

How to use agents to send tokens

Introduction

Within agent-based decentralized systems, efficient communication and secure data exchange are essential. In this guide, we will walk you through the process of setting up two AI Agents utilizing the `uagents` library to establish a dynamic workflow where one agent periodically sends payment requests to another, which in turn processes these requests, executes transactions, and provides transaction information back to sending agent.

Let's get started!

Walk-through

1. First of all, create a Python script for this task, and name it by running: `touch sending_tokens.py`
2. Then, import the necessary modules from `uagents`
3. `uagents.network`
4. `uagents.setup`
5. Let's then define two data models: `PaymentRequest`
6. `and TransactionInfo`
7. We then need to set up the values for the `AMOUNT`
8. `and DENOM`
9. variables, which define the default amount and denomination for the payment requests:
10. `from`
11. `uagents`
12. `import`
13. `Agent`
14. `,`
15. `Bureau`
16. `,`
17. `Context`
18. `,`
19. `Model`
20. `from`
21. `uagents`
22. `.`
23. `network`
24. `import`
25. `wait_for_tx_to_complete`
26. `from`
27. `uagents`
28. `.`
29. `setup`
30. `import`
31. `fund_agent_if_low`
32. `class`
33. `PaymentRequest`
34. `(`
35. `Model`
36. `):`
37. `wallet_address`
38. `:`
39. `str`
40. `amount`
41. `:`
42. `int`
43. `denom`
44. `:`
45. `str`
46. `class`
47. `TransactionInfo`
48. `(`
49. `Model`
50. `):`
51. `tx_hash`
52. `:`
53. `str`

```

54. AMOUNT
55. =
56. 100
57. DENOM
58. =
59. "atestfet"
60.
    ◦ ThePaymentRequest
61.
    ◦ model represents a payment request which contains thewallet_address
62.
    ◦ ,amount
63.
    ◦ , anddenom
64.
    ◦ .
65.
    ◦ TheTransactionInfo
66.
    ◦ model represents information about a transaction and contains a single attribute,tx_hash
67.
    ◦ .
68. Let's now define our agents,alice
69. andbob
70. . Ensure they have enough funds in their wallets to carry out transactions:
71. alice
72. =
73. Agent
74. (name
75. =
76. "alice"
77. , seed
78. =
79. "alice secret phrase"
80. )
81. bob
82. =
83. Agent
84. (name
85. =
86. "bob"
87. , seed
88. =
89. "bob secret phrase"
90. )
91. fund_agent_if_low
92. (bob.wallet.
93. address
94. ( ), min_balance
95. =
96. AMOUNT)
97. We can now define our agents behaviour and functions:
98. @alice
99. .
100. on_interval
101. (period
102. =
103. 10.0
104. )
105. async
106. def
107. request_funds
108. (
109. ctx
110. :
111. Context):
112. await
113. ctx

```

```

114. .
115. send
116. (
117. bob.address,
118. PaymentRequest
119. (
120. wallet_address
121. =
122. str
123. (alice.wallet.
124. address
125. ()), amount
126. =
127. AMOUNT, denom
128. =
129. DENOM
130. ),
131. )
132. This defines an event handler foralice
133. using the.on_interval()
134. decorator. This event handler is triggered at regular intervals of10.0
135. seconds in this case. The event handler function is namedrequest_funds()
136. and takes actx
137. parameter of typeContext
138. . Within the function,alice
139. sends a payment request message tobob
140. by using thectx.send()
141. method. Thectx.send()
142. method is called with the recipient address,bob.address
143. , which specifies that the message should be sent tobob
144. . The message is an instance of thePaymentRequest()
145. model. It containsalice
146. 's wallet address (alice.wallet.address())
147. ), the amount (AMOUNT
148. ), and the denomination (DENOM
149. ).
150. We can now define aconfirm_transaction()
151. message handler foralice
152. to handle incoming messages frombob
153. of typeTransactionInfo
154. :
155. @alice
156. .
157. on_message
158. (model
159. =
160. TransactionInfo)
161. async
162. def
163. confirm_transaction
164. (
165. ctx
166. :
167. Context
168. ,
169. sender
170. :
171. str
172. ,
173. msg
174. :
175. TransactionInfo):
176. ctx
177. .
178. logger
179. .
180. info
181. (

```

```
182. f
183. "Received transaction info from
184. {
185. sender
186. }
187. :
188. {
189. msg
190. }
191. "
192. )
193. tx_resp
194. =
195. await
196. wait_for_tx_to_complete
197. (msg.tx_hash, ctx.ledger)
198. coin_received
199. =
200. tx_resp
201. .
202. events
203. [
204. "coin_received"
205. ]
206. if
207. (
208. coin_received
209. [
210. "receiver"
211. ]
212. ==
213. str
214. (alice.wallet.
215. address
216. ())
217. and
218. coin_received
219. [
220. "amount"
221. ]
222. ==
223. f
224. "
225. {
226. AMOUNT
227. }{
228. DENOM
229. }
230. "
231. )
232. :
233. ctx
234. .
235. logger
236. .
237. info
238. (
239. f
240. "Transaction was successful:
241. {
242. coin_received
243. }
244. "
245. )
246. The event handler function is named confirm_transaction()
247. and takes three parameters: ctx
248. , sender
249. , and msg
```

```

250. . Within the function,alice
251. logs an informational message using thectx.logger.info()
252. method, indicating the receipt of transaction information from the sender agent,bob
253. , and displaying themsg
254. object. Thewait_for_tx_to_complete()
255. function is used to await the completion of the transaction specified by thetx_hash
256. received in the message.
257. Once the transaction is completed, the code accesses thecoin_received
258. event from the transaction response usingtx_resp.events["coin_received"]
259. . It checks if the receiver address matchesalice
260. 's wallet address (alice.wallet.address())
261. ) and if the amount received matches the expected amount (AMOUNT + DENOM
262. ).
263. If the conditions are met,alice
264. logs another informational message indicating the success of the transaction and displaying the details of the received
    coins.
265. Let's now define an event handler forbob
266. . This event handler is triggered whenbob
267. receives a message of typePaymentRequest
268. fromalice
269. :
270. @bob
271. .
272. on_message
273. (model
274. =
275. PaymentRequest, replies
276. =
277. TransactionInfo)
278. async
279. def
280. send_payment
281. (
282. ctx
283. :
284. Context
285. ,
286. sender
287. :
288. str
289. ,
290. msg
291. :
292. PaymentRequest):
293. ctx
294. .
295. logger
296. .
297. info
298. (
299. f
300. "Received payment request from
301. {
302. sender
303. }
304. :
305. {
306. msg
307. }
308. "
309. )

```

310. **send the payment**

```

311. transaction
312. =
313. ctx

```

```

314. .
315. ledger
316. .
317. send_tokens
318. (
319. msg.wallet_address, msg.amount, msg.denom, bob.wallet
320. )

```

321. **send the tx hash so alice can confirm**

```

322. await
323. ctx
324. .
325. send
326. (alice.address,
327. TransactionInfo
328. (tx_hash
329. =
330. transaction.tx_hash))
331. The event handler function is named send_payment()
332. and takes three parameters: ctx
333. , sender
334. , and msg
335. . Within the function, bob
336. logs an informational message using the ctx.logger.info()
337. method, indicating the receipt of a payment request from the sender agent, bob
338. , and displaying the msg
339. object.
340. Next, the code performs a payment transaction using the ctx.ledger.send_tokens()
341. method. It takes the wallet address (msg.wallet_address
342. ), amount (msg.amount
343. ), denomination (msg.denom
344. ), and bob.wallet()
345. as parameters. This method is responsible for sending the requested payment.
346. Once the transaction is completed, bob
347. sends a message back to alice
348. to inform her about the transaction, using ctx.send()
349. . The message is created using the TransactionInfo
350. model with the tx_hash
351. obtained from the transaction response. The ctx.send()
352. method is used to send this message to alice using her address (alice.address
353. ).
354. We are now ready to use the Bureau class to create a bureau
355. object and add both uAgents to it so for them to be run together:
356. bureau
357. =
358. Bureau
359. ()
360. bureau
361. .
362. add
363. (alice)
364. bureau
365. .
366. add
367. (bob)
368. if
369. name
370. ==
371. "main"
372. :
373. bureau
374. .
375. run
376. ()

```

The overall script for this example should look as follows:

```
sending_tokens.py from uagents import Agent , Bureau , Context , Model from uagents . network import
wait_for_tx_to_complete from uagents . setup import fund_agent_if_low
```

```
class
```

```
PaymentRequest ( Model ): wallet_address :
```

```
str amount :
```

```
int denom :
```

```
str
```

```
class
```

```
TransactionInfo ( Model ): tx_hash :
```

```
str
```

AMOUNT

```
100 DENOM =
```

```
"atestfet"
```

alice

```
Agent (name = "alice" , seed = "alice secret phrase" ) bob =
```

```
Agent (name = "bob" , seed = "bob secret phrase" )
```

```
fund_agent_if_low (bob.wallet. address (), min_balance = AMOUNT)
```

```
@alice . on_interval (period = 10.0 ) async
```

```
def
```

```
request_funds ( ctx : Context): await ctx . send ( bob.address, PaymentRequest ( wallet_address = str (alice.wallet. address
()), amount = AMOUNT, denom = DENOM ), )
```

```
@alice . on_message (model = TransactionInfo) async
```

```
def
```

```
confirm_transaction ( ctx : Context ,
```

```
sender :
```

```
str ,
```

```
msg : TransactionInfo): ctx . logger . info ( f "Received transaction info from { sender } : { msg } " ) tx_resp =
```

```
await
```

```
wait_for_tx_to_complete (msg.tx_hash, ctx.ledger)
```

coin_received

```
tx_resp . events [ "coin_received" ] if ( coin_received [ "receiver" ]
```

```
==
```

```
str (alice.wallet. address ())) and coin_received [ "amount" ]
```

```
==
```

```
f " { AMOUNT } { DENOM } " : ctx . logger . info ( f "Transaction was successful: { coin_received } " )
```

```
@bob . on_message (model = PaymentRequest, replies = TransactionInfo) async
```

```
def
send_payment ( ctx : Context ,
sender :
str ,
msg : PaymentRequest): ctx . logger . info ( f "Received payment request from { sender } : { msg } " )
```

send the payment

transaction

```
ctx . ledger . send_tokens ( msg.wallet_address, msg.amount, msg.denom, bob.wallet )
```

send the tx hash so alice can confirm

```
await ctx . send (alice.address, TransactionInfo (tx_hash = transaction.tx_hash))
```

bureau

```
Bureau () bureau . add (alice) bureau . add (bob)
```

```
if
```

```
name
```

```
==
```

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

Run the script

On your terminal, make sure to have activated the virtual environment.

Run the script:python sending_tokens.py

The output should be as follows:

```
[ bob]: Received payment request from agent1qdp9j2ev86k3h5acaayjm8tpx36zv4mjxn05pa2kwesspstzj697xy5vk2a:
wallet_address='fetch1967p3vvp0yngdfturv4ypq2p4g760ml705wcxy' amount=100 denom='atestfet' [alice]: Received
transaction info from agent1q2kxet3vh0scsf0sm7y2erzz33cve6tv5uk63x64upw5g68kr0chkv7hw50:
tx_hash='DB662CCF415C7B0C9A02928967BE1817506D02A041AA05CA48DCE5CF87D5A638' [alice]: Transaction was
successful: {'receiver': 'fetch1967p3vvp0yngdfturv4ypq2p4g760ml705wcxy', 'amount': '100atestfet'}
```

Was this page helpful?

[Public and private Agents Options for running your agents](#)