

AI Agents: broadcast

Introduction

Protocols cover a very important role within the uAgents Framework. [Protocols ↗](#) define the standards through which agents communication takes place. In this example, we will be showcasing a scenario where three agents, named `alice`, `bob`, and `charles`, 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 the `Agent`
3. `,Bureau`
4. `,Context`
5. `,Model`
6. `, andProtocol`
7. classes from the `uagents`
8. library, and the `fund_agent_if_low`
9. `fromuagents.setup`
10. `.` Then, let's create the 3 different agents using the class `Agent`
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, the `fund_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. `=`

```

53. Agent
54. (name
55. =
56. "charles"
57. , seed
58. =
59. "charles recovery phrase"
60. )
61. fund_agent_if_low
62. (alice.wallet.
63. address
64. ())
65. fund_agent_if_low
66. (bob.wallet.
67. address
68. ())
69. fund_agent_if_low
70. (charles.wallet.
71. address
72. ())
73. It is optional but useful to include aseed
74. parameter when creating an agent to set fixedaddresses
75. . Otherwise, random addresses will be generated every time you run the agent.
76. Let's then define two message data models to define the type of messages being handled and exchanged by the
    agents. We define aBroadcastExampleRequest
77. and aBroadcastExampleResponse
78. data models. Finally, create aprotocol
79. namedproto
80. with version1.0
81. :
82. class
83. BroadcastExampleRequest
84. (
85. Model
86. ):
87. pass
88. class
89. BroadcastExampleResponse
90. (
91. Model
92. ):
93. text
94. :
95. str
96. proto
97. =
98. Protocol
99. (name
100. =
101. "proto"
102. , version
103. =
104. "1.0"
105. )
106. Let's now define a message handler function for incoming messages of typeBroadcastExampleRequest
107. in the protocol:
108. @proto
109. .
110. on_message
111. (model
112. =
113. BroadcastExampleRequest, replies
114. =
115. BroadcastExampleResponse)
116. async
117. def
118. handle_request
119. (

```

```

120. ctx
121. :
122. Context
123. ,
124. sender
125. :
126. str
127. ,
128. _msg
129. :
130. BroadcastExampleRequest):
131. await
132. ctx
133. .
134. send
135. (sender,
136. BroadcastExampleResponse
137. (text
138. =
139. f
140. "Hello from
141. {
142. ctx.name
143. }
144. "
145. ))
146. Here we defined ahandle_request()
147. 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()
148. decorator indicating that this function is triggered whenever a message of typeBroadcastExampleRequest
149. 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.
150. Now, we need to include theprotocol
151. into the agents. Specifically, the protocol is included in bothalice
152. andbob
153. agents. This means they will follow the rules defined in the protocol when communicating:
154. alice
155. .
156. include
157. (proto)
158. bob
159. .
160. include
161. (proto)
162. i
163. After the first registration in theAlmanac ↗
164. smart contract, it will take about 5 minutes before the agents can be found through the protocol.
165. It is now time to define the behavior and function ofcharles
166. agent.
167. @charles
168. .
169. on_interval
170. (period
171. =
172. 5.0
173. )
174. async
175. def
176. say_hello
177. (
178. ctx
179. :
180. Context):
181. await
182. ctx
183. .
184. experimental_broadcast
185. (proto.digest, message

```

```

186. =
187. BroadcastExampleRequest
188. ())
189. @charles
190. .
191. on_message
192. (model
193. =
194. BroadcastExampleResponse)
195. async
196. def
197. handle_response
198. (
199. ctx
200. :
201. Context
202. ,
203. sender
204. :
205. str
206. ,
207. msg
208. :
209. BroadcastExampleResponse):
210. ctx
211. .
212. logger
213. .
214. info
215. (
216. f
217. "Received response from
218. {
219. sender
220. }
221. :
222. {
223. msg.text
224. }
225. "
226. )
227. In the first part, we use the.on_interval()
228. decorator to define an interval behavior for this agent when the script is being run. In this case, the agent will execute
    thesay_hello()
229. function every 5 seconds. TheContext
230. object is a collection of data and functions related to the agent. Inside thesay_hello()
231. function, the agent uses thectx.experimental_broadcast()
232. method to send a broadcast message. The message is of typeBroadcastExampleRequest()
233. and it is being sent using the protocol's digest (proto.digest
234. ).
235. Then, we defined a.on_message()
236. decorator which decorateshandle_response()
237. function. This function handles all incoming messages of typeBroadcastExampleResponse from other agents. When a
    response is received, it logs the information. Inside the
238. handle_response()function, the agent logs an informational message using
239. 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.
240. We are now ready to set up aBureau
241. object for agents to be run together at the same time, and we addalice
242. ,bob
243. , andcharles
244. to it using thebureau.add()
245. method:
246. bureau
247. =
248. Bureau
249. (port
250. =

```

```

251. 8000
252. , endpoint
253. =
254. "http://localhost:8000/submit"
255. )
256. bureau
257. .
258. add
259. (alice)
260. bureau
261. .
262. add
263. (bob)
264. bureau
265. .
266. add
267. (charles)
268. if
269. name
270. ==
271. "main"
272. :
273. bureau
274. .
275. run
276. ()
277. The bureau is assigned to listen on port=8000
278. and specifies an endpoint
279. at "http://localhost:8000/submit"
280. for submitting data.
281. 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 agent1q0mau8vkmg78xx0sh8cyl4tpl4ktx94ppq2e94cylu6haugt2hd7j9vequ7: Hello from bob [charles]: Received response from agent1qww3ju3h6kfcuqf54gkghvt2pqe8qp97a7nzm2vp8plfxflc0epzcjsv79t: Hello from alice [charles]: Received response from agent1q0mau8vkmg78xx0sh8cyl4tpl4ktx94ppq2e94cylu6haugt2hd7j9vequ7: Hello from bob [charles]: Received response from agent1qww3ju3h6kfcuqf54gkghvt2pqe8qp97a7nzm2vp8plfxflc0epzcjsv79t: Hello from alice [charles]: Received response from agent1q0mau8vkmg78xx0sh8cyl4tpl4ktx94ppq2e94cylu6haugt2hd7j9vequ7: 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](#)