Protocol Examples

Hi Kristian!

⏮️ Monday, you asked me to focus on delivering an Agent that could output the correct protocol example (from your GitHub) depending on a Natural Language input.

It's gonna work 🎉 because the example files in your Github are good material for finetuning

⚠️ **The problem:**

The protocols are full of patterns. However, the order of each part is sometimes different between protocols. The way calls are done differs sometimes (use 1 line in one protocol and 2 lines in another). This lack of standardization (even if marginal) decreases how easy to understand Eulith's product is; also it makes it more challenging to train a model to have standard answers to simple questions.

🦾 **The Solution**:

I recommend going further in the standardization of the code. Creating fixed and universal steps in which content will only change depending on the used protocol.

Here is the link to the GitHub repository with the modified files

ex:

- *#1: importing libraries*
- *#2: Fill constructor*
- *#3: Check wallet's funds' availability*
- *etc…*

💰 **The Benefits:**

1. Improve users' understanding of what is happening thanks to protocol patterns. The idea would be to create an accessible file that explains each step in detail and is accessible to the User. The URL of this page would be given by the AI agent when providing the output code to the user.
2. Having defined steps will help to create an accurate conversational AI Agent. For example, you could have this kind of exchange:

> User: "Hey, what is happening in Step 2"
> AI Agent: "In step 2, we are building the constructor of the *Simple Swap.* I provided you automatically with our *LocalSigner* method for custody, where you have to input your wallet PRIVATE_KEY as a string. be aware that our API supports securer custody methods here: https://docs.eulith.com/v/hgbRx2t48xMLL5xhyh04/client-libraries/python/custody-signers
> Then you should provide your personal EULITH_REFRESH_TOKEN as a string. If

> you lost yours, you can generate a new one here: https://eulithclient.com/home "
> User: "Why is there no Step 7 in the code?"
> AI Agent: "Step 7 opens an atomic bundle used in complex transactions that require a Eulith Proxy Contract. You don't need a Proxy Contract to perform a Simple Swap."

---

2. My questions:

⚠️ You can look at GitHub directly and compare the files with yours. I already made some modifications to standardize primarily in the following docs: *swap.py / simple_transfer.py / short.py / transfert_from_toolkit.py / atomic_transaction.py*

These others were less subject to change because of more different protocols.

General:

▼ I don't understand why setting an `argparse.ArgumentParser(...)` in *simple_transfert.py*

👉 Why is it like that? Is it something necessary or could we define the destination like in the other files?

▼ Make *simple_transfert.py* adaptable to other tokens that ETH

In this file, the amount variable is `amount_in_eth` instead of `amount`.

👉 Can ETH be used as the universal example in all the protocols?

⇒ If yes, I removed all the "_eth" of the variables

⚠️ if the answer is no, many of the following questions won't make sense so stop here.

▼ Variable nomenclature unification:

I marginally modified the variable names to make them match between documents *example:* `tx_hash` *and* `rec` ⇒ `tx_hash` *only.*

▼ *erc20_handling.py*

👉 What is the point of this protocol?
Why are the balances printed instead of being just checked with an if-statement?
Is 0.01 the determined amount of the transaction?

▼ `exit(1)`

Shall the `exit(1)` be generalized after printing that a transaction cannot be done or shall they all be removed?

example in *Swap.py :*

```
#4. Check wallet's funds' availability
    if ew3.eth.get_balance(wallet.address) / 1e18 < amount + 0.03:
        print(f'insufficient balance, deposit at least {amount + 0.002} ETH to perform operation')
        exit(1)
```

▼ *market_data.py*

It would be coud to add a transactional part not have only something descriptive

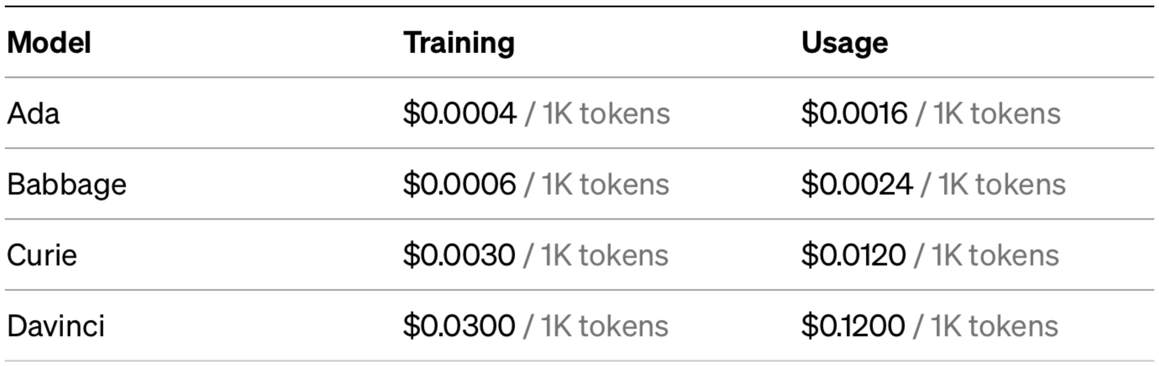
▼ What is the exact difference between *transfert_from_toolkit* and *atomic_transactions* ?

Is it just the number of transactions in the bundle that changes?

Specific to each defined step:

▼ #1. Import Libraries & #2. Fill the Constructor

- Each token processed to train the model costs money and each time the model is called it costs money. Removing those two parts would decrease the cost of creating the model and its usage cost.



Model	**Training**	**Usage**
Ada	$0.0004 / 1K tokens	$0.0016 / 1K tokens
Babbage	$0.0006 / 1K tokens	$0.0024 / 1K tokens
Curie	$0.0030 / 1K tokens	$0.0120 / 1K tokens
Davinci	$0.0300 / 1K tokens	$0.1200 / 1K tokens

Would you prefer me to create a base file with all the imports and possible constructors (custody methods) so that the user can choose which one to use and then use the AI Agent for the rest of the code?

⇒ This question is the reason why I excluded (for now) *kms_signer.py* and *defi_amor.py* from this doc.

▼ #3. Define transaction and its amounts

▼ #4. Check wallet's funds availability

- In *simple_transfert.py,* the amount of funds in the wallet is checked after setting the transaction amount. This enables the check to be performed for gas fees and the transaction amount; on the contrary, in *swap.py,* the check is done before setting the transaction amount (same in *transfer_from_toolkit*). This only enables a check on the gas fees.

  👉 Could we perform the check after defining the transaction amount for all protocols? ⇒ This implies that ETH is the example token (like in the code bellow)

  Also, the if-statement to check the funds' availability is made differently depending on the protocol (number of lines, gas_fees, printed error messages…).

  👉 Can we standardize it like the following?
  - Defined in 1 line
  - Including the transaction amount.
  - Creating a variable for the gas_fee (the good amount would be assigned before the statement and automatically for each type of protocol according to the one you set up in your GitHub)
  - universal error message indicating the amount of the needed deposit

```
if ew3.eth.get_balance(wallet.address) / 1e18 < amount_in_eth + gas_fee:
    print(f'insufficient balance, deposit at least {amount_in_eth + 0.002} ETH to perform operation')
    exit(1)
```

▼ #5. Call and check funds availability in Eulith toolkit

- If-statement (for funds availability check)

  👉 Should we use a if-statement checking the availability of the funds in the toolkit contract or a try/except in every transaction with a toolkit? (see #6)

  So this:

```
if toolkit_sell_token_balance < amount:
        deposit_tx = weth.deposit_eth(amount, {'from': wallet.address, 'gas': 100000})
        deposit_hash = ew3.eth.send_transaction(deposit_tx)
        ew3.eth.wait_for_transaction_receipt(deposit_hash)

        transfer_to_toolkit_contract = weth.transfer_float(
            toolkit_address, amount, {'from': wallet.address, 'gas': 100000})
        transfer_hash = ew3.eth.send_transaction(transfer_to_toolkit_contract)
        ew3.eth.wait_for_transaction_receipt(transfer_hash)
```

  or that:

```
try:
        tx_hash = ew3.eth.send_transaction(atomic_tx)
        receipt = ew3.eth.wait_for_transaction_receipt(tx_hash)
        print(f"\nTransaction hash: {receipt['transactionHash'].hex()}")
    except web3.exceptions.TimeExhausted:
        print("Error: Transaction not found in the chain after 120 seconds. Try the atomic transaction again.")
        exit(1)
```

▼ #6. Open and append transactions to the atomic bundle

- `approve_float(…)`

> 👉 Why is in *transfert_from_toolkit.py* and not in *atomic_transaction.py*? I would have thought that both would need an approval.

```
approval_tx = weth.approve_float(wallet.address, amount)
```

- `try/except` and `print()`

> 👉 Why is there this `try/except` architecture on some protocols *atomic_transaction.py* and not on other? Is it really necessay at this stage to know where is the error comming from with a `print` statement ?

```
try:
        ew3.eth.send_transaction({'from': wallet.address,
                                  'to': t1_wallet_address,
                                  'value': hex(int(t1_send_amount * 1e18))})
        ew3.eth.send_transaction({'from': wallet.address,
                                  'to': t2_wallet_address,
                                  'value': hex(int(t2_send_amount * 1e18))})
        atomic_tx = ew3.v0.commit_atomic_transaction()
except Exception as e:
        print("Error: Failed to execute transactions or commit atomic transaction:", str(e))
        exit(1)
```

▼ #7. Close bundle

▼ #8. Perform transaction

▼ #9. Empty remaining balance

- In *transfer_from_toolkit* the remaining balance in the Proxy Contract Wallet is sent back to the original wallet after the commit (as advised in the doc). It is not the case in *Swap.py.*

> 👉 Would it be ok to add the emptying of the balance to all protocols which involve a proxy contract?

- If yes, do we have to add the `approval_tx = weth.approve_float(wallet.address, amount)` when openning the bundle everywhere? like in *transfer_from_toolkit* ?

```
#6. Open and append transactions to the atomic bundle
    ew3.v0.start_atomic_transaction(wallet.address)
    approval_tx = weth.approve_float(wallet.address, amount)

    ew3.eth.send_transaction(approval_tx)
```

- The *Short.py* case:

👉 Can I use this too on the short protocol replacing the amount by the collateral amount?

```
#10. Empty remaining balance
    withdraw_weth_from_toolkit_tx = weth.transfer_from_float(
        toolkit_address, wallet.address, collateral_amount, {'from': wallet.address, 'gas': 100000})
    tx_hash = ew3.eth.send_transaction(withdraw_weth_from_toolkit_tx)
    print(f'Withdraw tx: {tx_hash.hex()}')
```