LESSON MC3 – Synchronize MC

University of West Attica

Department of Electrical and Electronics Engineering

Ioannis Christidis

Christoforos Kachris

Support by Ethereum Foundation ESP

What will we accomplish!

In this lesson we will continue building our `ICR` project.

Now that we finished the account creation, let's go to the next step which is finding the `ICR` and the `Car` that our microcontroller is registered to. This essentially means that we need to find the `ICR address` that has the `Car` with the microcontroller and the `Car ID`.

We discussed before that the microcontroller will create the account and then the owner will send its address to `IIoTC` to register it to an `ICR` contract.

What synchronization needs to do:

It needs to check if the `ICR contract` and `Car` is already known. This means that it will check if the `Car ID` and the `ICR contract address` are already saved in the `.env`

- If they are, it will load them
- If not, it will check in the `ICR Factory` if its address is used in an `ICR` contract with the `getMcToIcr` function that we build in the last contract changes.
  - If it is, it will search through every car in the `ICR` until it finds the car it is registered to and save it along with the `ICR address` to the `.env`. (This is a safe measure in case it loses the data for any reason, and it should not happen)
  - If not, it will start a `websocket connection`, subscribing to the `CarRegisteredToICR` event and wait until one is emitted with `ICR address, Car ID` and its own `address`. It will then save the `ICR` address and `Car ID` to the `.env`.

Creating `icr_synchronizer.py`

We will need to do multiple calls and subscribe to an event so we will use `AsyncWeb3` and `WebSocket` for the synchronization.

Also, here we will need the `ICR Factory address` and `ABI` as well as the `ICR ABI`.

```
import os
from web3 import AsyncWeb3, WebSocketProvider
from eth_abi.abi import decode
from dotenv import load_dotenv, set_key
import load_json


# Load environment variables from .env file
load_dotenv()


# WebSocket RPC URL
WSS_URL = os.getenv("WSS_URL")
# ICR Factory Address
FACTORY_ADDRESS = os.getenv("FACTORY_ADDRESS")
# ICR Factory ABI
factory_abi = load_json.load_contract_abi("factory_abi.json")
# ICR ABI
icr_abi = load_json.load_contract_abi("icr_abi.json")
```

Creating icr_synchronizer.py

Next let's make our entry point function for this one and start the `WebSocket` connection.

```python
async def synchronize_with_icr():
    print("Opening Async Web3")
    try:
        async with AsyncWeb3(WebSocketProvider(WSS_URL)) as w3:
            await fetch_icr_info_or_subscribe(w3)
    except Exception as e:
        print(f"Error during WebSocket connection: {e}")
```

You can see the `fetch_icr_info_or_subscribe` function which we will build at the end.

We will now create the function that allows us to subscribe to `CarRegisteredToICR` `events`. We will pass the `WebSocket` connection as a parameter here as well as the `address` of the microcontroller. We are also going to write in the `.env` file in this function, so we will also pass it as a parameter. We will add the logic inside a `try/except` block.

```python
async def subscribe_to_car_registered_event(w3, mc_address, env_file=".env"):
    try:
        pass
    except Exception as e:
        print(f"Error during subscription: {e}")
```

Subscribing to car registered to ICR event

The `event` we are going to subscribe to is this one:

```
event CarRegisteredToICR(address indexed _icrAddress, uint256 indexed _carId, address indexed _mc);
```

As you can see it logs the `address of the ICR contract` and the Car `ID` that we want to store, and it also `emits the address of the microcontroller` which we will compare with the `address in our microcontroller` to ensure that it is the correct `event`.

We will do a similar process to the one we did at the example in the previous lesson.

```
        car_registered_event_topic = w3.keccak(text="CarRegisteredToICR(address,uint256,address)") # topic we want to subscribe to

        filter_params = {
            "address": FACTORY_ADDRESS,
            "topics": [car_registered_event_topic],
        } # Parameters of the filter

        subscription_id = await w3.eth.subscribe("logs", filter_params) # create subscription
        print(f"Subscribing to CarRegisteredToICR event for ICRFactory at {subscription_id}")

        async for payload in w3.socket.process_subscriptions(): # get the payload
            result = payload["result"]

            _icr_address = decode(["address"], result["topics"][1])[0] # ICR Address
            _car_id = decode(["uint256"], result["topics"][2])[0] # Car ID
            _mc = decode(["address"], result["topics"][3])[0] # Microcontroller Address
```

Comparing microcontroller addresses

Then we compare the microcontroller `addresses` and if they are the same, we store the `ICR Address` and `Car ID` into the `.env` and we unsubscribe from the event.

```
            if _mc.lower() == mc_address.lower():
                set_key(env_file, "ICR_ADDRESS", _icr_address)
                set_key(env_file, "CAR_ID", str(_car_id))
                print(f"Match found! Stored ICR Address: {_icr_address}, Car ID: {_car_id}")


                # Stop subscribing to CarRegisteredToICR events
                await w3.eth.unsubscribe(subscription_id)
                print("Unsubscribed from CarRegisteredToICR events.")
                return
```

`subscribe_to_car_registered_event` should only be called after we make sure that the microcontroller does not already know the `ICR Address` and the `Car ID`. So, first, we will check if `ICR Address` and `Car ID` exist or not in the `.env` and `return true or false` accordingly.

```
def check_if_icr_info_exist(env_file):
    print("Checking if ICR info exists")
    existing_icr_address = os.getenv("ICR_ADDRESS")
    existing_car_id = os.getenv("CAR_ID")

    if existing_icr_address and existing_car_id:
        print(f"ICR Address and Car ID already exist in .env: {existing_icr_address}, {existing_car_id}")
        return True
    return False
```

Search for ICR and Car ID in the Smart Contract

If the data is not already stored in the `.env` we will try to find them in the `ICR` SC. We can make sure if the microcontroller has already been registered by calling the `getMcToIcr` function from the `ICRfactory` with the microcontroller `address` as a parameter. This will return the `ICR contract address` that our microcontroller is registered to and if it is not registered yet it will return `address(0)`. So, we will create a function that does that.

```
async def search_for_icr_info(w3, env_file, mc_address, factory_address, factory_abi, icr_abi):
    print("Check factory for matching ICR")

    # Interact with the factory contract
    factory = w3.eth.contract(address=w3.to_checksum_address(factory_address), abi=factory_abi)
    icr_address = await factory.functions.getMcToIcr(mc_address).call()


    # Check if the ICR address is not the zero address
    zero_address = "0x0000000000000000000000000000000000000000"
    if icr_address.lower() == zero_address:
        print("Matching ICR Address not found.")
        return False

    # ICR address found
    print(f"ICR Address found: {icr_address}")
    print("Storing address to env")
    set_key(env_file, "ICR_ADDRESS", icr_address) # store the address of ICR
```

Continue the Search

If we find an `ICR address` that our microcontroller is registered to, it will also be registered to a car, and we need to find its ID. To find the car we will search the data of each car in that ICR SC until we find a match. To do this, we will first need the `next Car ID` to use it for a `for` loop through every car.

```
async def search_for_icr_info(w3, env_file, mc_address, factory_address, factory_abi, icr_abi):
    # … Rest of the code
    icr = w3.eth.contract(address=w3.to_checksum_address(icr_address), abi=icr_abi)
    next_car_id = await get_next_car_id(icr)
    if next_car_id == 0:
        return False
    # … More code


async def get_next_car_id(icr):
    print('Retrieving next car ID')
    next_car_id = await icr.functions.getNextCarId().call()
    if next_car_id == 0:
        print("No cars registered in the ICR contract.")
        return 0
    return next_car_id
```

Loop through the cars

Now that we have the `Next Car ID`, we can make a loop to get each of the car data from the blockchain and check if their microcontroller address matches the one that this microcontroller has. After that we will save the Car ID.

```python
async def search_for_icr_info(w3, env_file, mc_address, factory_address, factory_abi, icr_abi):
    # … Rest of the code
    # Find matching car
    found = await find_matching_car(icr, mc_address, next_car_id, env_file)
    if not found:
        print("No matching car found in the ICR contract.")
    return found
    # … More code


async def find_matching_car(icr, mc_address, next_car_id, env_file=".env"):
    print("Check each car to find the match")
    for i in range(next_car_id):
        car = await icr.functions.getCar(i).call()
        mc = car[0]  # First element of the tuple is the microcontroller address
        if mc.lower() == mc_address.lower():
            set_key(env_file, "CAR_ID", str(i))
            print(f"Car found: ID {i} with Microcontroller {mc_address}")
            return True  # Return True when a matching car is found
    return False  # Return False if no matching car is found
```

Finish the script

And finally, now that we have all the functionality we need, we can build the function `fetch_icr_info_or_subscribe`.

```python
async def fetch_icr_info_or_subscribe(w3, env_file=".env"):
    MC_ADDRESS = os.getenv("MC_ADDRESS")
    if check_if_icr_info_exist(env_file): # Check if ICR_ADDRESS and CAR_ID are already set
        print("ICR exists, continueing")
        return  # Exit if ICR_ADDRESS and CAR_ID are already set
    print("ICR info do not exist")
    print("Try to find matching ICR")
    try:
        # Process the ICR contract
        found = await search_for_icr_info( # Check if ICR_ADDRESS and CAR_ID are already stored in an ICR cotnract
            w3,
            env_file,
            MC_ADDRESS,
            FACTORY_ADDRESS,
            factory_abi,
            icr_abi
        )
        if not found:
            print("No matching car found.")
            # Call the function for subscribing to events
            await subscribe_to_car_registered_event(w3, MC_ADDRESS, env_file)
    except Exception as e:
        print(f"Error occurred: {e}")
```

Update `main.py`

Now, we can just call the function `synchronize_with_icr` in the `main` function.

```python
async def main():
    # Load environment variables from .env
    load_dotenv(override=True)
    env_file = ".env"
    # Check if the .env file exists and has PUBLIC_ADDRESS and PRIVATE_KEY
    if not path.exists(env_file) or not os.getenv("MC_ADDRESS") or not os.getenv("MC_PRIVATE_KEY"):
        print("No account found. Creating a new Ethereum account...")
        # Create an address and private key
        create_and_save_account(env_file)
        load_dotenv(override=True)
    else:
        # If account already exists, load it
        MC_ADDRESS = os.getenv("MC_ADDRESS")
        print("Account already exists:")
        print(f"Public Address: {MC_ADDRESS}")

    print("Fetch ICR data from event or already existing")
    # Synchronize with ICR Contract
    await synchronize_with_icr()
    load_dotenv(override=True)
```

Outro

Next let's go over creating the script to unlock the car.