AI Agents: broadcast

Introduction

Protocols cover a very important role within the uAgents FrameworkProtocols ↗ define the standards through which agents communication takes place. In this example, we will be showcasing a scenario where three agents, namedalice ,bob , andcharles , use a custom protocol to communicate. In the example, Alice and Bob support the protocol, whereas Charles attempts to send broadcast messages to all agents using the protocol.

Let's get started!

Walk-through

1. First of all, let's create a Python script for this task, and name it:touch broadcast.py
2. We then need to import theAgent
3. ,Bureau
4. ,Context
5. ,Model
6. , andProtocol
7. classes from theuagents
8. library, and thefund_agent_if_low
9. fromuagents.setup
10. . Then, let's create the 3 different agents using the classAgent
11. . Each agent is initialized with a unique name and a seed phrase for wallet recovery. Additionally, if an agent's wallet balance is low, thefund_agent_if_low()
12. function is called to add funds to their wallet:
13. from
14. uagents
15. import
16. Agent
17. ,
18. Bureau
19. ,
20. Context
21. ,
22. Model
23. ,
24. Protocol
25. from
26. uagents
27. .
28. setup
29. import
30. fund_agent_if_low
31. alice
32. =
33. Agent
34. (name
35. =
36. "alice"
37. , seed
38. =
39. "alice recovery phrase"
40.)
41. bob
42. =
43. Agent
44. (name
45. =
46. "bob"
47. , seed
48. =
49. "bob recovery phrase"
50.)
51. charles
52. =

Agent
(name
=
"charles"
, seed
=
"charles recovery phrase"
)
fund_agent_if_low
(alice.wallet.
address
())
fund_agent_if_low
(bob.wallet.
address
())
fund_agent_if_low
(charles.wallet.
address
())
It is optional but useful to include aseed
parameter when creating an agent to set fixedaddresses ↗
. Otherwise, random addresses will be generated every time you run the agent.
Let's then define two message data models to define the type of messages being handled and exchanged by the agents. We define aBroadcastExampleRequest
and aBroadcastExampleResponse
data models. Finally, create aprotocol
namedproto
with version1.0
:
class
BroadcastExampleRequest
(
Model
):
pass
class
BroadcastExampleResponse
(
Model
):
text
:
str
proto
=
Protocol
(name
=
"proto"
, version
=
"1.0"
)
Let's now define a message handler function for incoming messages of typeBroadcastExampleRequest
in the protocol:
@proto
.
on_message
(model
=
BroadcastExampleRequest, replies
=
BroadcastExampleResponse)
async
def
handle_request
(

ctx
:
Context
,
sender
:
str
,
_msg
:
BroadcastExampleRequest):
await
ctx
.
send
(sender,
BroadcastExampleResponse
(text
=
f
"Hello from
{
ctx.name
}
"
))
Here we defined ahandle_request()
function which is used whenever a request is received. This sends a response back to the sender. This function is decorated with the.on_message()
decorator indicating that this function is triggered whenever a message of typeBroadcastExampleRequest
is received. The function sends a response containing a greeting message with the name of the agent that sent the request in the first place.
Now, we need to include theprotocol
into the agents. Specifically, the protocol is included in bothalice
andbob
agents. This means they will follow the rules defined in the protocol when communicating:
alice
.
include
(proto)
bob
.
include
(proto)
ℹ
After the first registration in theAlmanac ↗
smart contract, it will take about 5 minutes before the agents can be found through the protocol.
It is now time to define the behavior and function ofcharles
agent.
@charles
.
on_interval
(period
=
5.0
)
async
def
say_hello
(
ctx
:
Context):
await
ctx
.
experimental_broadcast
(proto.digest, message

=
BroadcastExampleRequest
())
@charles
.
on_message
(model
=
BroadcastExampleResponse)
async
def
handle_response
(
ctx
:
Context
,
sender
:
str
,
msg
:
BroadcastExampleResponse):
ctx
.
logger
.
info
(
f
"Received response from
{
sender
}
:
{
msg.text
}
"
)
In the first part, we use the.on_interval()
decorator to define an interval behavior for this agent when the script is being run. In this case, the agent will execute thesay_hello()
function every 5 seconds. TheContext
object is a collection of data and functions related to the agent. Inside thesay_hello()
function, the agent uses thectx.experimental_broadcast()
method to send a broadcast message. The message is of typeBroadcastExampleRequest()
and it is being sent using the protocol's digest (proto.digest
).
Then, we defined a.on_message()
decorator which decorateshandle_response()
function. This function handles all incoming messages of typeBroadcastExampleResponse from other agents. When a response is received, it logs the information. Inside the
handle_response()function, the agent logs an informational message using
ctx.logger.info()` method to print the sender and the content of the message. The message includes the sender's name and the text content of the response message.
We are now ready to set up aBureau
object for agents to be run together at the same time, and we addalice
,bob
, andcharles
to it using thebureau.add()
method:
bureau
=
Bureau
(port
=

8000
, endpoint
=
"http://localhost:8000/submit"
)
bureau
.
add
(alice)
bureau
.
add
(bob)
bureau
.
add
(charles)
if
name
==
"**main**"
:
bureau
.
run
()
The bureau is assigned to listen onport=8000
and specifies anendpoint
at"http://localhost:8000/submit"
for submitting data.
Save the script.

The overall script should look as follows:

broadcast.py from uagents import Agent , Bureau , Context , Model , Protocol from uagents . setup import fund_agent_if_low

create agents

alice and bob will support the protocol

charles will try to reach all agents supporting the protocol

alice

Agent (name = "alice" , seed = "alice recovery phrase") bob =

Agent (name = "bob" , seed = "bob recovery phrase") charles =

Agent (name = "charles" , seed = "charles recovery phrase")

fund_agent_if_low (alice.wallet. address ()) fund_agent_if_low (bob.wallet. address ()) fund_agent_if_low (charles.wallet. address ())

class

BroadcastExampleRequest (Model): pass class

BroadcastExampleResponse (Model): text :

str

define protocol

proto

Protocol (name = "proto" , version = "1.0")

@proto . on_message (model = BroadcastExampleRequest, replies = BroadcastExampleResponse) async

def

handle_request (ctx : Context ,

sender :

str ,

_msg : BroadcastExampleRequest): await ctx . send (sender, BroadcastExampleResponse (text = f "Hello from { ctx.name } "))

include protocol

Note: after the first registration on the almanac smart contract, it will

take about 5 minutes before the agents can be found through the protocol

alice . include (proto) bob . include (proto)

let charles send the message to all agents supporting the protocol

@charles . on_interval (period = 5) async

def

say_hello (ctx : Context): await ctx . experimental_broadcast (proto.digest, message = BroadcastExampleRequest ())

@charles . on_message (model = BroadcastExampleResponse) async

def

handle_response (ctx : Context ,

sender :

str ,

msg : BroadcastExampleResponse): ctx . logger . info (f "Received response from { sender } : { msg.text } ")

bureau

Bureau (port = 8000 , endpoint = "http://localhost:8000/submit") bureau . add (alice) bureau . add (bob) bureau . add (charles)

if

name

==

"**main**" : bureau . run ()

Run the script

Make sure to have activated your virtual environment correctly.

Run the script:python broadcast.py

The output would be:

[setup]: Adding testnet funds to agent... [setup]: Adding testnet funds to agent...complete [alice]: Registering on almanac contract... [alice]: Registering on almanac contract...complete [bob]: Registering on almanac contract... [bob]: Registering on almanac contract...complete [charles]: Registering on almanac contract... [charles]: Registering on almanac contract...complete [bureau]: Starting server on http://0.0.0.0:8000 (Press CTRL+C to quit) [charles]: Received response from agent1q0mau8vkmg78xx0sh8cyl4tpl4ktx94pqp2e94cylu6haugt2hd7j9vequ7: Hello from bob [charles]: Received response from agent1qww3ju3h6kfcuqf54gkghvt2pqe8qp97a7nzm2vp8plfxflc0epzcjsv79t: Hello from alice [charles]: Received response from agent1q0mau8vkmg78xx0sh8cyl4tpl4ktx94pqp2e94cylu6haugt2hd7j9vequ7: Hello from bob [charles]: Received response from agent1qww3ju3h6kfcuqf54gkghvt2pqe8qp97a7nzm2vp8plfxflc0epzcjsv79t: Hello from alice [charles]: Received response from agent1q0mau8vkmg78xx0sh8cyl4tpl4ktx94pqp2e94cylu6haugt2hd7j9vequ7: Hello from bob [charles]: Received response from agent1qww3ju3h6kfcuqf54gkghvt2pqe8qp97a7nzm2vp8plfxflc0epzcjsv79t: Hello from alice

Was this page helpful?

How to use the agents to simulate a cleaning scenario ✨Agents name service