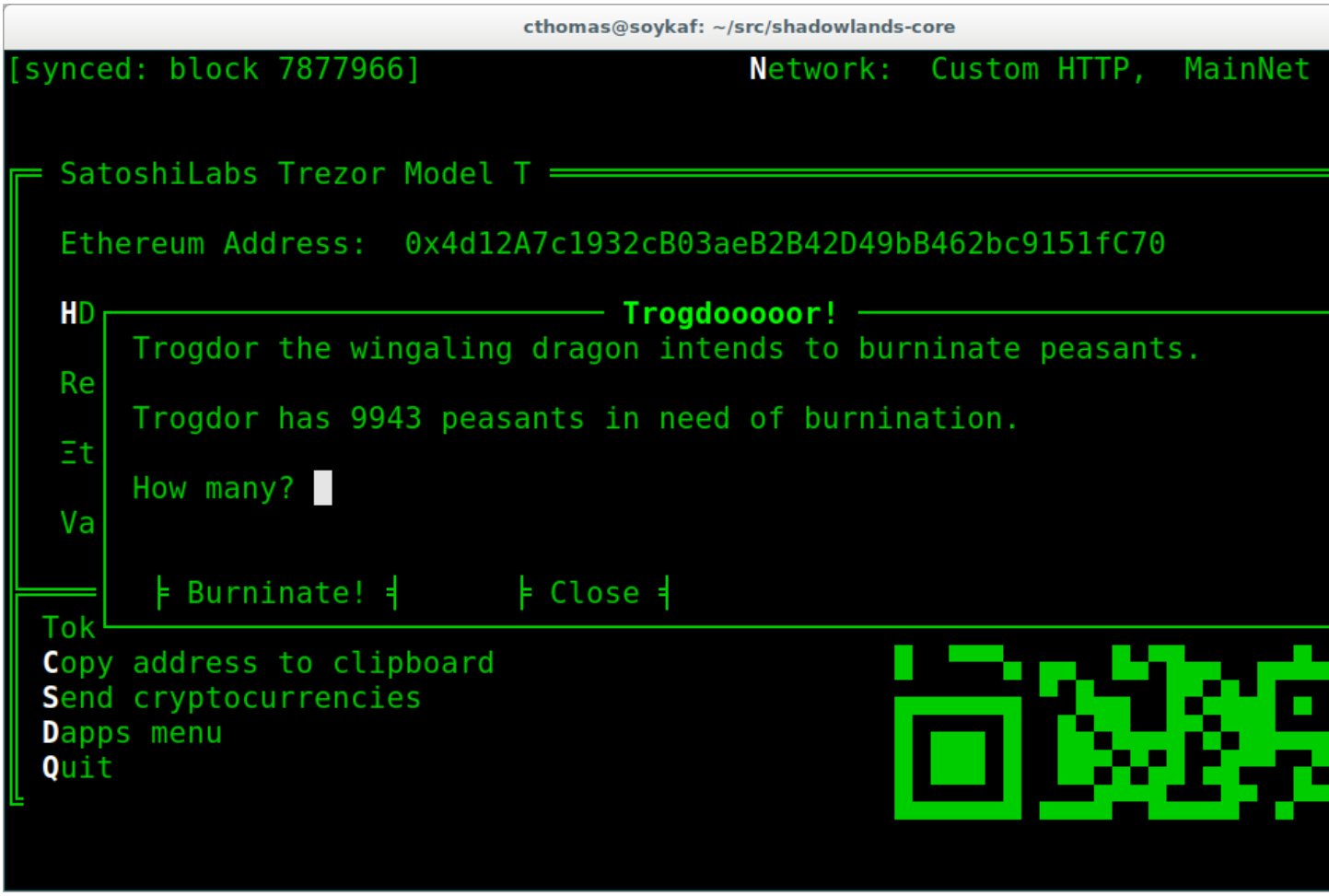
So, the time has come for the peasants to meet their final, fiery farewell at the nostrils of the mighty Trogdor.

Note on line 56 that `peasant_contract.transfer()` returns a method, which we will feed into `add_transaction_dialog()` on line 61.

Let's see how this looks in practice.



And so, we see there are a few less peasants in the kingdom of peasantry.



1.6 Subclassing `Erc20` and `SLContract`

Trogdor is distraught to discover that the peasants have not actually been burninated, but only banished to the cave of 0xdeadbeef. He demands true burnination.

Luckily, the PSNT contract supports burn(), although this is not a standard Erc20 function. Let's subclass *Erc20* and use some of the features of SLContract to make our lives easier.

```
from shadowlands.sl_contract.erc20 import Erc20

class PeasantCoin(Erc20):
    MAINNET='0x8B654789353b0B622667F105eAEF9E97d3C33F44'
    ABI='''
        [
                {
                        "constant": true,
                        "inputs": [],

        ..(ABI truncated for brevity)...


        ]
```

(continues on next page)

(continued from previous page)

```
        '''
```

First we create a file called `peasant_coin.py` in our `trogdor` directory to house our subclass.

`PeasantCoin` subclasses `Erc20`. The default ABI for the `Erc20` subclass doesn't understand burn(), so we need to supply our own ABI.

Subclassing `SLContract` (the superclass of `Erc20`) works the same way - you can define MAINNET, KOVAN, and other network names that are defined by `Node.NETWORK_DICT`, and set these to the deployment address of the contract.

We also can paste the ABI here. See the documentation for `SLContract` and `Erc20` to fully understand everything they provide.

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame
from trogdor.peasant_coin import PeasantCoin
from decimal import Decimal
from shadowlands.tui.debug import debug, end_debug
import pdb

class Dapp(SLDapp):
    def initialize(self):
        self.peasant_contract = PeasantCoin(self.node)
        self.peasants = Decimal(self.peasant_contract.my_balance() / (10 ** 18))
        self.total_peasants =  Decimal(self.peasant_contract.totalSupply() / (10 **␣
↪18))
        self.add_sl_frame(MyMenuFrame(self, height=12, width=70, title="Trogdooooor!
↪"))

class MyMenuFrame(SLFrame):
    def initialize(self):
        self.add_label("Trogdor the wingaling dragon intends to burninate peasants.")
        self.add_label("There are {} peasants (PSNT) in the world.".format(
                self.peasant_decorator(self.dapp.total_peasants)
        ))
        self.add_label("Trogdor has {} peasants in need of burnination.".format(
                self.peasant_decorator(self.dapp.peasants)
        ))
        self.text_value = self.add_textbox("How many?")
        self.add_divider()
        self.add_button_row([
            ("Burninate!", self.burninate, 0),
            ("Close", self.close, 1)
        ])

    def peasant_decorator(self, peasants):
        return "{:f}".format(peasants)[:12]
```

We import `PeasantCoin` on line 3 and instantiate it on line 10. We also grab the `totalSupply()` on line 12. Some refactoring into a decorator on line 31 makes things a little nicer.

```
def burninate(self):
    if not self.peasants_validated():
        return

    peasantcoins_to_burninate = self.peasants_to_burninate * Decimal(10 ** 18)
```

(continues on next page)

(continued from previous page)

```
    burn_fn = self.dapp.peasant_contract.functions.burn(
        int(peasantcoins_to_burninate)
    )

    self.dapp.add_transaction_dialog(
        burn_fn,
        title="Trogdor burninates the peasantcoins",
        gas_limit=56000
    )

    self.close()
```

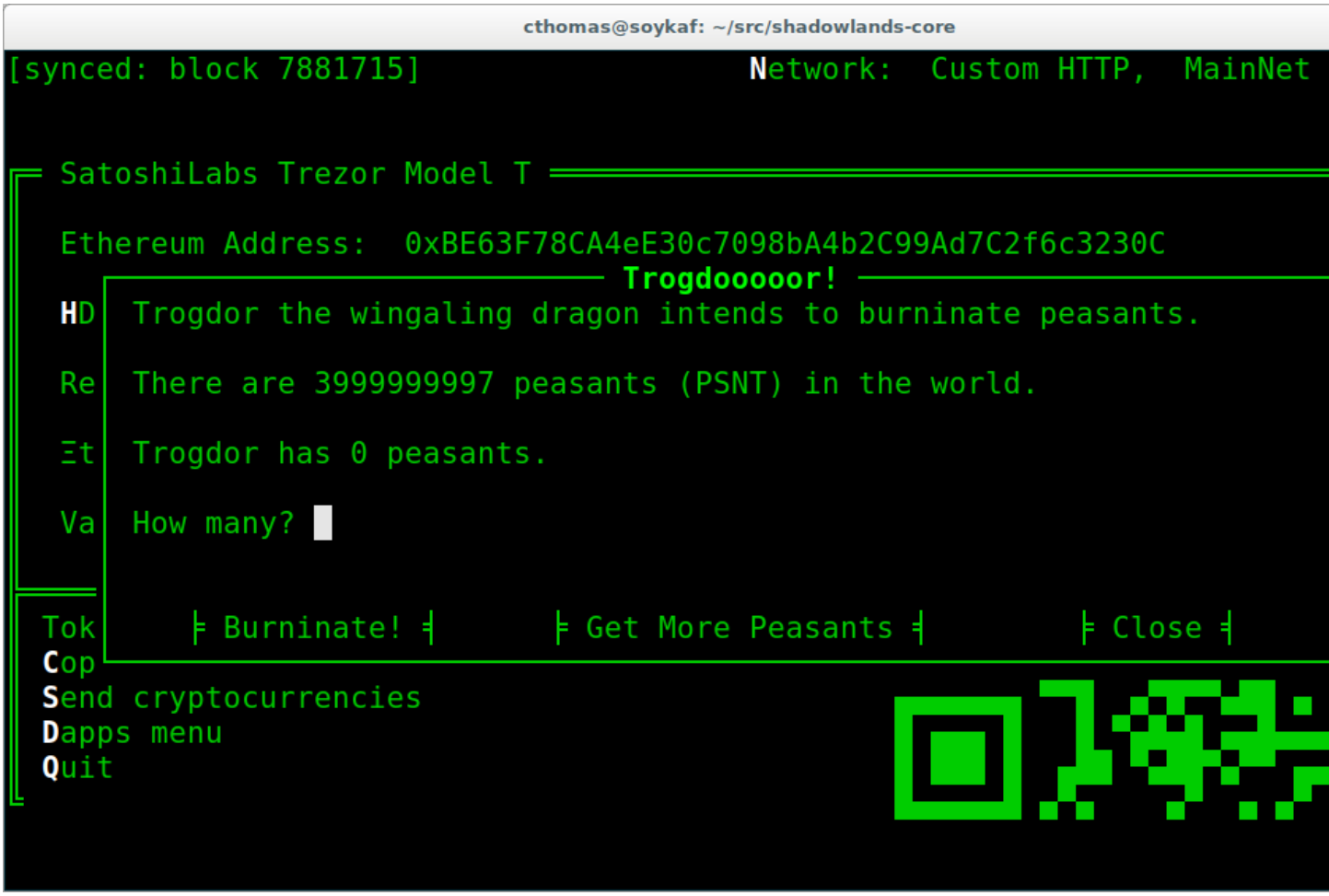
On line 7, we access the underlying function generated by web3.py with the `functions()` method. Now when we burn PSNT tokens, they will be taken out of the `totalSupply()`.

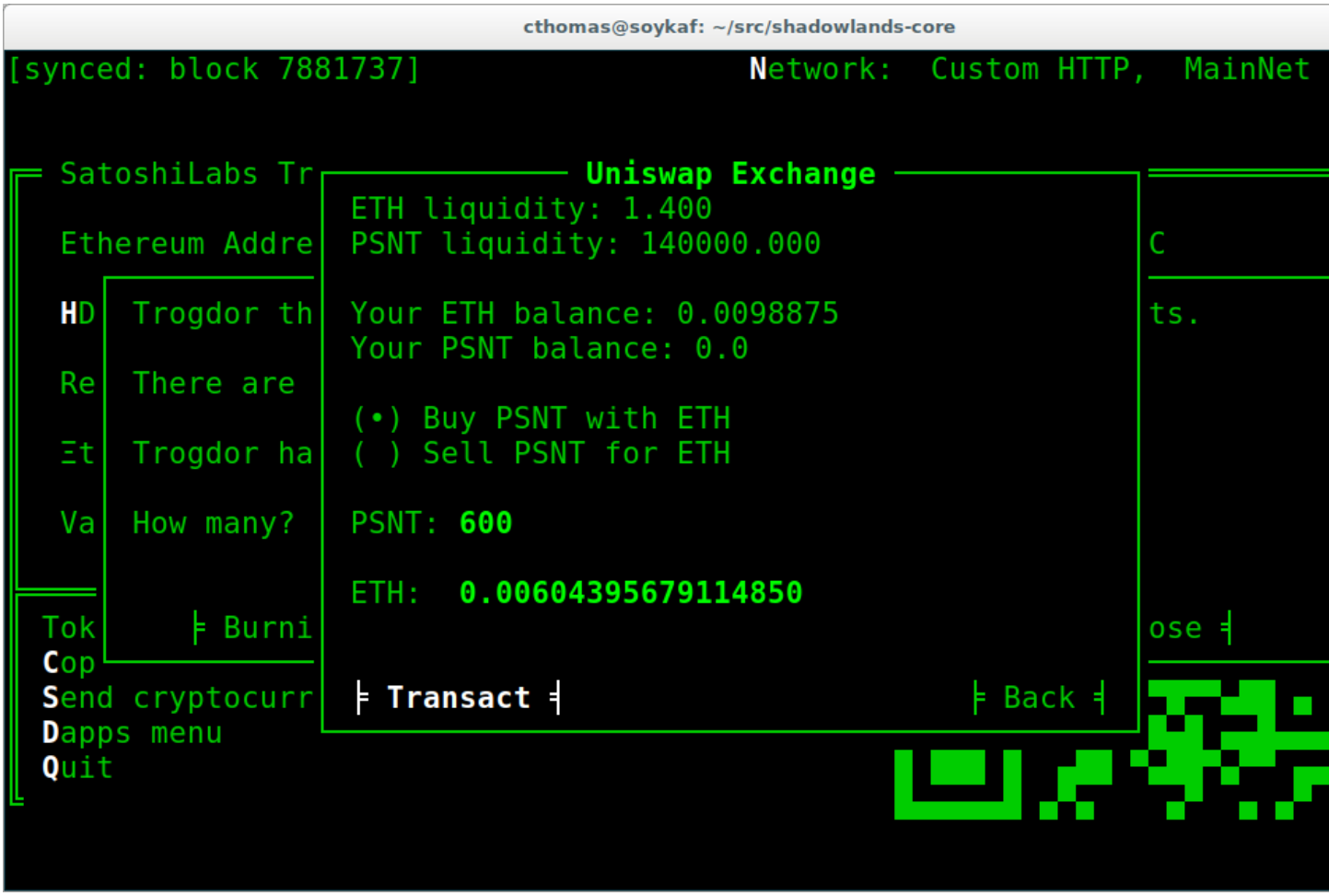
1.7 Uniswap Integration

Uh-oh, Trogdor has run out of peasants to burninate. What to do?

Shadowlands has native API integration with Uniswap, so let's add a button to acquire more PeasantCoin.

```
class MyMenuFrame(SLFrame):
    def initialize(self):
        self.add_label("Trogdor the wingaling dragon intends to burninate peasants.")
        self.add_label("There are {} peasants (PSNT) in the world.".format(
            self.peasant_decorator(self.dapp.total_peasants)
        ))
        self.add_label("Trogdor has {} peasants.".format(
            self.peasant_decorator(self.dapp.peasants)
        ))
        self.text_value = self.add_textbox("How many?")
        self.add_divider()
        self.add_button_row([
            ("Burninate!", self.burninate, 0),
            ("Get More Peasants", self.get_peasants, 1),
            ("Close", self.close, 2)
        ], layout=[30, 40, 30]
        )

    def get_peasants(self):
        self.dapp.add_uniswap_frame(self.dapp.peasant_contract.address)
```


cthomas@soykaf: ~/src/shadowlands-core
[synced: block 7881715]
Network: Custom HTTP, MainNet
SatoshiLabs Trezor Model T
Ethereum Address: 0xBE63F78CA4eE30c7098bA4b2C99Ad7C2f6c3230C
Trogdoooooor!
HD Trogdor the wingaling dragon intends to burninate peasants.
Re There are 3999999997 peasants (PSNT) in the world.
Et Trogdor has 0 peasants.
Va How many?
Tok
Burninate!
Get More Peasants
Close
Cop
Send cryptocurrencies
Dapps menu
Quit



1.8 The Hall of Maximum Burnination

What's the use of burninating peasants if nobody knows you did it? Let's create a leaderboard to show off our incindiary exploits.

There are a lot of additions here, but focus on lines 9 and 10, which checks for the victory condition upon startup.

We define the VictoryFrame class on line 98.

```
class Dapp(SLDapp):
    def initialize(self):
        self.token = PeasantCoin(self.node)
        self.peasants = Decimal(self.token.my_balance() / (10 ** 18))
        self.total_peasants =  self.token.totalSupply() / (10 ** 18)
        self.my_burninated_peasants = self.token.burninatedBy(self.node.credstick.
↪address) / (10 ** 18)
        self.add_sl_frame(MyMenuFrame(self, height=24, width=74))

        if self.token.victorious():
            self.add_sl_frame(VictoryFrame(self, height=9, width=62, title="Victory!!!
↪"))
```

(continues on next page)

(continued from previous page)

```
class MyMenuFrame(SLFrame):
    def initialize(self):
        self.add_label("The Hall Of Maximum Burnination", add_divider=False)
        self.add_divider(draw_line=True)
        self.add_label("Rank    Peasants           Hero", add_divider=False)

        for heroes in self.top_burninators_decorator():
            self.add_label(heroes, add_divider=False)
        self.add_divider(draw_line=True)

        self.add_label("Trogdor the wingaling dragon intends to burninate peasants.",␣
↪add_divider=False)
        self.add_label("There are {} peasants (BRNT) in the world.".format(self.
↪peasant_decorator(self.dapp.total_peasants)))
        self.add_label("Trogdor has {} peasants, and has burninated {}".format(self.
↪peasant_decorator(self.dapp.peasants), self.peasant_decorator(self.dapp.my_
↪burninated_peasants)))
        self.text_value = self.add_textbox("How many to burninate?", default_value='
↪')
        self.add_button_row([
            ("Burninate!", self.burninate, 0),
            ("Get More Peasants", self.get_peasants, 1),
            ("Close", self.close, 2)
        ], layout=[30, 40, 30]
        )


    def top_burninators_decorator(self):
        burninators = self.dapp.token.top_burninators()
        i = 0
        heroes = []

        #debug(); pdb.set_trace()
        for hero in burninators:
            hero_name = self.dapp.node._ns.name(hero[0])
            if hero_name is None:
                hero_name = hero[0]
            heroes.append("{}       {:14s}     {}".format(i, self.peasant_
↪decorator(hero[1]), hero_name))
            i += 1

        if len(heroes) < 10:
            for x in range(len(heroes), 10):
                heroes.append(
                    "{}       Unclaimed".format(str(x)))

        return heroes

    def peasant_decorator(self, peasants):
        return "{:f}".format(peasants)[:14]


    def get_peasants(self):
        self.dapp.add_uniswap_frame(self.dapp.token.address)

    def peasants_validated(self):
```

(continues on next page)

(continued from previous page)

```
        try:
            self.peasants_to_burninate = Decimal(self.text_value())
        except:
            self.dapp.add_message_dialog("That number of peasants doesn't make sense.
↪")
            return False

        if self.peasants_to_burninate > self.dapp.peasants:
            self.dapp.add_message_dialog("You don't even *have* that many peasants!")
            return False
        elif self.peasants_to_burninate < 0.5:
            self.dapp.add_message_dialog("This will not satisfy Trogdor.")
            return False

        return True



    def burninate(self):
        if not self.peasants_validated():
            return

        tokens_to_burninate = self.peasants_to_burninate * Decimal(10 ** 18)

        burn_fn = self.dapp.token.burninate(
            int(tokens_to_burninate)
        )

        self.dapp.add_transaction_dialog(
            burn_fn,
            title="Trogdor burninates the tokens",
            gas_limit=56000
        )

        self.close()

class VictoryFrame(SLFrame):
    def initialize(self):
        self.add_label("Congratulations!  You have racked up a truly impressive", add_
↪divider=False)
        self.add_label("count of {} burninated peasants, as well".format(self.peasant_
↪decorator(self.dapp.my_burninated_peasants)), add_divider=False)
        self.add_label("as several incinerated thatched roof cottages and various",␣
↪add_divider=False)
        self.add_label("counts of petty theft and vandalism.  Your throne in the",␣
↪add_divider=False)
        self.add_label("Hall of Maximum Burnination awaits your Dragonly Personage!")
        self.add_button_row(
            [("Claim Victoriousness", self.claim_victory, 0),
            ("Back", self.close, 1)],
            layout=[50, 50],
        )

    def claim_victory(self):
        self.dapp.add_transaction_dialog(
            self.dapp.token.claimVictory(),
            title="Claiming victory",
```

(continues on next page)

(continued from previous page)

```
            gas_limit=100000
        )
        self.close()

    def peasant_decorator(self, peasants):
        return "{:f}".format(peasants)[:14]
```

1.8.1 A closer look at the peasantcoin contract

The *Erc20* class and its SLContract base class give you a great deal of functionality for free, but it's often useful to add on some extra methods that have close connection to our contract calls.

The `self.functions()` is an easy way to get at the underlying function of the web3.py contract class.

Erc20 subclasses also provide passthrough methods to all standard erc20 functions, as well as helper methods like `my_balance()`

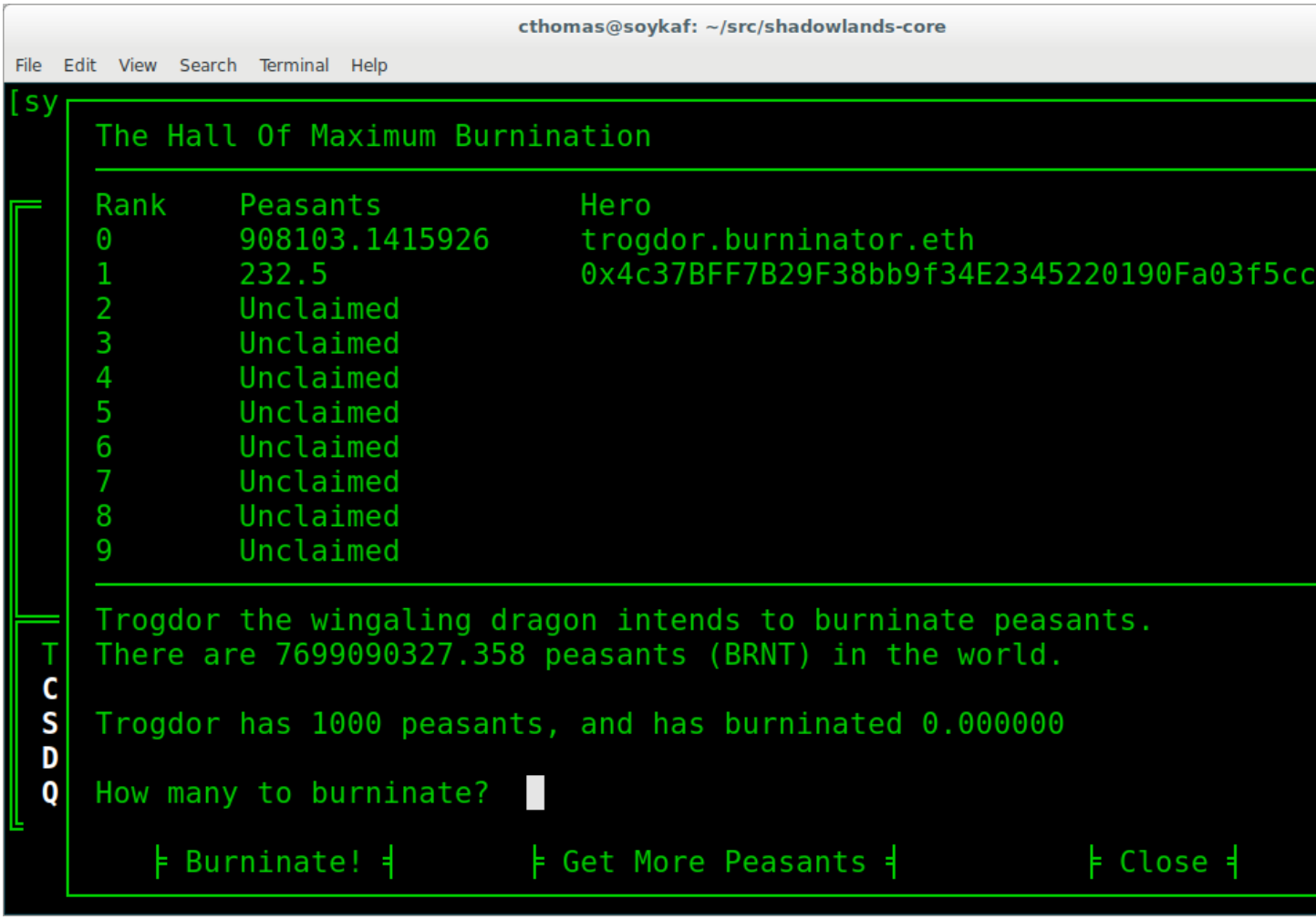
```
from shadowlands.sl_contract.erc20 import Erc20
from shadowlands.tui.debug import debug, end_debug
from decimal import Decimal
import pdb


class PeasantCoin(Erc20):

    ### Passthrough calls to contract

    # Similar to balanceOf, but keeps track of burninated peasants
    def burninatedBy(self, address):
        return self.functions.burninatedBy(address).call()

    def topBurninators(self):
        return self.functions.topBurninators().call()


    ### Helper methods

    def my_burninated_peasants(self):
        return self.burninatedBy(self.node.credstick.address)


    def top_burninators(self):
        '''
        Returns a sorted list of lists of integers and addresses,
        representing the top burninators. Maximum 10 results.
        '''
        burninators = set(self.topBurninators())
        burninators.remove('0x0000000000000000000000000000000000000000')
        if len(burninators) == 0:
            return []

        burninators = [[x, Decimal(self.burninatedBy(x)) / (10 ** 18)] for x in␣
↪list(burninators)]
        burninators.sort(key=lambda x: x[1], reverse=True)
        return burninators
```

(continues on next page)

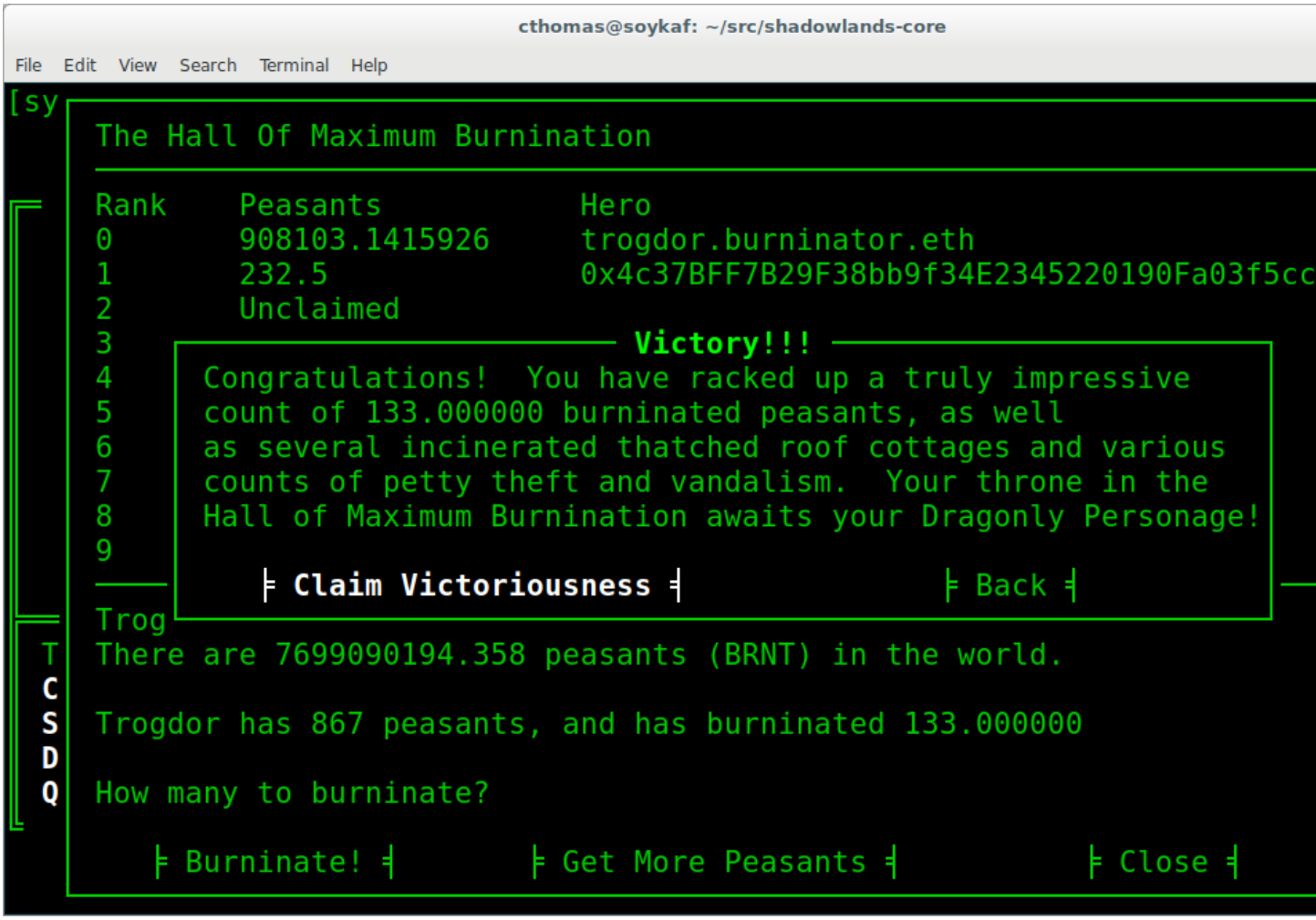
(continued from previous page)

```
    def victorious(self):
        '''
        Returns True or False.
        True only if user is not currently in the hall but is
        allowed to take a spot.
        '''
        if self.my_burninated_peasants() == 0:
            return False

        # Are we already in the hall of max burnination?
        if self.node.credstick.address in [self.topBurninators()]:
            return False

        if len(top) < 10:
            return True

        # Weakest burninator first
        top = self.top_burninators()
        top.sort(key=lambda x: x[1])

        if top[0][1] < Decimal(self.my_burninated_peasants()) / 10 ** 18:
            return True
        return False


    ### TXs
    def burninate(self, peasants):
        return self.functions.burn(peasants)


    def claimVictory(self):
        return self.functions.claimVictory()

    ABI='''
    [{"name":"Transfer","inputs":[{"type":"addre...
    '''
```

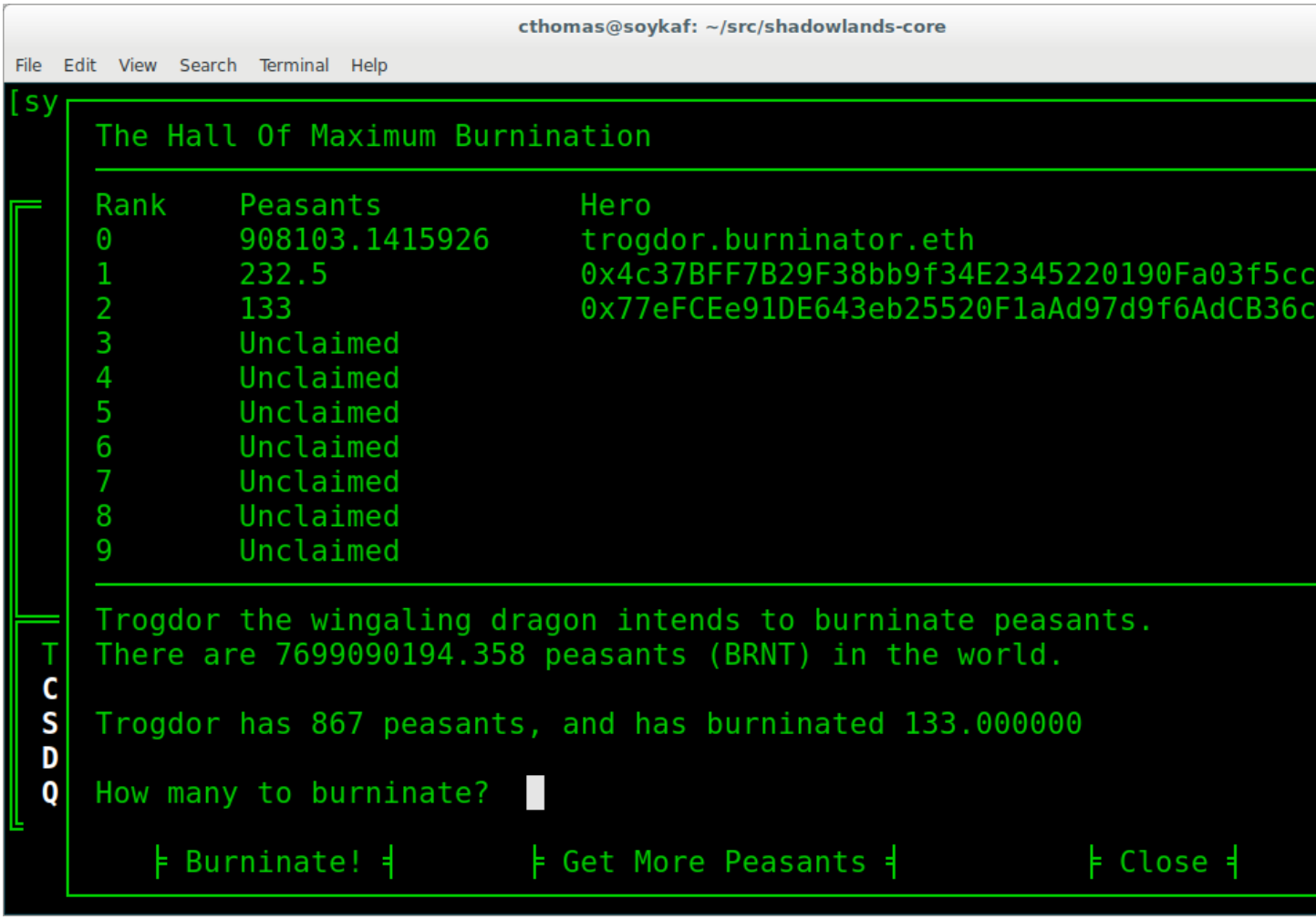
Here is the hall of Maximum Burnination, in all its glory:



And so, 133 fiery peasants later (and after restarting the dapp)…



And so, after the transaction is run and we restart the app. . .



Huzzah! We are immortal - for the time being.

1.9 Making your dapp update dynamically

It's hella lame that we have to keep restarting the app in order to react to changes on the blockchain. Luckily, help is on the way.

The label widgets on shadowlands can take either a string, or a function reference (or lambda) that returns a string. That will let us make the displays dynamic, but it also can make your dapp VERY SLOW.

To help solve this performance problem, the *SLDapp* and *SLFrame* classes will automatically expire the cache on any `@cached_property` when a new block appears.

Using lambdas to cached properties as input to labels combines the best of both worlds - any function reference you pass to a label will be both dynamic and reasonably performant.

In addition, *SLDapp* and *SLFrame* will both trigger the `new_block_callback()` which you can override for your own purposes. This callback will be called immediately after the cached properties are expired,

Let's put our informational display strings in cached properties to let the app update dynamically. We can also implement `new_block_callback()` to make the victory frame pop up when appropriate.

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame
from burninator.peasant_coin import PeasantCoin
from decimal import Decimal
from cached_property import cached_property
from shadowlands.tui.debug import debug, end_debug
import pdb

class Dapp(SLDapp):
    def initialize(self):
        self.token = PeasantCoin(self.node)
        self.add_sl_frame(MyMenuFrame(self, height=24, width=74 ))

        self.victory_notification_has_been_seen = False
        self.victory_check()

    def victorious_check(self):
        if self.victory_notification_has_been_seen:
            return

        if self.token.victorious():
            self.add_sl_frame(VictoryFrame(self, height=9, width=62, title="Victory!!!
↪"))
            self.victory_notification_has_been_seen = True


    def new_block_callback(self):
        self.victory_check()

    @cached_property
    def total_peasants(self):
        return self.token.totalSupply() / (10 ** 18)

    @cached_property
    def my_burninated_peasants(self):
        return self.token.burninatedBy(self.node.credstick.address) / (10 ** 18)

    @cached_property
    def peasants(self):
        return Decimal(self.token.my_balance() / (10 ** 18))

    def peasant_decorator(self, peasants):
        return "{:f}".format(peasants)[:14]
```

Here we implement `new_block_callback()` and use it to call `victory_check()`. It may be useful to know that `new_block_callback()` is called immediately after the cache is expired.

On line 5 we import the `cached_property` decorator.

We declare most of the dapp variables as `@cached_property` now - this will let them update dynamically, as well as keeping performant when any other classes in the dapp need to reference them.

```
class MyMenuFrame(SLFrame):
    def initialize(self):
        self.add_label("The Hall Of Maximum Burnination", add_divider=False)
        self.add_divider(draw_line=True)
        self.add_label("Rank    Peasants           Hero", add_divider=False)
```

(continues on next page)

(continued from previous page)

```
        for i in range(10):
            self.add_label(self.burninator_hero(i), add_divider=False)
        self.add_divider(draw_line=True)

        self.add_label("Trogdor the wingaling dragon intends to burninate peasants.",␣
↪add_divider=False)
        self.add_label(lambda: self.total_peasants_string)
        self.add_label(lambda: self.my_peasant_status_string)
        self.text_value = self.add_textbox("How many to burninate?", default_value='
↪')
        self.add_button_row([
            ("Burninate!", self.burninate, 0),
            ("Get More Peasants", self.get_peasants, 1),
            ("Close", self.close, 2)
        ], layout=[30, 40, 30]
        )

    @cached_property
    def total_peasants_string(self):
        return "There are {} peasants (BRNT) in the world.".format(self.dapp.peasant_
↪decorator(self.dapp.total_peasants))

    @cached_property
    def my_peasant_status_string(self):
        return "Trogdor has {} peasants, and has burninated {}".format(self.dapp.
↪peasant_decorator(self.dapp.peasants), self.dapp.peasant_decorator(self.dapp.my_
↪burninated_peasants))

    def burninator_hero(self, index):
        return lambda: self.top_burninators_decorator[index]

    @cached_property
    def top_burninators_decorator(self):
        burninators = self.dapp.token.top_burninators()
        i = 0
        heroes = []

        for hero in burninators:
            hero_name = self.dapp.node._ns.name(hero[0])
            if hero_name is None:
                hero_name = hero[0]
            heroes.append("{}       {:14s}     {}".format(i, self.dapp.peasant_
↪decorator(hero[1]), hero_name))
            i += 1

        if len(heroes) < 10:
            for x in range(len(heroes), 10):
                heroes.append(
                    "{}       Unclaimed".format(str(x)))

        return heroes
```

The magic happens on lines 12 and 13, where we send a `lambda: self.property_name` into the labels. I had to get a little bit fancy at line 8 and call a function to return lambdas that index the array returned by the cached property `top_burninators_decorator`.

And now our app updates live.

1.10 Tutorial Source Code

The source code for the Burninator app is available on github at https://github.com/kayagoban/burninator

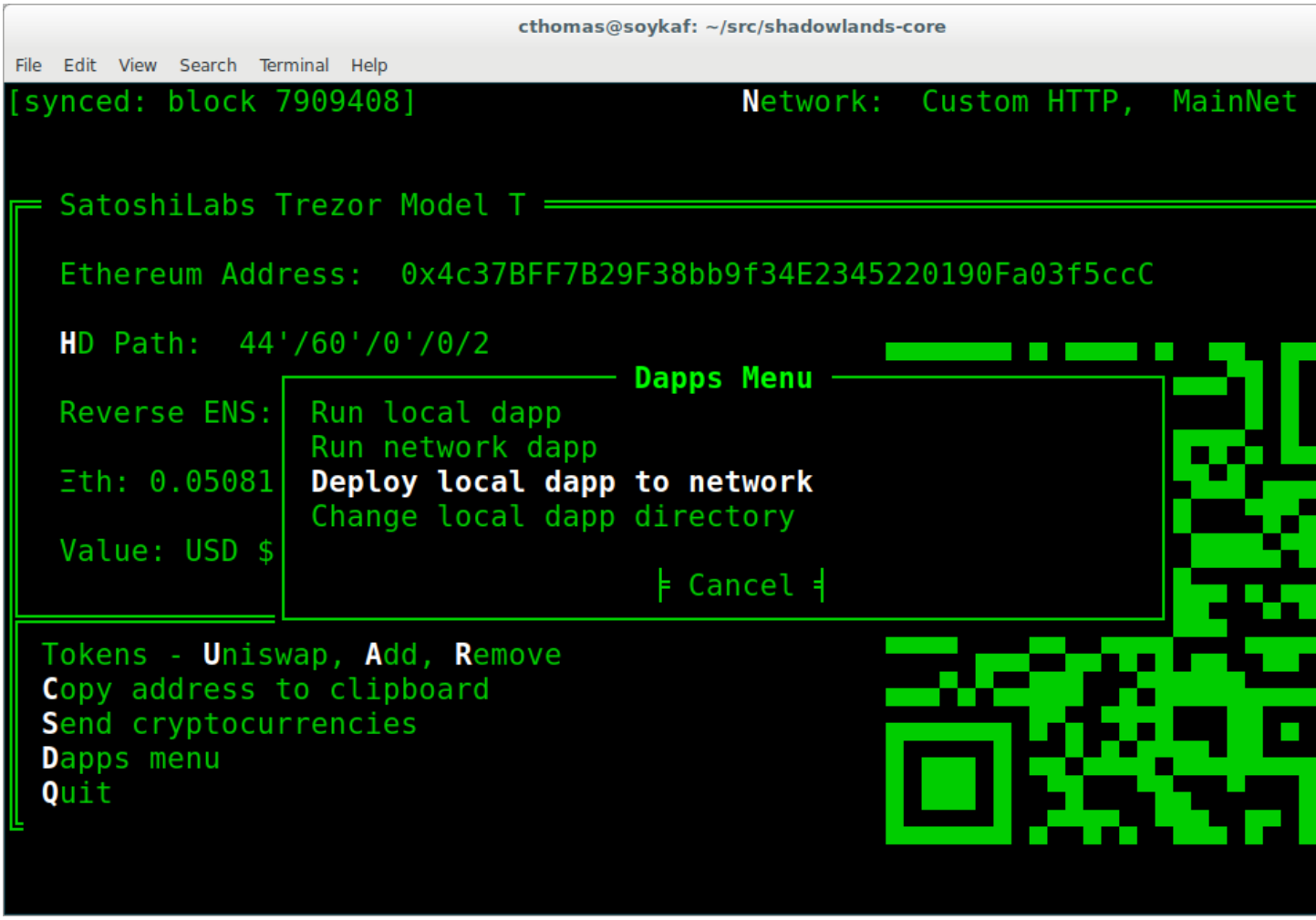
1.11 Deploying your dapp

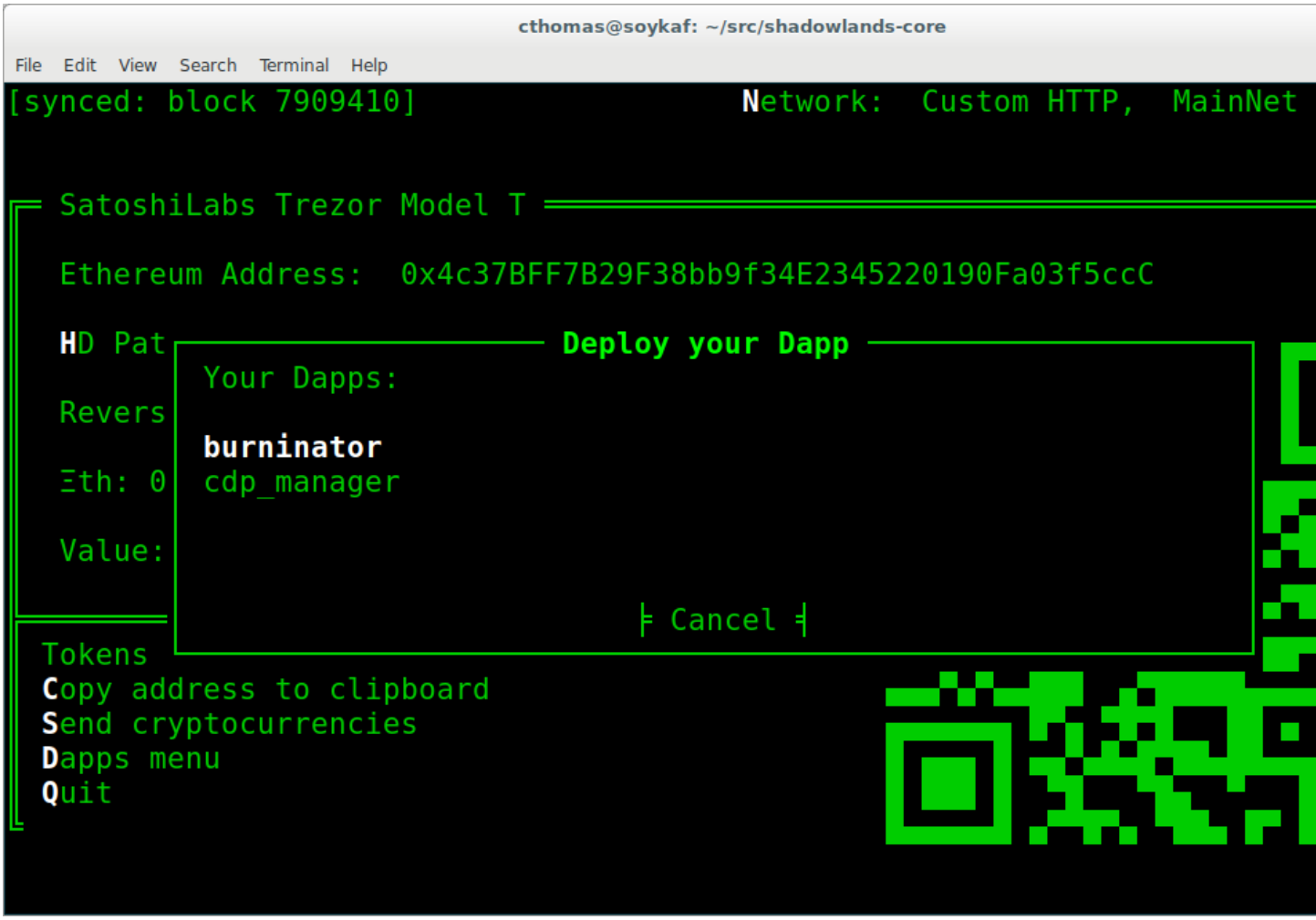
Shadowlands has a package management contract at `sloader.shadowlands.eth` that allows you to deploy your dapp so you can share it with the world.

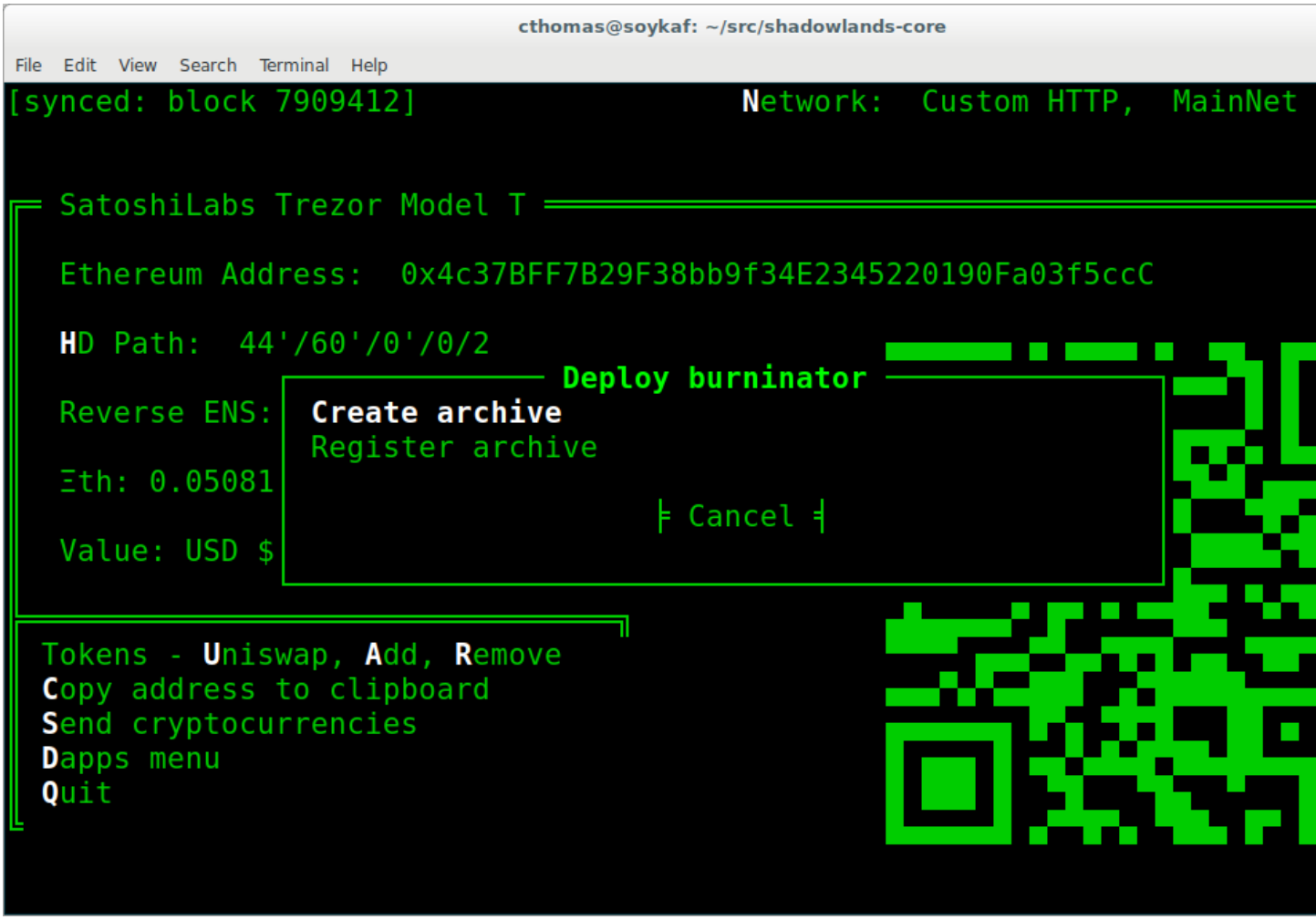
Once registered, anyone can run your dapp by using the ethereum address you used to register the software (they can also reference your ENS, which is much nicer). If you want to register more than one dapp, use a different address to register each.

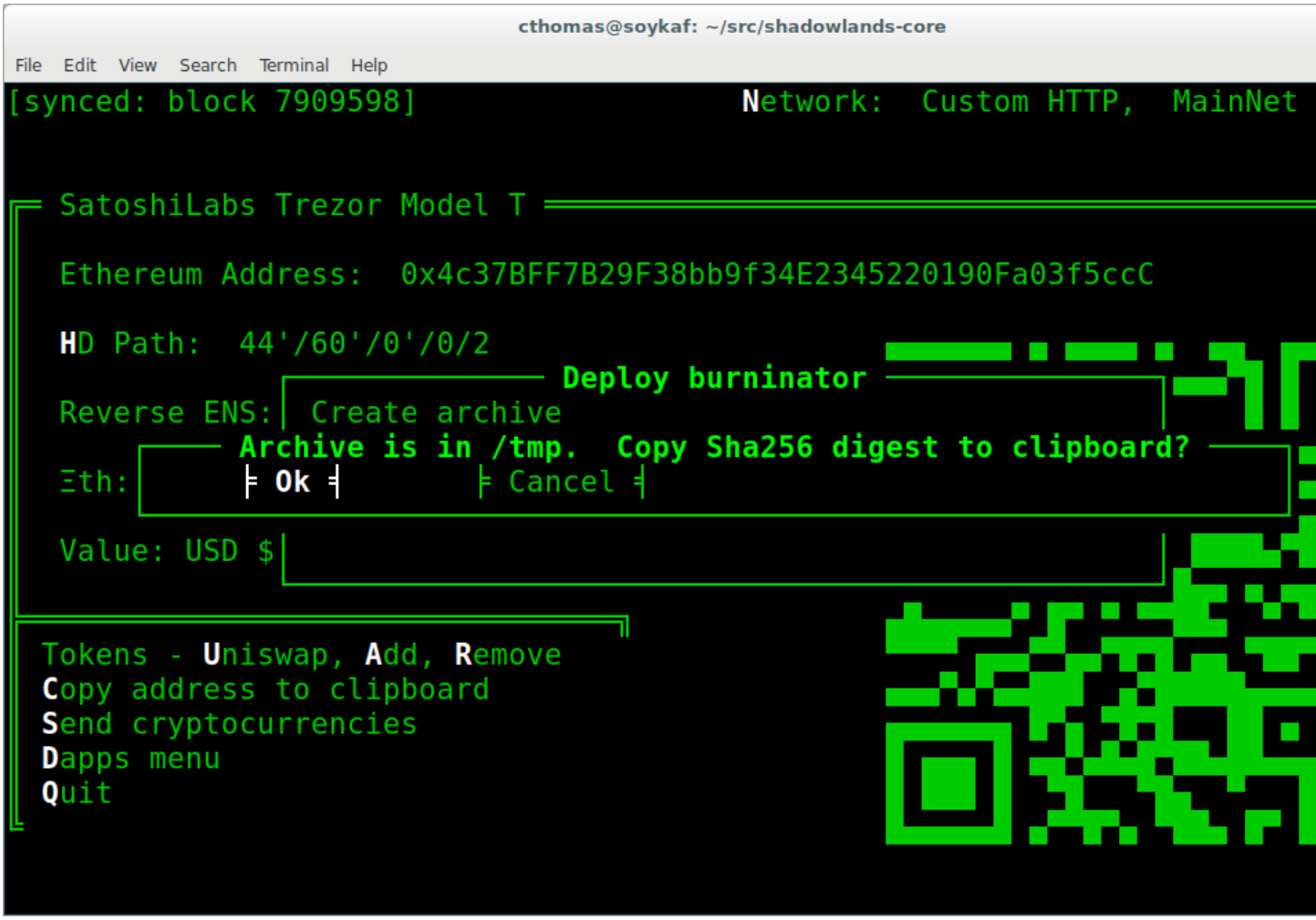
In this example, I am using address `0x4c37BFF7B29F38bb9f34E2345220190Fa03f5ccC` which is is resolved by the ENS name `burninator.eth`.

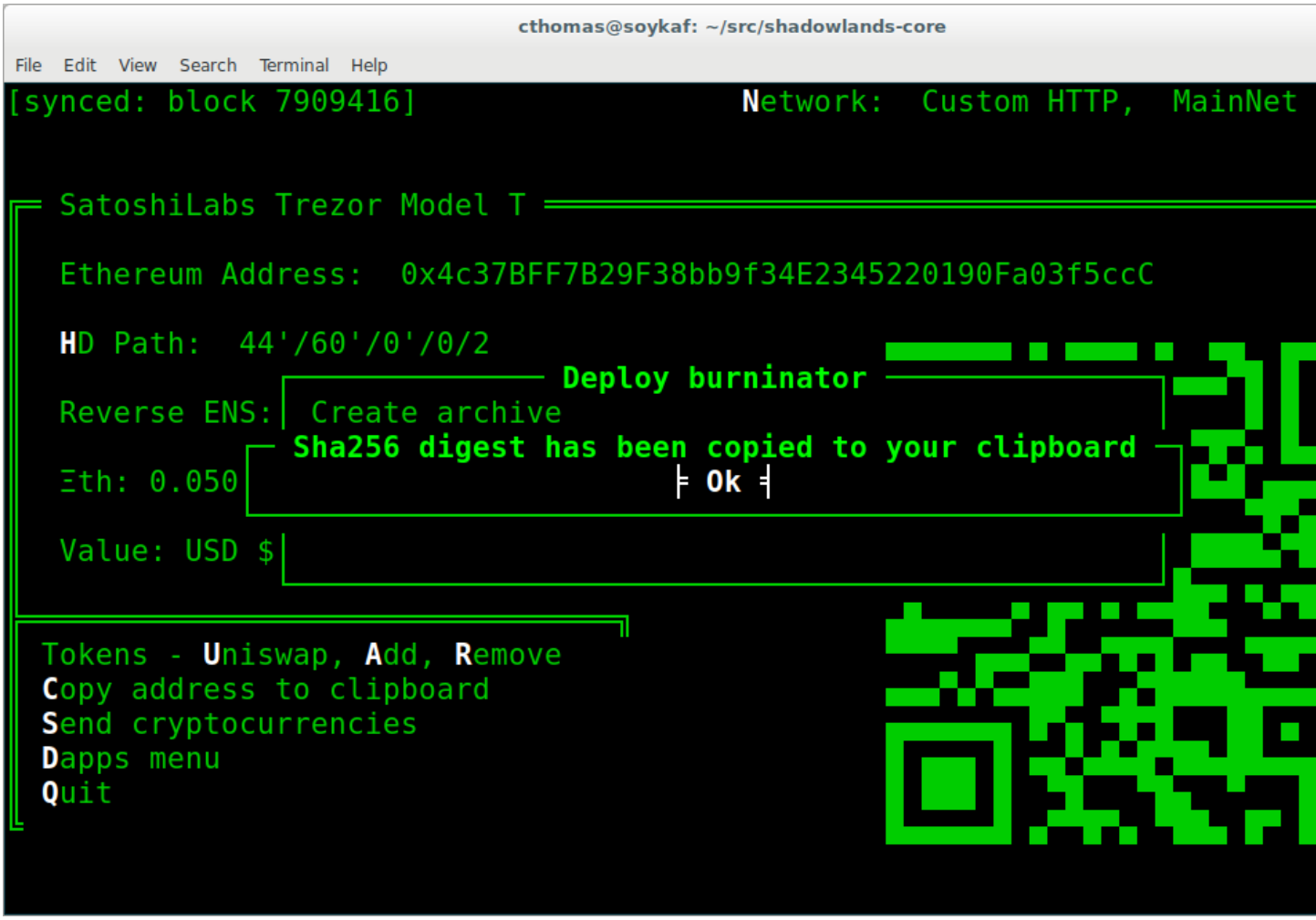
Select the `Deploy local dapp to network` from the Dapps menu.



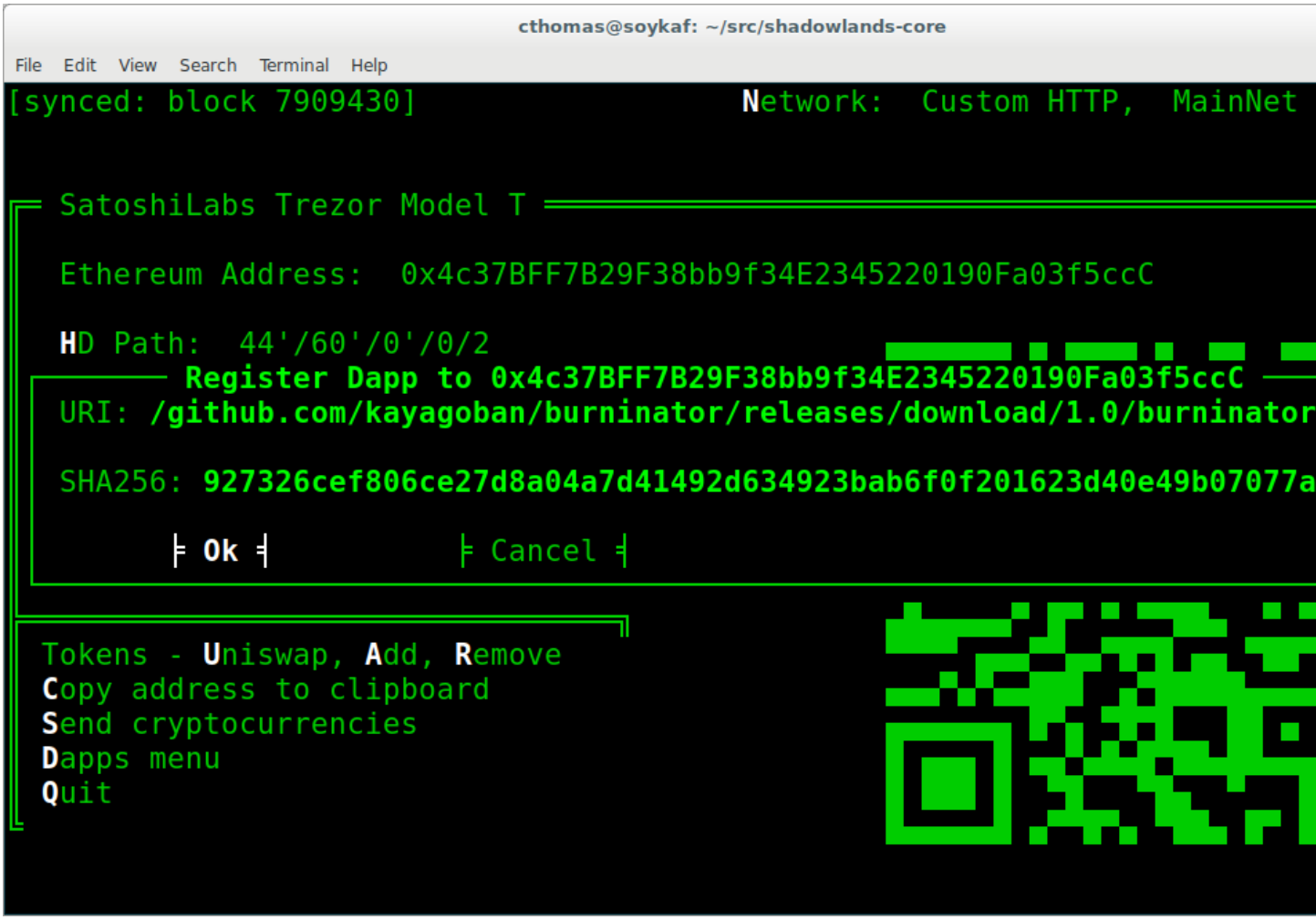

cthomas@soykaf: ~/src/shadowlands-core
File Edit View Search Terminal Help
[synced: block 7909410] Network: Custom HTTP, MainNet
SatoshiLabs Trezor Model T
Ethereum Address: 0x4c37BFF7B29F38bb9f34E2345220190Fa03f5ccC
HD Pat
Revers
Ξth: 0
Value:
Deploy your Dapp
Your Dapps:
burninator
cdp_manager
Cancel
Tokens
Copy address to clipboard
Send cryptocurrencies
Dapps menu
Quit


cthomas@soykaf: ~/src/shadowlands-core
File Edit View Search Terminal Help
[synced: block 7909412]
Network: Custom HTTP, MainNet
SatoshiLabs Trezor Model T
Ethereum Address: 0x4c37BFF7B29F38bb9f34E2345220190Fa03f5ccC
HD Path: 44'/60'/0'/0/2
Reverse ENS:
Ξth: 0.05081
Value: USD $
Deploy burninator
Create archive
Register archive
Cancel
Tokens - Uniswap, Add, Remove
Copy address to clipboard
Send cryptocurrencies
Dapps menu
Quit

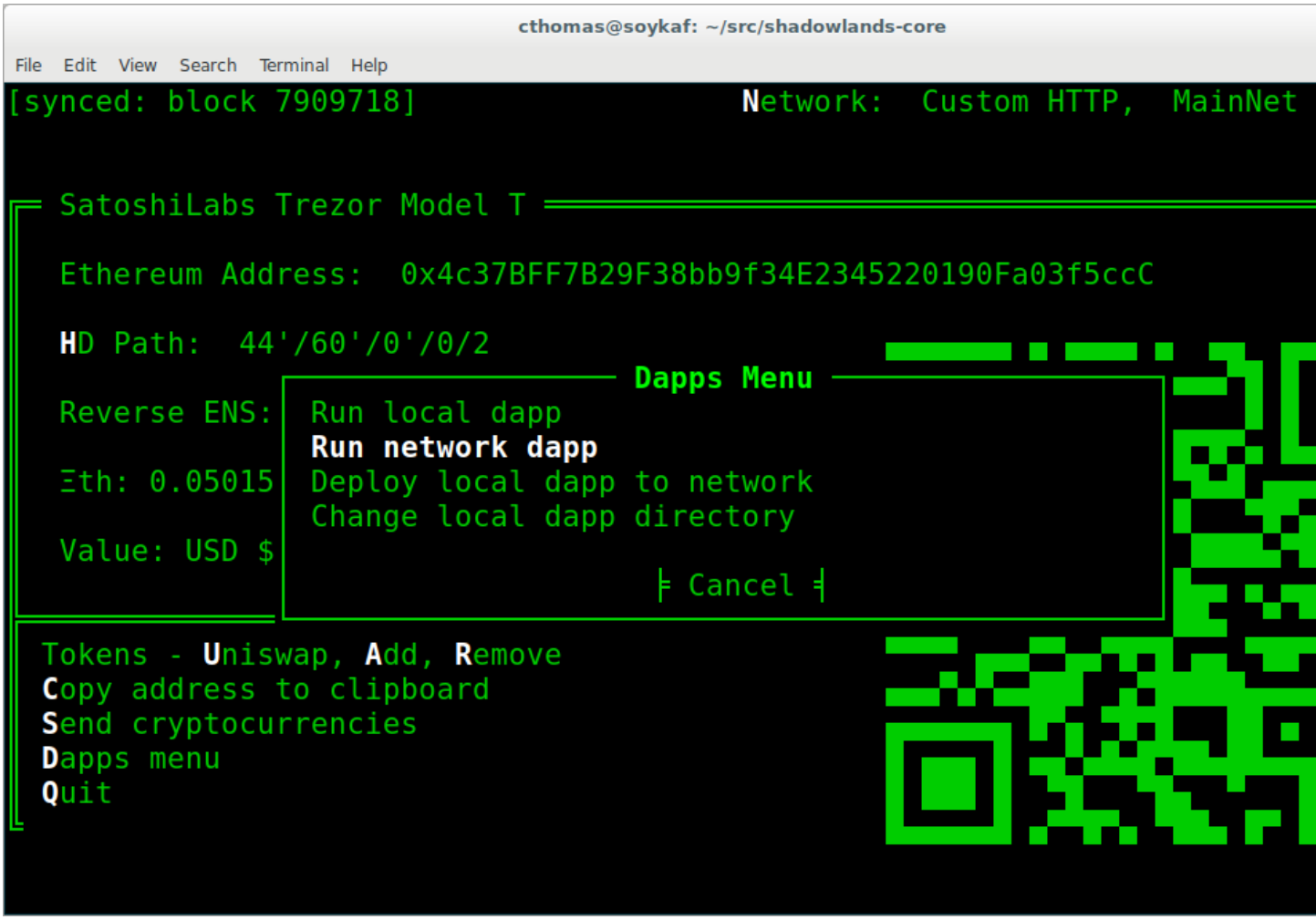

cthomas@soykaf: ~/src/shadowlands-core
File Edit View Search Terminal Help
[synced: block 7909598] Network: Custom HTTP, MainNet
SatoshiLabs Trezor Model T
Ethereum Address: 0x4c37BFF7B29F38bb9f34E2345220190Fa03f5ccC
HD Path: 44'/60'/0'/0/2
Deploy burninator
Reverse ENS: Create archive
Archive is in /tmp. Copy Sha256 digest to clipboard?
Eth: Ok Cancel
Value: USD $
Tokens - Uniswap, Add, Remove
Copy address to clipboard
Send cryptocurrencies
Dapps menu
Quit

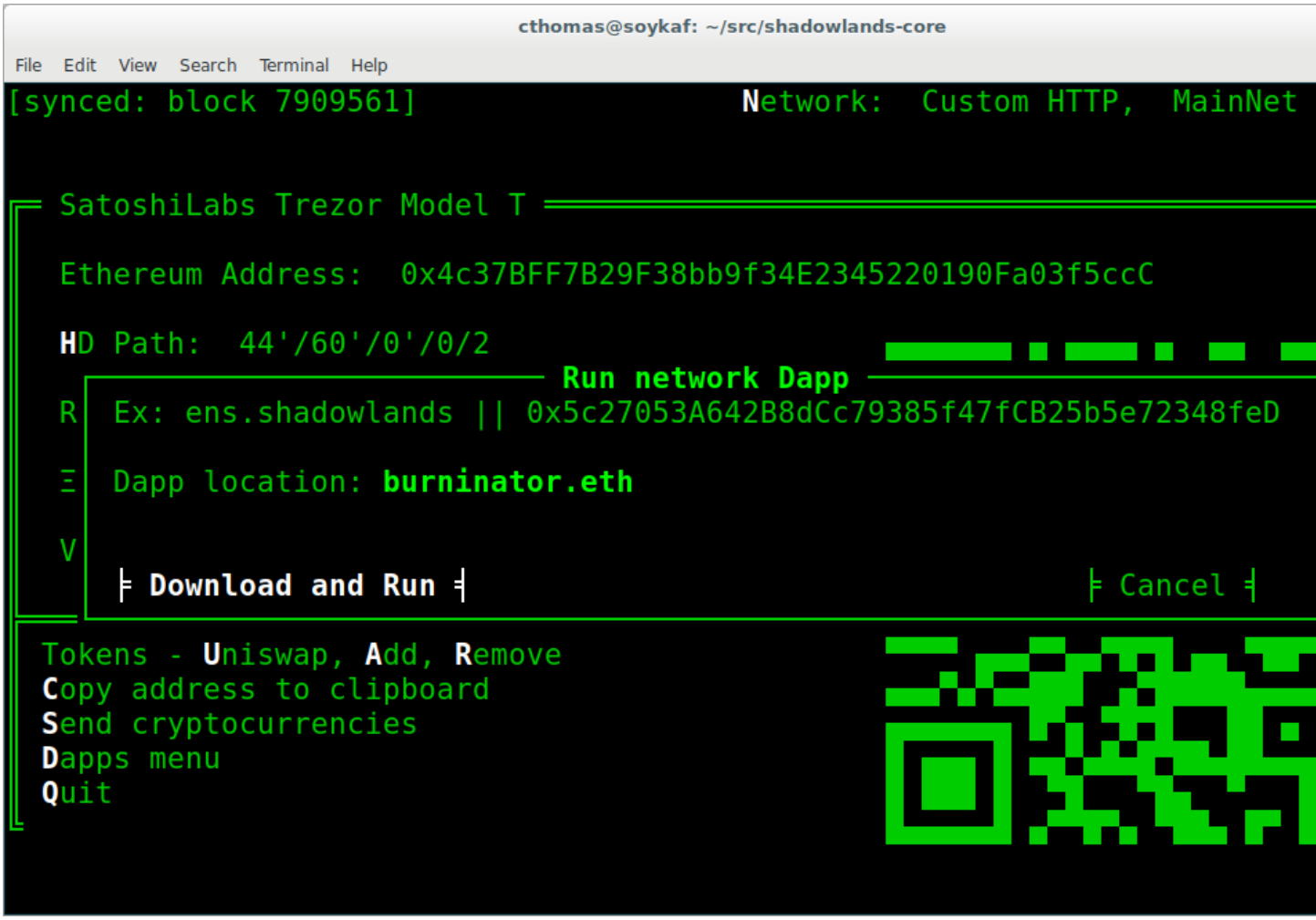


Now, copy the zip file to some place on the internet. Do *not* use the zip files generated by github releases - for some reason the ZipImporter class does not accept them as legit zip files. You can add the shadowlands-generated zip file to a github release though - this is an easy way to get free hosting.

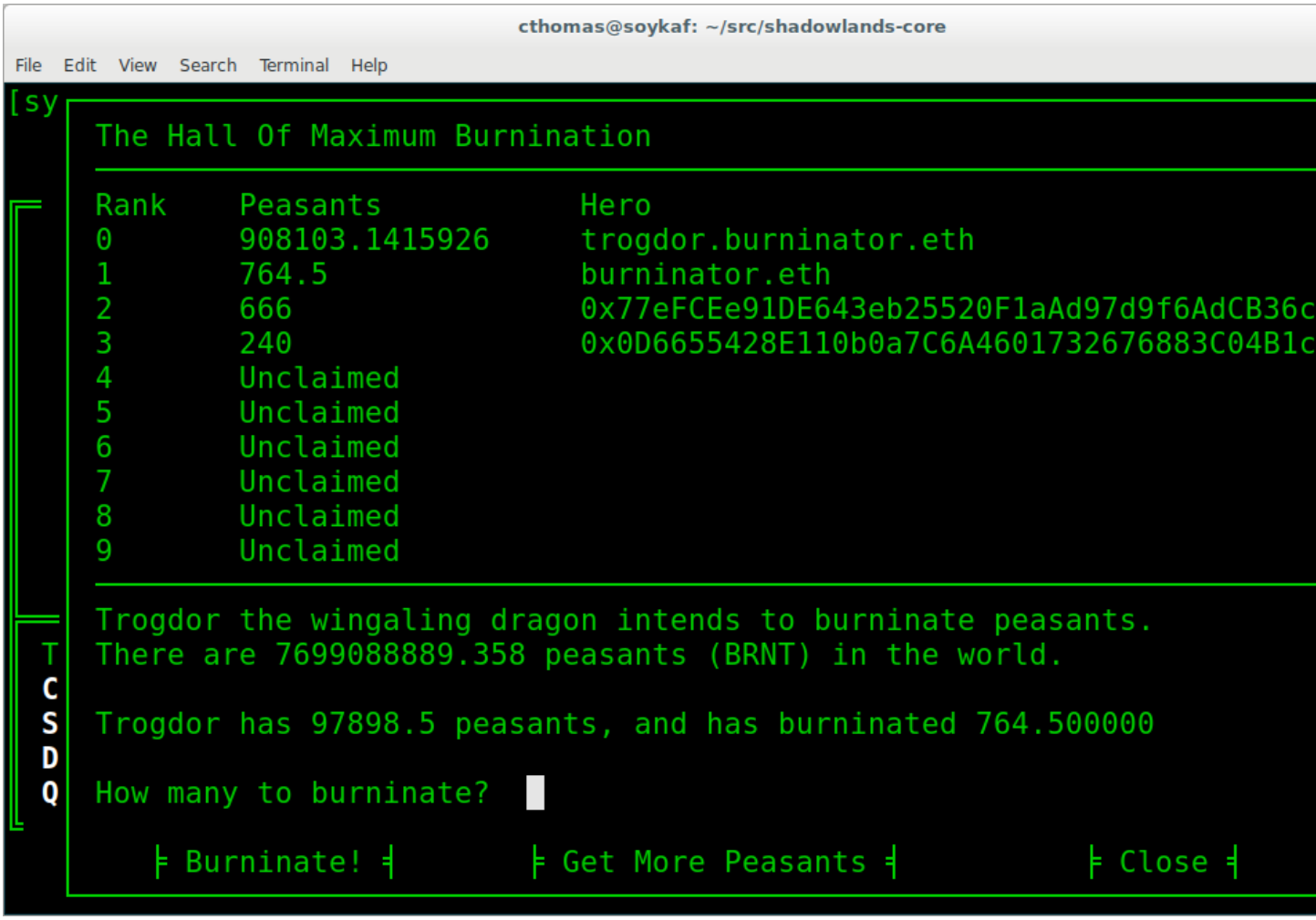


Now you will register this URL and checksum. Once the TX is mined, anyone can run your app:


cthomas@soykaf: ~/src/shadowlands-core
File Edit View Search Terminal Help
[synced: block 7909718] Network: Custom HTTP, MainNet
SatoshiLabs Trezor Model T
Ethereum Address: 0x4c37BFF7B29F38bb9f34E2345220190Fa03f5ccC
HD Path: 44'/60'/0'/0/2
Reverse ENS:
Ξth: 0.05015
Value: USD $
Dapps Menu
Run local dapp
Run network dapp
Deploy local dapp to network
Change local dapp directory
Cancel
Tokens - Uniswap, Add, Remove
Copy address to clipboard
Send cryptocurrencies
Dapps menu
Quit



And there we are!



1.12 Disclaimer

No peasants were harmed during the writing of this tutorial.

CHAPTER 2

SLDapp

class SLDapp

2.1 Abstract

SLDapp is the class which defines a Shadowlands Dapp. It provides many useful properties and methods to make writing dapps easier.

See the *Tutorial* to get started quickly.

Listing 1: Example

```
from shadowlands.sl_dapp import SLDapp

class Dapp(SLDapp):
    def initialize(self):
        self.add_message_dialog("Hello world!")
```

2.2 Properties

SLDapp.**w3**
Read-only property. A web3 object as provided by the web3.py framework.

SLDapp.**node**
Read-only property. An instance of *Node*.

```
# Find your address
my_address = self.node.credstick.address
```

SLDapp.**config_key**
A string to use as a key for storing config properties. Defaults to the name of your dapp module.

Feel free to change this to something very unique at the top of your *initialize()* method.

SLDapp.**config_properties**
Read-only property. A persistent dictionary of properties specific to your dapp. To load a single property, use *SLDapp.load_config_property*.

2.3 Methods

SLDapp.**initialize**()
An abstract callback that you must implement. It will fire upon the initialization of the SLDapp object. Do your setup here and add SLFrames or other dialogs.

SLDapp.**new_block_callback**()
An optional callback that you may implement. It will be fired when new blocks appear.

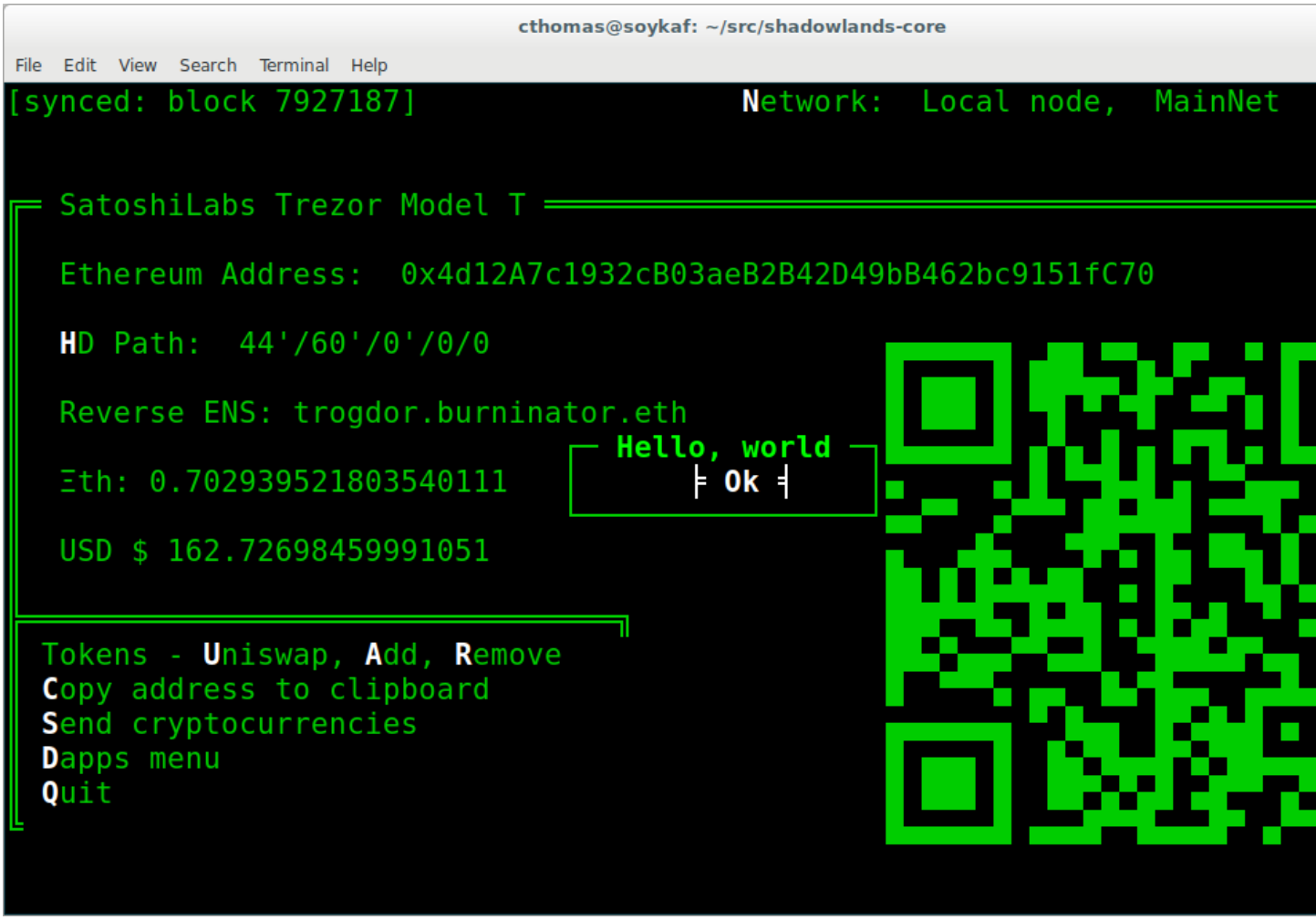
SLDapp.**add_sl_frame**(*sl_frame*)
Display a custom frame. Takes an instantiated subclass of *SLFrame* as the sole argument.

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame

class Dapp(SLDapp):
    def initialize(self):
        myframe = MyFrame(self, 5, 50, title="frame title")
        self.add_sl_frame(myframe)

class MyFrame(SLFrame):
    def initialize(self):
        self.add_button(self.close, "Select")
```

SLDapp.**add_message_dialog**(*message*, ***kwargs*)

Display a message dialog with the string supplied by `message`. You may pass in kwargs which apply to `asciimatics.Frame`.

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame

class Dapp(SLDapp):
    def initialize(self):
        self.add_message_dialog("Hello, world")
```



`SLDapp.`**`add_transaction_dialog`**(*tx_fn*, *tx_value=0*, *gas_limit=300000*, *title="Sign & Send Transaction"*, *destroy_window=None*, ***kwargs*)

Display a transaction dialog, which allows the user to select gas price and gives a gas cost estimate.

You must pass in a transaction function to `tx_fn` as the first argument. Instances of *Erc20* have many build-in methods which return transaction functions. You can also access the underlying function generators of any *SLContract* instance with *SLContract.functions()*.

You can provide a `tx_value` - Decimal value denominated in Ether.

You may pass in an integer `gas_limit`, which defaults to 300000. It is best practice to always set this.

A string `title` can be set.

If there is a frame which needs to be programmatically destroyed upon the exit of the transaction dialog, pass the object into `destroy_window`.

You may pass in kwargs which apply to `asciimatics.Frame`.

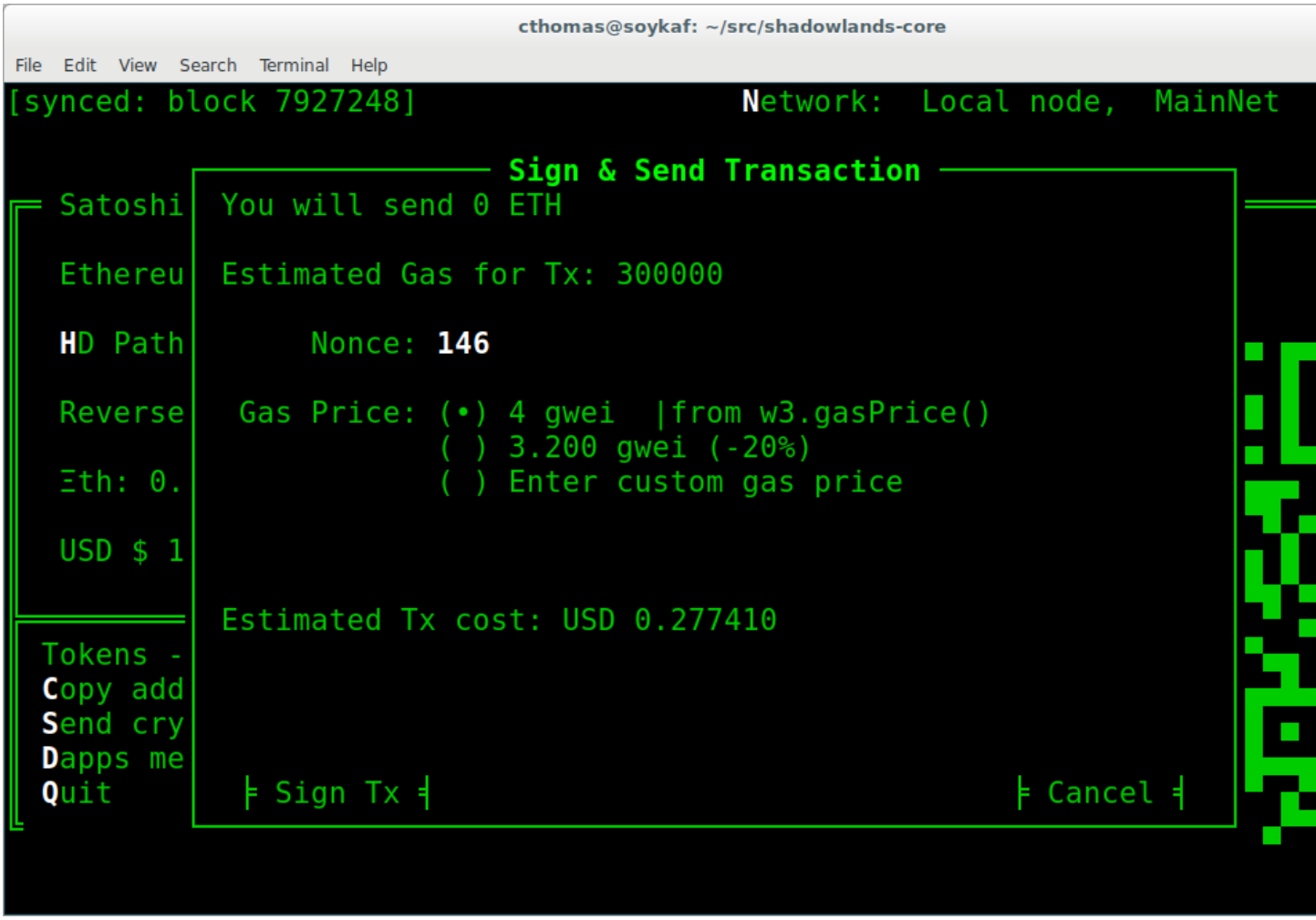
Listing 2: Example

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame
from shadowlands.sl_contract.erc20 import Erc20
```

(continues on next page)

(continued from previous page)

```
    class Dapp(SLDapp):
        def initialize(self):
            token = Erc20(
                self.node,
                address='0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359'
            )
            tx_fn = token.transfer(
                self.node.credstick.address, 1 * (10 ** token.decimals())
            )
            # we send ourselves 1.0 token
            self.add_transaction_dialog(tx_fn)
```



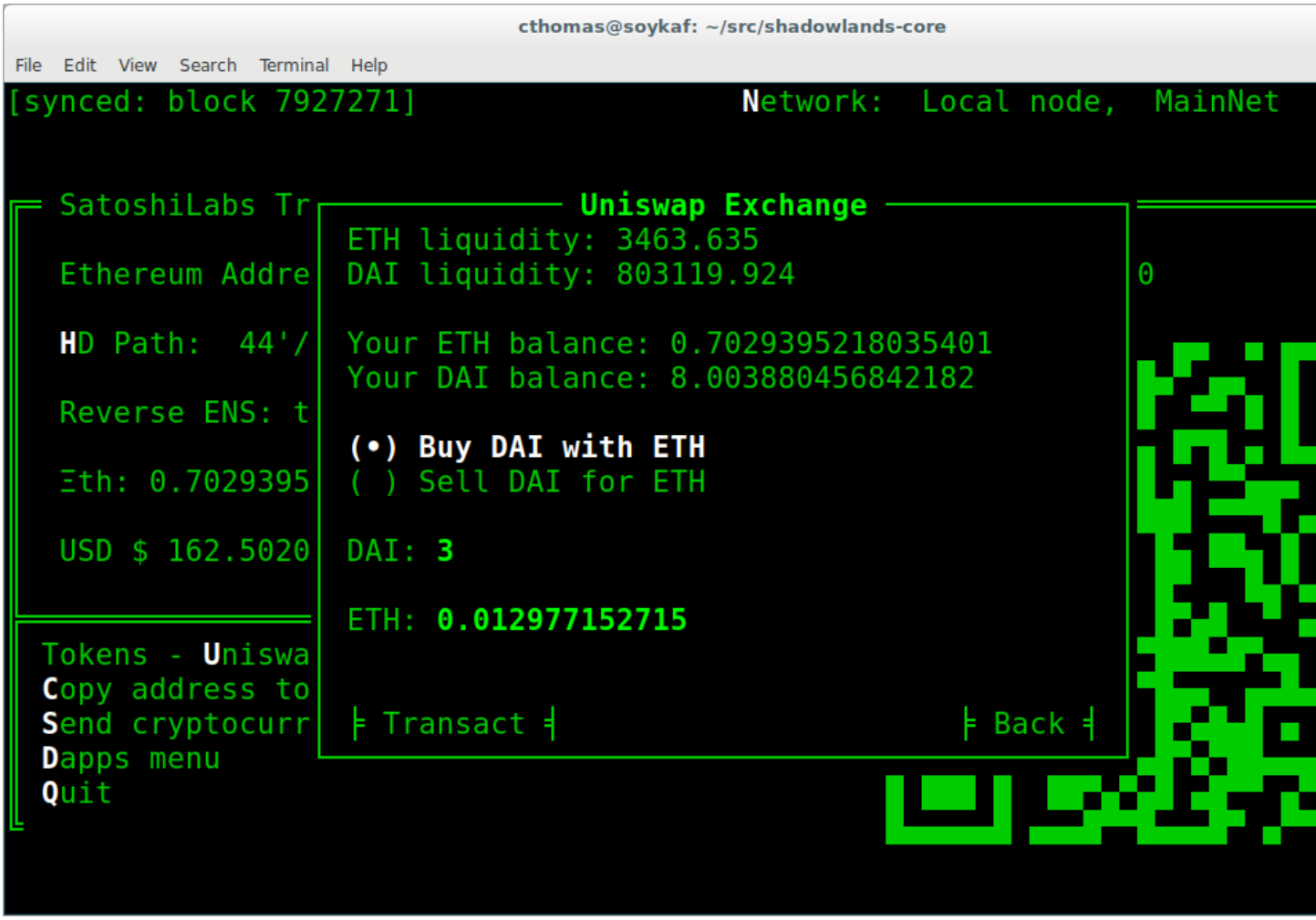
SLDapp.**add_uniswap_frame**(*ec20_address*, *action='buy'*, *buy_amount=''*, *sell_amount=''*)

Adds a Uniswap dialog if there exists a Uniswap exchange for the Erc20 token which resides at *erc20_address*.

If no Exchange exists, a dialog will be displayed, informing the user of this.

Listing 3: Example

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame

class Dapp(SLDapp):
    def initialize(self):
        address='0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359'
        self.add_uniswap_frame(address, buy_amount='3')
```



SLDapp.**show_wait_frame**(*message*)

Display a wait message frame with string *message*.

Use in case you have a thread doing work which will take time. Call this right *before* you start your new thread. The user will not be able to remove this frame; it needs to be programmatically removed by calling *SLDapp.hide_wait_frame()*

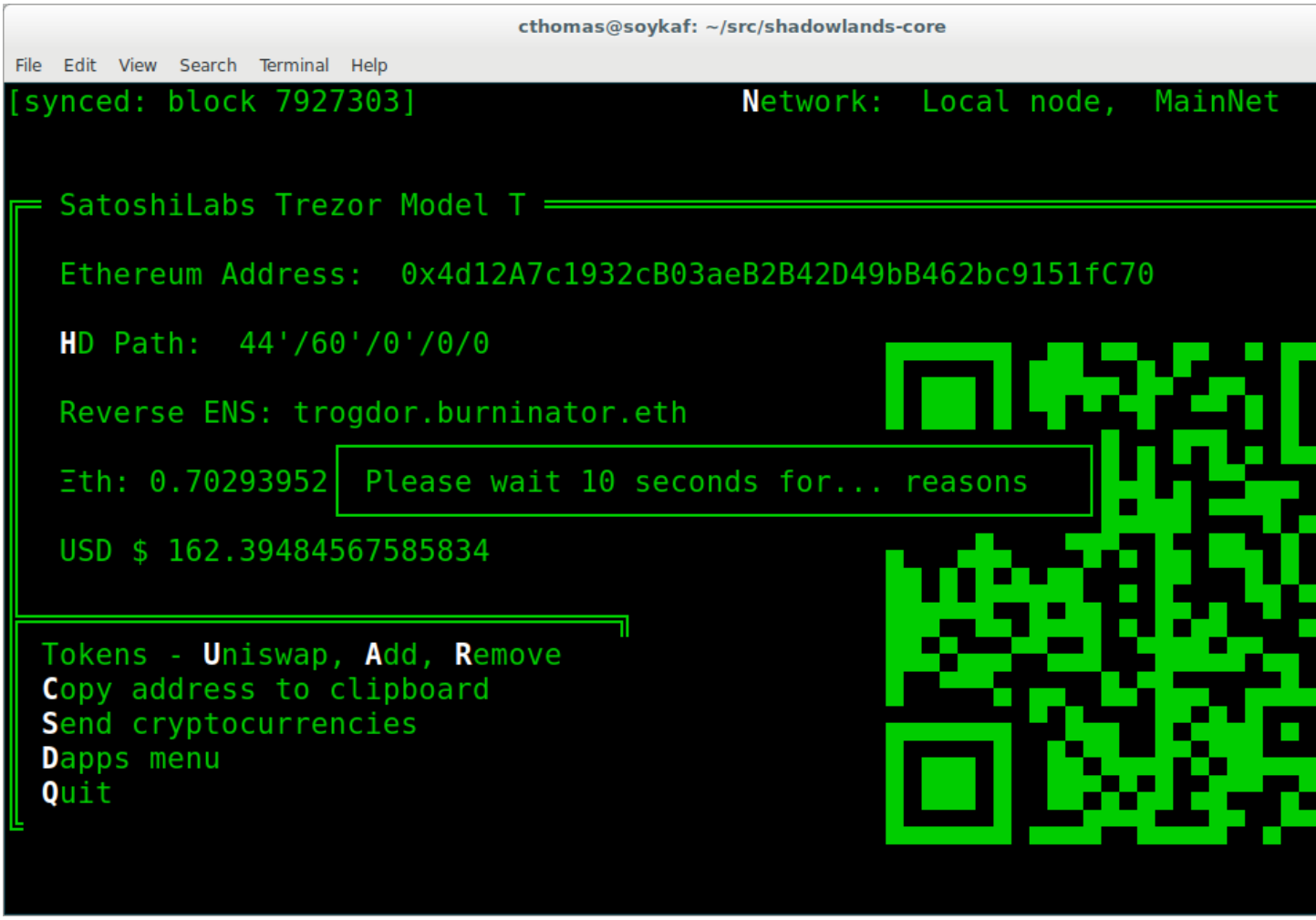
Listing 4: Example

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame
```

(continues on next page)

(continued from previous page)

```
    import threading
    from time import sleep

    class Dapp(SLDapp):
        def initialize(self):
            self.show_wait_frame("Please wait 10 seconds for... reasons")
            threading.Thread(target=self._my_thread).start()

        def _my_thread(self):
            sleep(10)
            self.hide_wait_frame()
```



SLDapp.**hide_wait_frame**()
: Remove the wait message frame. If it is not currently displayed, this method is a no-op.

  This should be called inside your new thread, as the last thing it does.

SLDapp.**save_config_property**(*property_key*, *value*)
: Save a serializable object to the persistent data store.

SLDapp.**load_config_property**(*property_key*, *value*)
: Load a serializable object from the persistent data store.

CHAPTER 3

SLFrame

class SLFrame

3.1 Abstract

SLFrame provides a 'window' for interacting with users in your *SLDapp*. Create a subclass of *SLFrame* and then add *Widgets* to it in the `initialize`() method.

Listing 1: Example

```
from shadowlands.sl_dapp import SLDapp
from shadowlands.sl_frame import SLFrame

class Dapp(SLDapp):
    def initialize(self):
        self.add_sl_frame(
            MyFrame(self, 5, 20)
        )

class MyFrame(SLFrame):
    def initialize(self):
        self.add_button(self.close,"Push Me")
```

3.2 Constructor

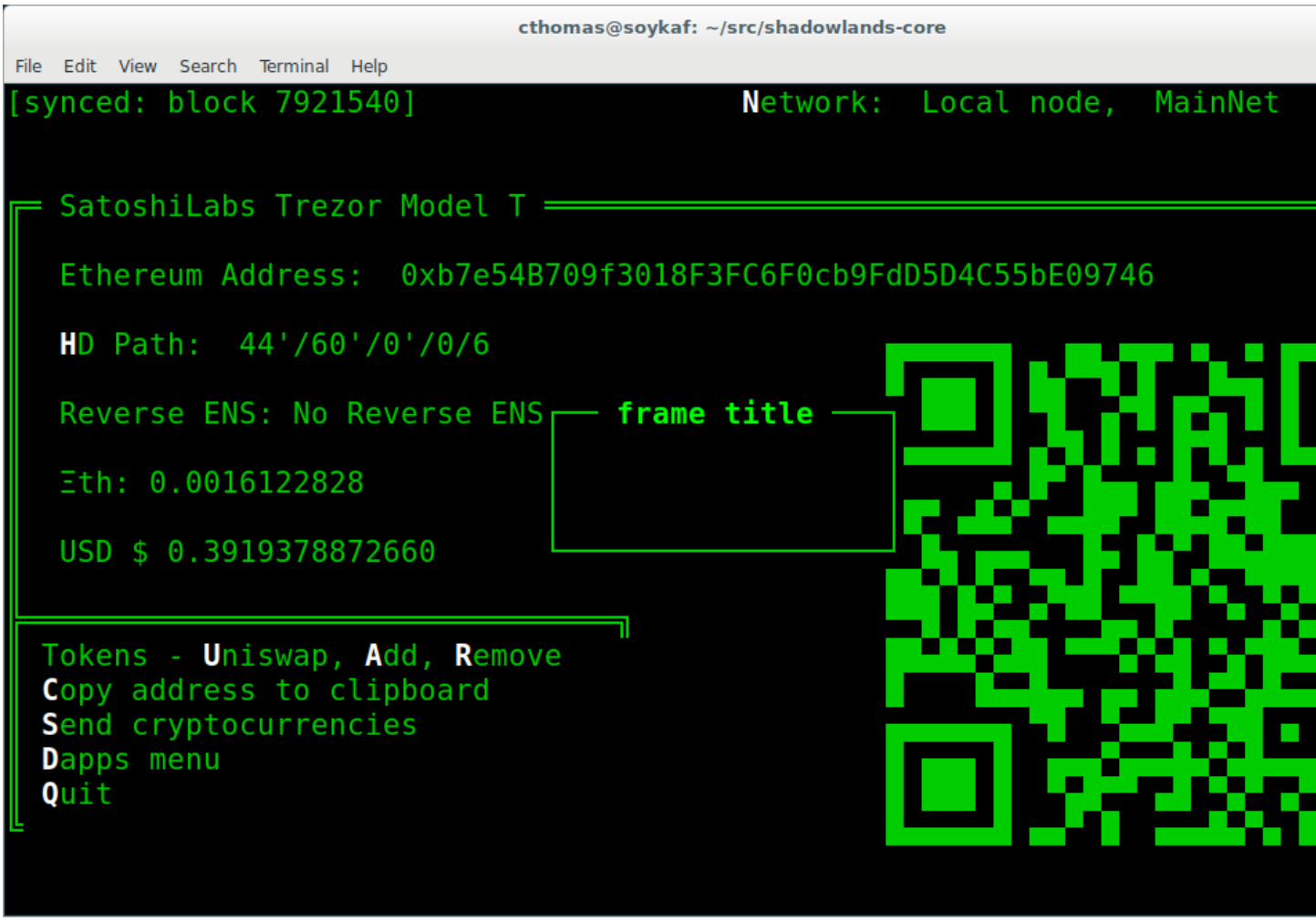
SLFrame(*dapp*, *height*, *width*, *title=None*)

SLFrame is not meant to be directly instantiated. Subclass it and instantiate the subclass.

The constructor's first argument should be an instance of *SLDapp*. Integer *height* and *width* parameters are also required. An optional *title* string will be displayed at the top of the frame.

Listing 2: Example

```
MyFrame(self.dapp, 5, 20, title="frame title")
```



3.3 Properties

`SLFrame.`**`dapp`**
: The instance of *`SLDapp`* which the *`SLFrame`* belongs to.

3.4 Methods

`SLFrame.`**`initialize`**`()`
: An abstract callback that you must implement. It will fire upon the initialization of the object. Do your setup here and add widgets.

`SLFrame.`**`close`**`()`

Call to close the current frame. In your dapp, be sure to open a new frame or dialog before calling `close()` on the current one.

3.5 Widgets

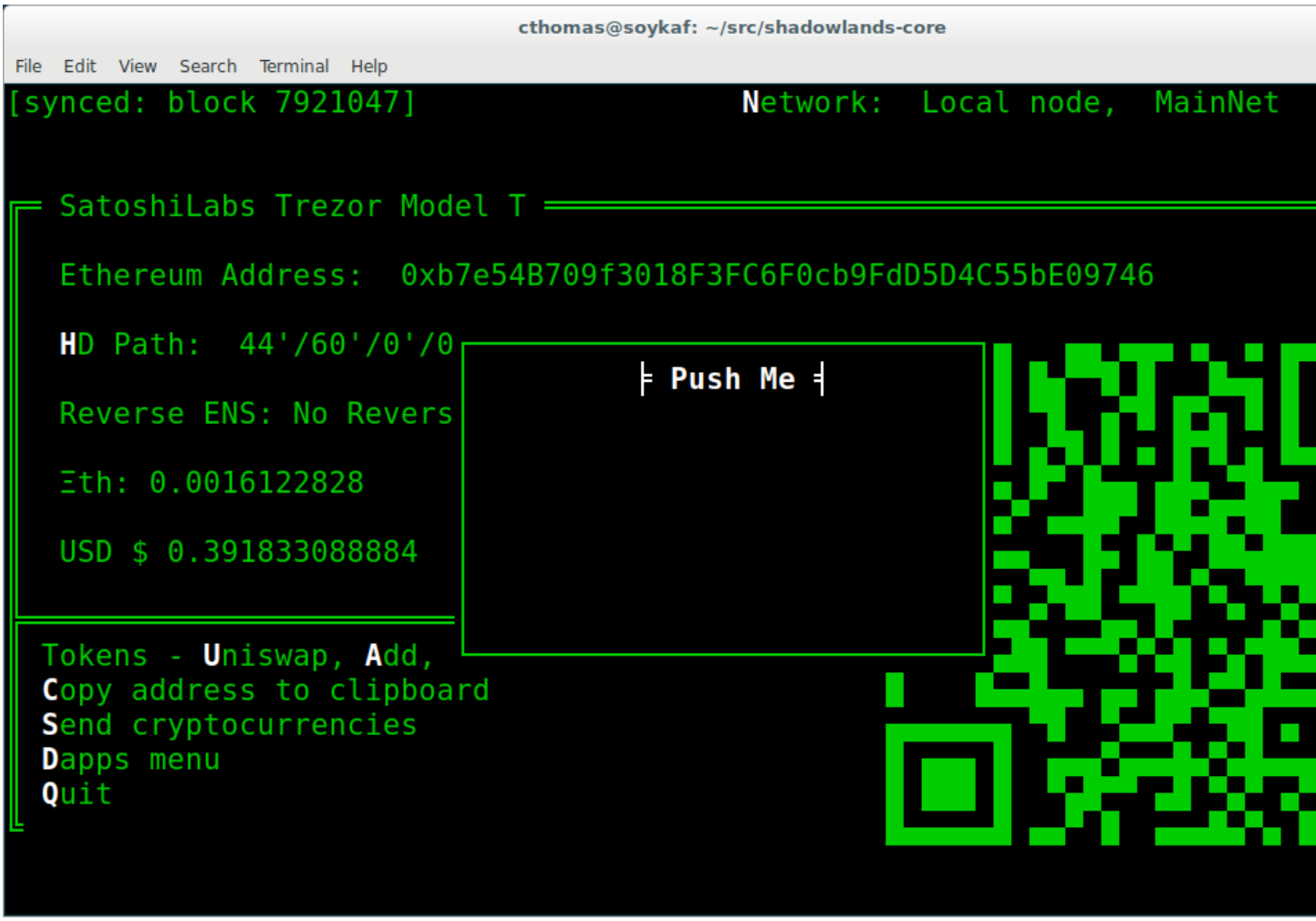
- *SLFrame.add_button()*
- *SLFrame.add_button_row()*
- *SLFrame.add_checkbox()*
- *SLFrame.add_qrcode()*
- SLFrame.add_textbox()
- *SLFrame.add_divider()*
- *SLFrame.add_radiobuttons()*
- *SLFrame.add_listbox()*
- *SLFrame.add_label()*
- *SLFrame.add_label_row()*
- *SLFrame.add_label_with_button()*
- *SLFrame.add_file_browser()*

SLFrame.**add_button**(*fn, text, layout=[100], layout_index=0, add_divider=True*)

Add a single button to your SLFrame. `fn` is a function to run (lambdas are useful for this) when the button is pressed. You can place a string within the button by setting `text`. The optional `layout` and `layout_index` variables follow the `asciimatics` widget layout rules (see AsciimaticsLayout docs for details)

Listing 3: Example

```
class MyFrame(SLFrame):
    def initialize(self):
        self.add_button(self.close,"Push Me")
```



`SLFrame`.**`add_button_row`**(*buttons, layout=[1, 1, 1, 1], add_divider=True*)

A row of buttons. The argument `buttons` is an array of `(string, function, index)` tuples. `layout` is an AsciimaticsLayout array, which defines the indices available for the buttons.

Listing 4: Example

```
class MyFrame(SLFrame):
    def initialize(self):
        my_buttons = [
            ("Say hello", self.say_hi, 0),
            ("Say bye", self.say_bye, 2),
            ("Close", self.close, 3)
        ]
        self.add_button_row(my_buttons)
```

`SLFrame`.**`add_checkbox`**(*text*, *on_change=None*, *default=False*, ***kwargs*)

Add a checkbox for boolean input. A string variable `text` will appear alongside the checkbox. You can supply a function to `on_change` which will be executed when the checkbox changes state. The function returns a method which you can call later, to retrieve the value in the checkbox.

Listing 5: Example

```
    class MyFrame(SLFrame):
        def initialize(self):
            self.boxvalue = self.add_checkbox("sometext", on_change=self.
↪show_value, default = True)
```

(continues on next page)

(continued from previous page)

```
            self.add_divider()
            self.add_button(self.close, "close")

        def show_value(self):
            self.dapp.add_message_dialog(str(
                self.boxvalue()
            ))
```

SLFrame.**add_qrcode**(*data*)

Displays a QRCode from the data given.

Listing 6: Example

```
class Dapp(SLDapp):
    def initialize(self):
        myframe = MyFrame(self, 20, 40)
        self.add_sl_frame(myframe)

class MyFrame(SLFrame):
    def initialize(self):
```

(continues on next page)

(continued from previous page)

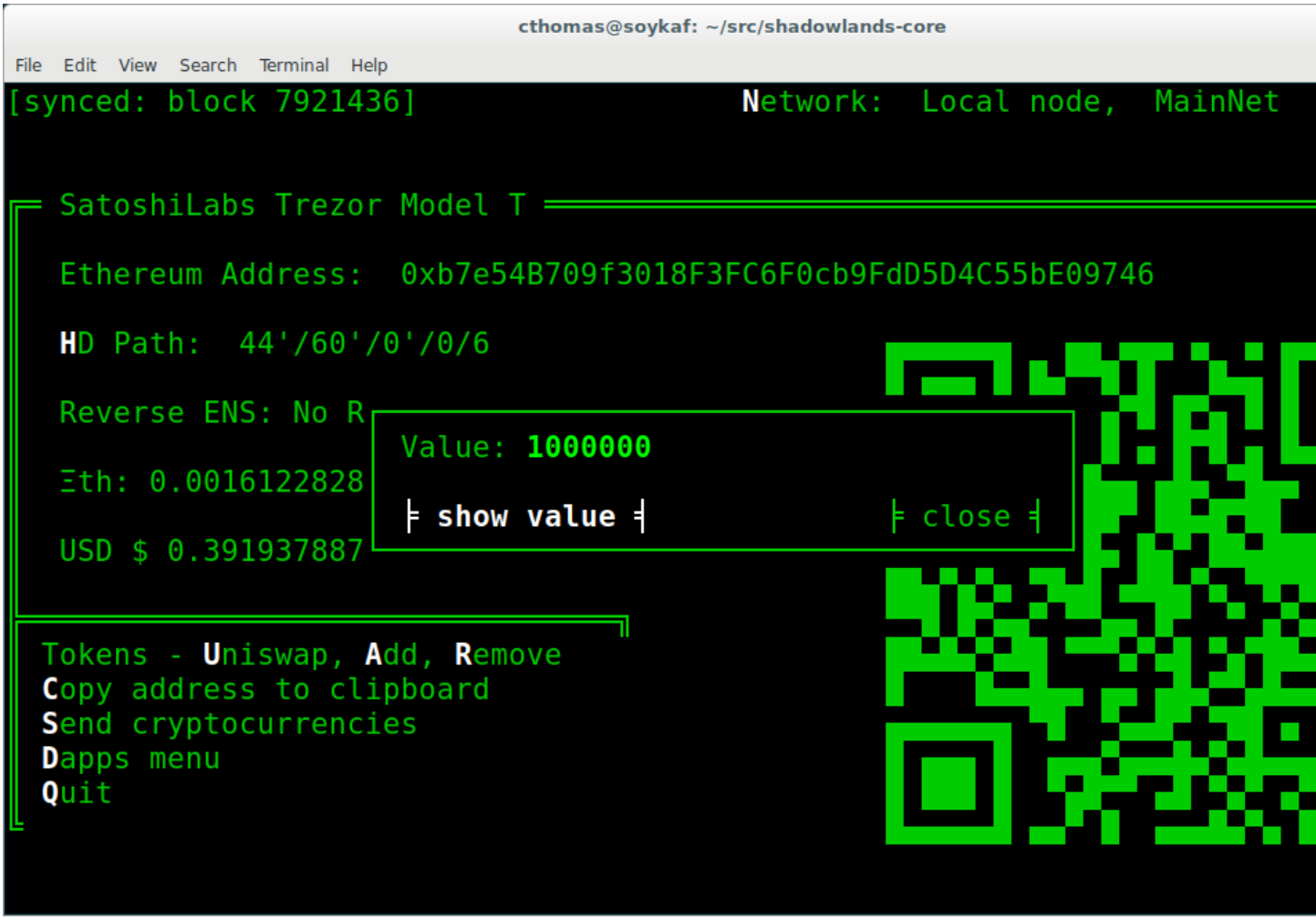
```
        self.add_qrcode(self.dapp.node.credstick.address)
        self.add_button(self.close, "close")
```

SLFrame.add_textbox(label_text, default_value=None, add_divider=True, on_change=None, **kwa

Diplays a textbox for input. `on_change` takes a function that is run when the textbox changes value.

Listing 7: Example

```
class MyFrame(SLFrame):
    def initialize(self):
        self.textbox_value = self.add_textbox("Value:", default_value="1000000")
        self.add_button_row([
            ("show value", self.show_value,0),
            ("close", self.close, 3)
        ])

    def show_value(self):
        self.dapp.add_message_dialog(str(
            self.textbox_value()
        ))
```



SLFrame.**add_divider**(*draw_line=False*, ***kwargs*)

Add a horizontal spacer. `draw_line` will cause a line to be drawn across the space.

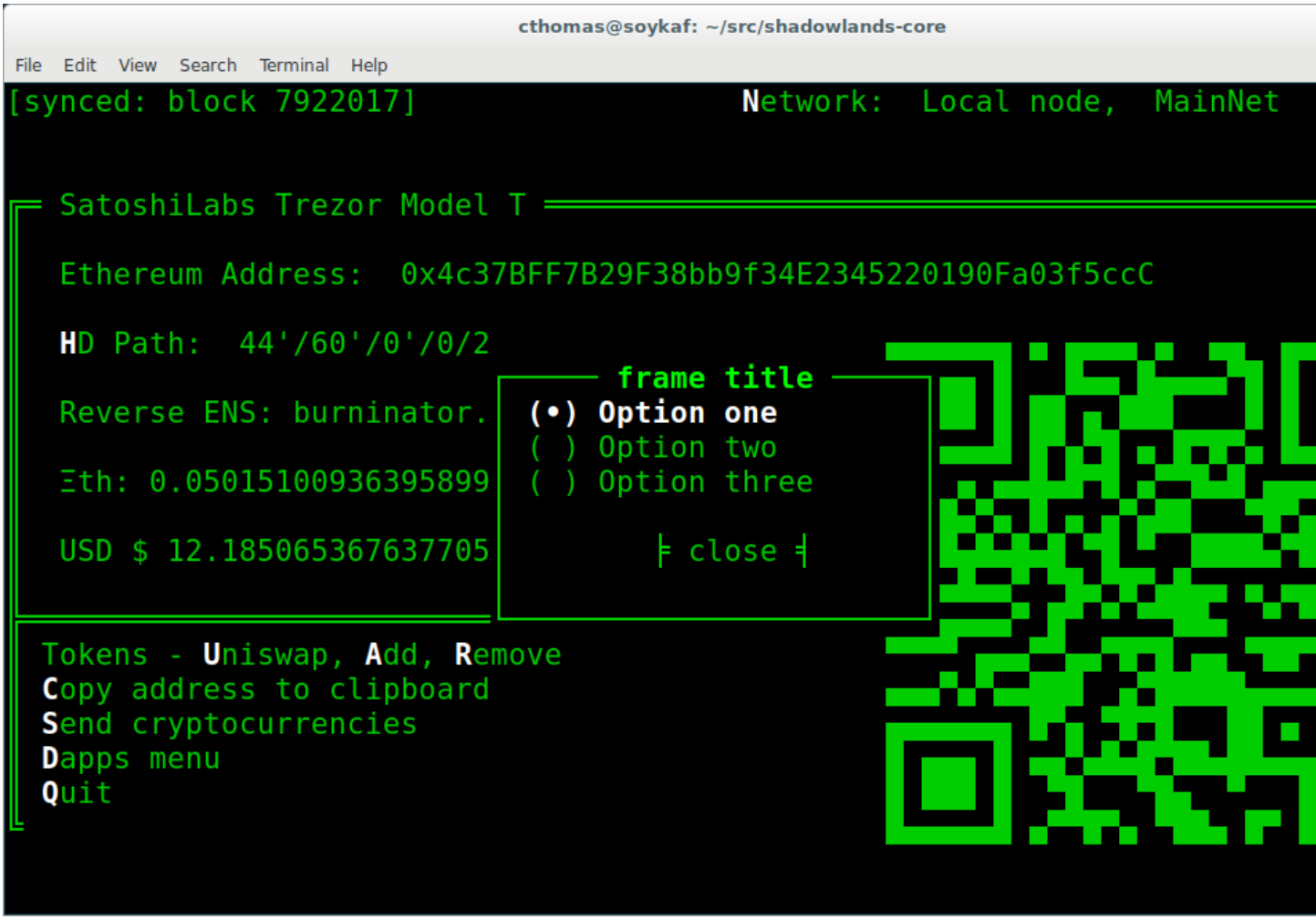
SLFrame.**add_radiobuttons**(*options, default_value=None, layout=[100], layout_index=0, add_divider=True, on_change=None **kwargs*)

Radiobuttons widget. Returns a function which, when executed, gives the value chosen. `options` is an array of

tuples, filled with (label, value). `layout` follows the layout rules described in AsciimaticsLayout. You can provide an optional `on_change` function.

Listing 8: Example

```
class MyFrame(SLFrame):
    def initialize(self):
        options = [
            ("Option one", 1),
            ("Option two", 2),
            ("Option three", 3)
        ]
        self.options_value = self.add_radiobuttons(
            options,
            default_value = 2,
            on_change=self.useful_fn
        )
        self.add_button(self.close, "close")

    def useful_fn(self):
        self.dapp.add_message_dialog(self.options_value())
```

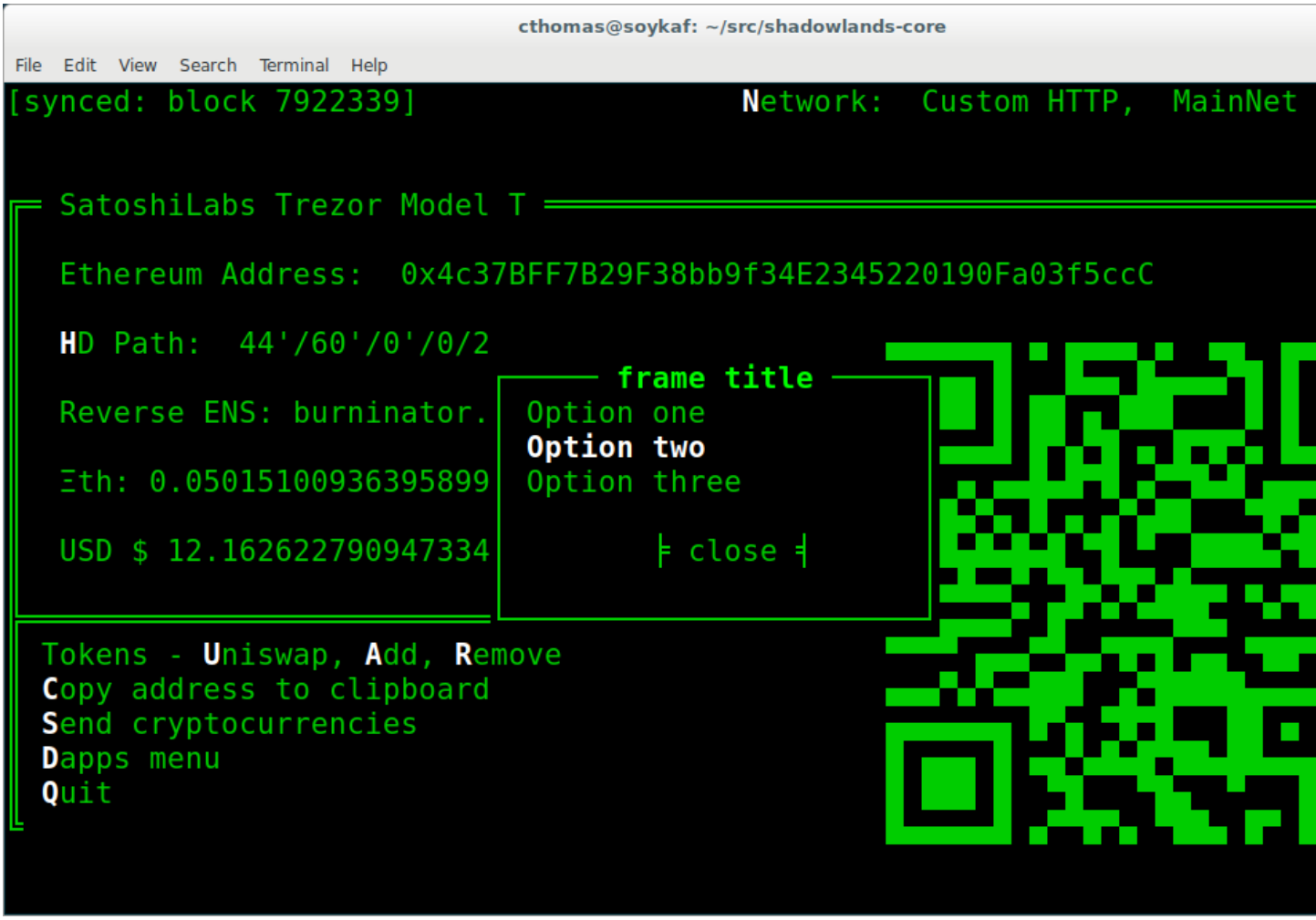


SLFrame.**add_listbox**(*height, options, default_value=None, on_select=None, layout=[100], layout_index=0, **kwargs*)

Returns a function which, when executed, gives the value chosen. `height` is the number of lines high the listbox reaches. If the length of the `options` array exceeds `height`, the user will be able to scroll to see all the options. `options` is an array of tuples, filled with (label, value). `layout` follows the layout rules described in AsciimaticsLayout. You can provide an optional `on_change` function.

Listing 9: Example

```
class Dapp(SLDapp):
    def initialize(self):
        myframe = MyFrame(self, 8, 25, title="frame title")
        self.add_sl_frame(myframe)

class MyFrame(SLFrame):
    def initialize(self):
        options = [
            ("Option one", 1),
            ("Option two", 2),
            ("Option three", 3)
        ]
        self.options_value = self.add_listbox(
            options,
            default_value = 2,
            on_change=self.useful_fn
        )
        self.add_button(self.close, "close")

    def useful_fn(self):
        self.dapp.add_message_dialog(self.options_value())
```

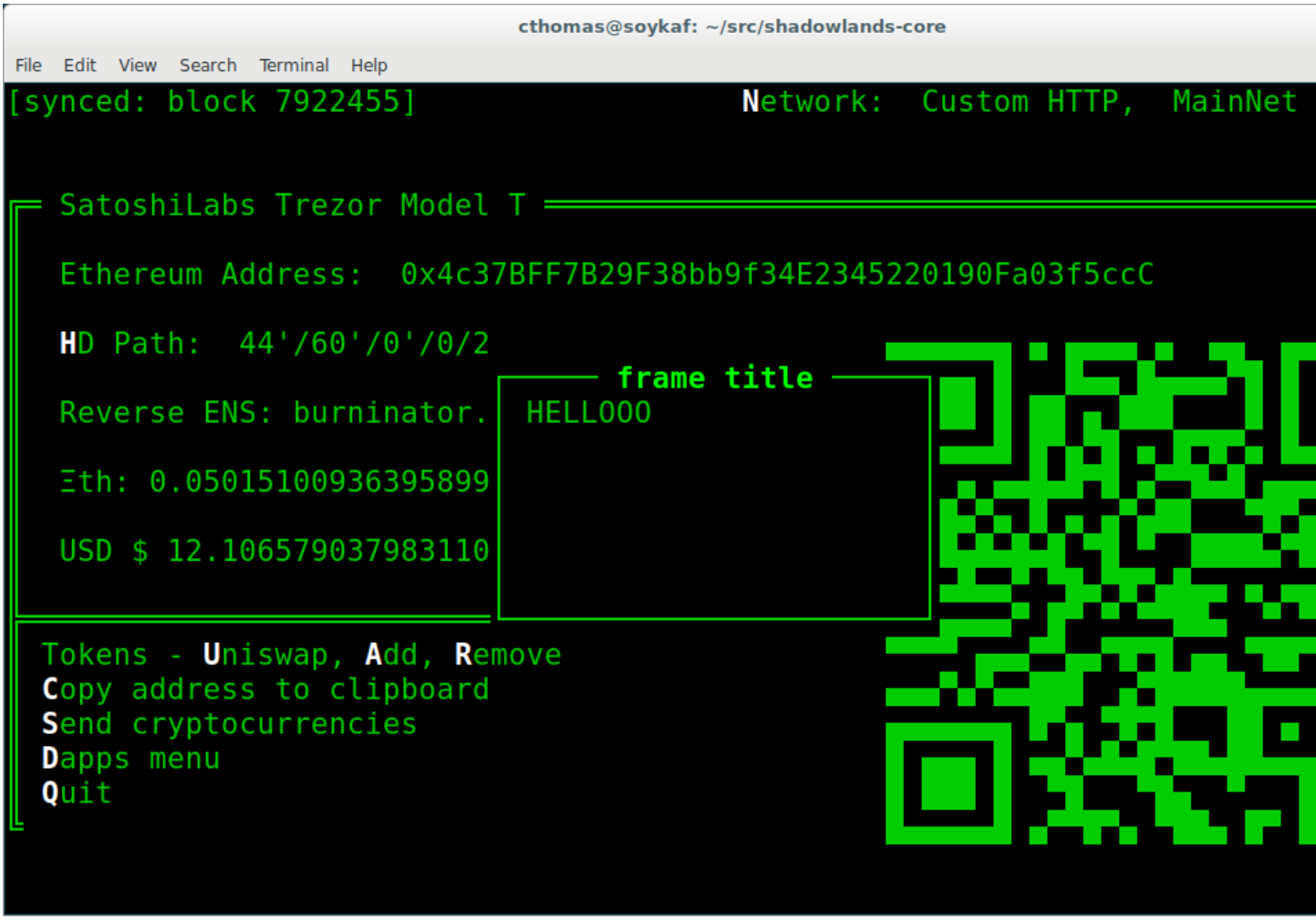


`SLFrame.`**`add_label`**(*label_text, layout=[100], layout_index=0, add_divider=True*)

Display the string `label_text`.‘‘layout‘‘ follows the layout rules described in AsciimaticsLayout.

Listing 10: Example

```
class Dapp(SLDapp):
    def initialize(self):
        myframe = MyFrame(self, 8, 60, title="frame title")
        self.add_sl_frame(myframe)

class MyFrame(SLFrame):
    def initialize(self):
        self.add_label("HELLOOO")
```



SLFrame.**add_label_row**(*self, labels, layout=[1, 1, 1, 1], add_divider=True*)

Add multiple labels. `labels` is an array of tuples of format (string, index) where index is the layout index. `layout` follows the layout rules described in AsciimaticsLayout.

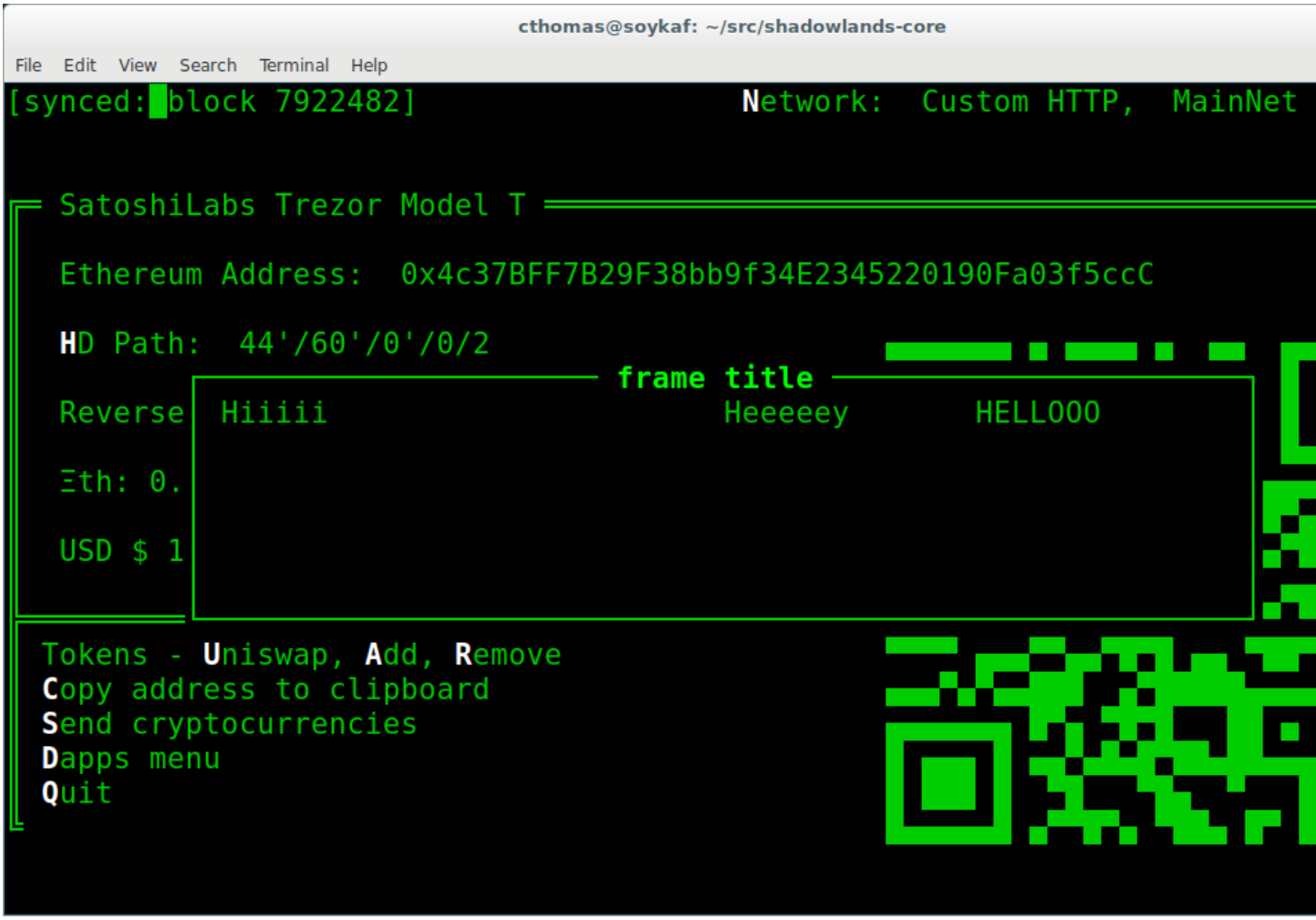
Listing 11: Example

```
class MyFrame(SLFrame):
    def initialize(self):
        labels = [
```

(continues on next page)

(continued from previous page)

```
            ("Hiiiii", 0),
            ("Heeeeey", 2),
            ("HELLOOO", 3)
        ]
        self.add_label_row(labels)
```



SLFrame.**add_label_with_button**(*label_text, button_text, button_fn, add_divider=True, layout=[70, 30]*)

A label on the left and button on the right. `button_fn` will be executed upon button press. `layout` follows the layout rules described in AsciimaticsLayout.

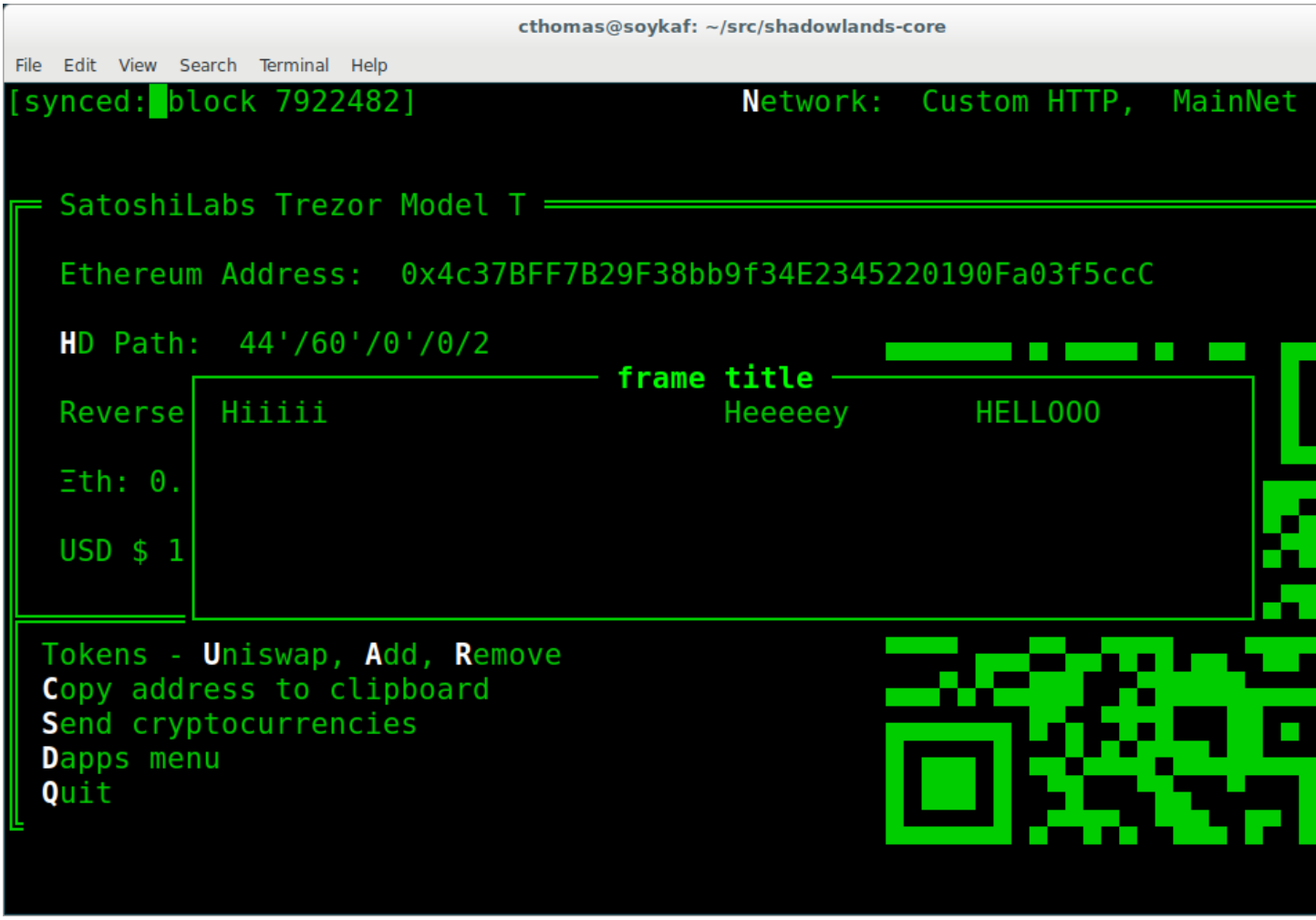
Listing 12: Example

```
class Dapp(SLDapp):
    def initialize(self):
        myframe = MyFrame(self, 20, 70, title="frame title")
        self.add_sl_frame(myframe)

class MyFrame(SLFrame):
    def initialize(self):
```

(continues on next page)

(continued from previous page)

```
        labels = [
            ("Hiiiii", 0),
            ("Heeeeey", 2),
            ("HELLOOO", 3)
        ]
        self.add_label_row(labels)
```



`SLFrame`.**`add_file_browser`**(*path='/'*, *height=15*, *on_change_fn=None*)

Creates a file browser to select directories and files.

Returns a function that returns the selected filepath.

`path` is the default filepath to start at. `height` is an integer number of how many files to display. You can scroll through the rest.

`on_change_fn` will fire whenever the filepath is changed.

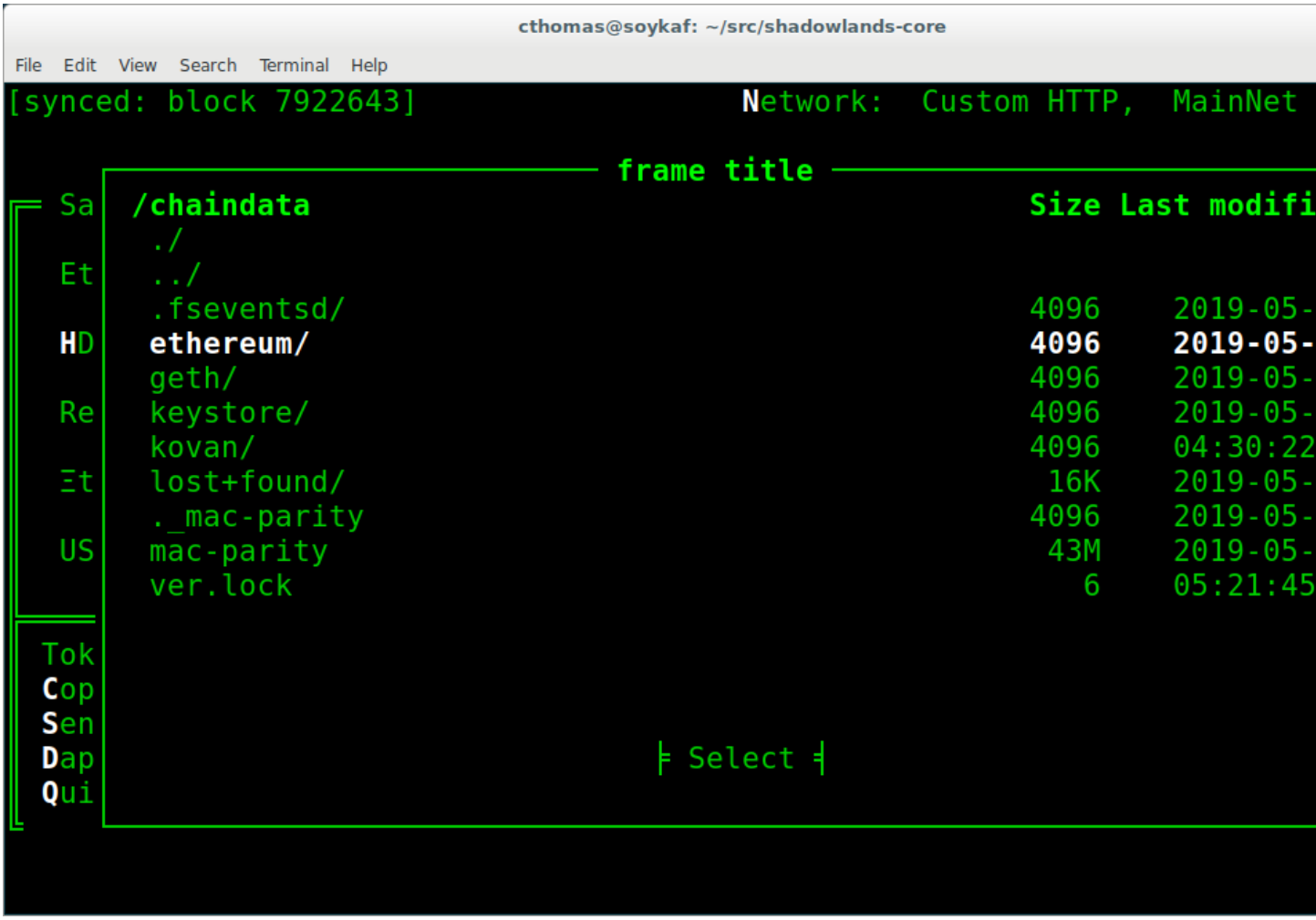
Listing 13: Example

```
class Dapp(SLDapp):
    def initialize(self):
```

(continues on next page)

(continued from previous page)

```
        myframe = MyFrame(self, 20, 70, title="frame title")
        self.add_sl_frame(myframe)

class MyFrame(SLFrame):
    def initialize(self):
        self.file_value = self.add_file_browser(path='/chaindata')
        self.add_button(self.useful_fn, "Select")

    def useful_fn(self):
        self.dapp.add_message_dialog(self.file_value())
```




cthomas@soykaf: ~/src/shadowlands-core
File Edit View Search Terminal Help
[synced: block 7922645] Network: Custom HTTP, MainNet
frame title
/chaindata
Size Last modifi
./
../
.fseventsd/ 4096 2019-05-
ethereum/ 4096 2019-05-
geth/ 4096 2019-05-
keystore/ 4096 2019-05-
kovan/ 4096 04:30:22
lost+found/ 16K 2019-05-
._mac-parity 4096 2019-05-
mac-parity 43M 2019-05-
ver.lock 6 05:21:45
/chaindata/ethereum
Ok
Select

CHAPTER 4

SLContract

class SLContract

4.1 Abstract

This is a helper and wrapper around the w3 contract class.

4.2 Constructor

SLContract (*node*, *address=None*, *provided_abi=None*)

Initialize an SLContract by passing it the *Node* object.

The SLContract must be fed an address and the abi, but you have options for how to do this:

You can pass the string keyword arg `address` to the constuctor, or optionally you can subclass SLContract and assign the address to constants such as `MAINNET`, `ROPSTEN`, `KOVAN`, and other etherum network names.

You can pass the the string keyword arg `provided_abi` to the constructor, or optionally you can subclass SLContract and assign the JSON abi string to the `ABI` constant.

Listing 1: Example 1

```
abi = '''[{"constant":false,"inputs":[{"name":"owner_", "type":"address"}]
,"name":"setOwner","outputs":[], "payable":false,"stateMutability":"nonpayable",
"type":"function"}'''
contract_address='0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359'

''' then, get the node and...'''
contract = SLContract(node, address=contract_address, provided_abi=abi)
```

Listing 2: Example 2

```
from shadowlands.sl_contract import SLContract

class SaiPip(SLContract):

    KOVAN='0xa944bd4b25c9f186a846fd5668941aa3d3b8425f'
    ABI='[{"constant":false,"inputs":[{"name":"owner_","type":"address"}],
    "name":"setOwner","outputs":[],"payable":false,
    "stateMutability":"nonpayable","type":"function"}'

''' then, later, import SaiPip, get the node and...'''
sai_pip = SaiPip(node)
```

4.3 Properties

SLContract.**w3**

The w3 object as provided by the web3.py library

SLContract.**node**

The *Node* object

SLContract.**functions**

This is an autogenerated object that contains the contract function generators as described in the ABI the SLContract was initialized with.

Passed through from the web3.py Contract object.

Listing 3: Example 1

```
abi = '''[{"constant":false,"inputs":[{"name":"owner_", "type":"address"}]
,"name":"setOwner","outputs":[], "payable":false,"stateMutability":"nonpayable",
"type":"function"}'''
contract_address='0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359'

''' then, get the node and...'''
contract = SLContract(node, address=contract_address, provided_abi=abi)
set_owner_fn = contract.functions.setOwner(node.credstick.address)
```

SLContract.**address**

The string address of the SLContract.

4.4 Methods

SLContract.**sha3**(*data*)

The sha3 function as used in ethereum contracts.

SLContract.**bytes32**(*an_int*)

Turns an int into bytes32 value for use in an ethereum contract.

SLContract.**to_sol_addr**(*address*)

Removes the '0x' and hex decodes an address for use in an ethereum contract.

`SLContract.`**`toWei`**(*amount*, *from_denomination*)

Equivalent of `amount * 10 ** 18`

`SLContract.`**`fromWei`**(*amount*, *to_denomination*)

Equivalent of `amount / 10 ** 18`

CHAPTER 5

Erc20

class Erc20

5.1 Abstract

A subclass of *SLContract* specialized to handle contracts which conform to the Erc20 standard.

5.2 Integer vs. Decimal values

Integer values are the base denomination of a token, similar to Wei. Decimal values are the human readable denomination of a token, similar to Ether.

You can convert using *Erc20.convert_to_decimal()* or *Erc20.convert_to_integer()*

5.3 Constructor

Erc20(*node*, *address=None*)

Initialize an Erc20 by passing it the *Node* object.

The Erc20 must be fed an address, but you have options for how to do this:

Usually you will pass the string keyword arg `address` to the constuctor, or optionally you can subclass Erc20 and assign the address to constants such as `MAINNET`, `ROPSTEN`, `KOVAN`, and other etherum network names.

Listing 1: Example 1

```
contract_address='0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359'
''' then, get the node and...'''
token = Erc20(node, address=contract_address)
```

5.4 Properties

`Erc20.`**`decimal_balance`**

Returns the current address holder's token balance, normalized in a human-readable Decimal. This is like getting the balance in Ether instead of in Wei.

5.5 Methods

`Erc20.`**`my_balance`**`()`

Returns the current address holder's token balance, as an integer. This is like getting the balance in Wei instead of in Ether.

`Erc20.`**`totalSupply`**`()`

Returns integer value of the total supply.

`Erc20.`**`allowance`**`(`*owner*, *proxy*`)`

Returns integer value of tokens that address `proxy` is allowed to access on behalf of address `owner`.

`Erc20.`**`self_allowance`**`(`*proxy*`)`

Returns integer value of tokens that address `proxy` is allowed to access on behalf of the user's current address.

`Erc20.`**`symbol`**`()`

Returns string symbol of token.

`Erc20.`**`decimals`**`()`

Returns integer number of decimals supported by token.

`Erc20.`**`balanceOf`**`(`*target*`)`

Returns integer value of tokens owned by address `target`.

`Erc20.`**`my_balance`**`()`

Returns integer value of tokens owned by user's current address.

`Erc20.`**`convert_to_decimal`**`(`*amount*`)`

Returns decimal token value of integer `amount`

`Erc20.`**`convert_to_integer`**`(`*amount*`)`

Returns integer token value of decimal `amount`

`Erc20.`**`my_balance_str`**`(`*length=18*`)`

Returns string interpretation of user's decimal token value, truncated to `length` characters.

5.6 Tx function generators

`Erc20.`**`approve`**`(`*proxy_address*, *value*`)`

Returns Tx function that will approve `proxy_address` to spend integer `value` amount of tokens on behalf of the current user address.

`Erc20.`**`approve_unlimited`**`(`*proxy_address*`)`

Returns Tx function that will approve `proxy_address` to spend an unlimited amount of tokens on behalf of the current user address.

`Erc20.`**`transfer`**(*target*, *value*)

Returns Tx function that will transfer integer `value` amount of tokens to address `target`

CHAPTER 6

Node

class Node

The *Node* class provides ethereum networking functions and other useful properties.

6.1 Obtaining the Node object

You do not need to instantiate the Node object. You can get it by calling *SLDapp.node*, *SLContract.node* or `Erc20.node`.

6.2 Properties

`Node.`**ens**

Returns the `Web3.ens` object. Useful for calling `ens.resolve(name)` to perform a lookup or `ens.name(address)` to perform reverse lookup.

`Node.`**best_block**

Returns the highest known block.

`Node.`**blocks_behind**

Returns the number of blocks behind, or None if synced.

`Node.`**eth_price**

Returns the current eth price in USD, as listed in the Maker contract oracle used by the CDP/DAI system.

`Node.`**network_name**

Returns the uppercase network name string. Returns a string value of the network ID if not recognized.

`Node.`**eth_balance**

Returns the decimal ether value of the user's current address.

Node.**ens_domain**

Return reverse lookup ENS domain of the user's current address, or None.

6.3 Low Level Methods

These are low level methods that do not generate any UI.

Node.**push**(*contract_function*, *gas_price*, *gas_limit=None*, *value=0*, *nonce=None*)

Pushes a Tx function directly to the credstick for signing and publishing on the network.

`value` is the integer-value of wei to send with Tx. `gas_price` is an integer wei value. You need to set `gas_limit` on this method. If `nonce` is not set, it will be automatically calculated for the next valid nonce on the current user address.

Node.**send_ether**(*destination*, *amount*, *gas_price*, *nonce=None*)

Requests an ether send for credstick signing and publishing on the network.

`destination` is a string ethereum address. `amount` is the ether value in decimal format. `gas_price` is an integer wei value. If `nonce` is not set, it will be automatically calculated for the next valid nonce on the current user address.

Node.**send_erc20**(*token*, *destination*, *amount*, *gas_price*, *nonce=None*)

Requests an Erc20 send for signing and publishing on the network.

`token` is an *Erc20* or *SLContract* object which has an Erc20 compatible `transfer()` function.

`destination` is an address. `amount` is a human-friendly decimal amount. `gas_price` is an integer wei value. If `nonce` is not set, it will be automatically calculated for the next valid nonce on the current user address.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

T

W