

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. ())
95. fund_agent_if_low
96. (alice.wallet.
97. address
98. ())
99. We can now define our agents behaviour and functions:
100. @alice
101. .
102. on_interval
103. (period
104. =
105. 10.0
106. )
107. async
108. def
109. request_funds
110. (
111. ctx
112. :
113. Context):

```

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

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

```

250. ,sender
251. , andmsg
252. . Within the function,alice
253. logs an informational message using thectx.logger.info()
254. method, indicating the receipt of transaction information from the sender agent,bob
255. , and displaying themsg
256. object. Thewait_for_tx_to_complete()
257. function is used to await the completion of the transaction specified by thetx_hash
258. received in the message.
259. Once the transaction is completed, the code accesses thecoin_received
260. event from the transaction response usingtx_resp.events["coin_received"]
261. . It checks if the receiver address matchesalice
262. 's wallet address (ctx.wallet.address()
263. ) and if the amount received matches the expected amount (AMOUNT + DENOM
264. ).
265. If the conditions are met,alice
266. logs another informational message indicating the success of the transaction and displaying the details of the received
    coins.
267. Let's now define an event handler forbob
268. . This event handler is triggered whenbob
269. receives a message of typePaymentRequest
270. fromalice
271. :
272. @bob
273. .
274. on_message
275. (model
276. =
277. PaymentRequest, replies
278. =
279. TransactionInfo)
280. async
281. def
282. send_payment
283. (
284. ctx
285. :
286. Context
287. ,
288. sender
289. :
290. str
291. ,
292. msg
293. :
294. PaymentRequest):
295. ctx
296. .
297. logger
298. .
299. info
300. (
301. f
302. "Received payment request from
303. {
304. sender
305. }
306. :
307. {
308. msg
309. }
310. "
311. )

```

312. **send the payment**

```

313. transaction

```

```

314. =
315. ctx
316. .
317. ledger
318. .
319. send_tokens
320. (
321. msg.wallet_address, msg.amount, msg.denom, ctx.wallet
322. )

```

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

```

324. await
325. ctx
326. .
327. send
328. (alice.address,
329. TransactionInfo
330. (tx_hash
331. =
332. transaction.tx_hash))
333. The event handler function is named send_payment()
334. and takes three parameters: ctx
335. , sender
336. , and msg
337. . Within the function, bob
338. logs an informational message using the ctx.logger.info()
339. method, indicating the receipt of a payment request from the sender agent, bob
340. , and displaying the msg
341. object.
342. Next, the code performs a payment transaction using the ctx.ledger.send_tokens()
343. method. It takes the wallet address (msg.wallet_address
344. ), amount (msg.amount
345. ), denomination (msg.denom
346. ), and ctx.wallet()
347. as parameters. This method is responsible for sending the requested payment.
348. Once the transaction is completed, bob
349. sends a message back to alice
350. to inform her about the transaction, using ctx.send()
351. . The message is created using the TransactionInfo
352. model with the tx_hash
353. obtained from the transaction response. The ctx.send()
354. method is used to send this message to alice using her address (alice.address
355. ).
356. We are now ready to use the Bureau class to create a bureau
357. object and add both uAgents to it so for them to be run together:
358. bureau
359. =
360. Bureau
361. ()
362. bureau
363. .
364. add
365. (alice)
366. bureau
367. .
368. add
369. (bob)
370. if
371. name
372. ==
373. "main"
374. :
375. bureau
376. .
377. run
378. ()

```

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 () ) fund_agent_if_low ( alice.wallet. address () )
```

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

```
def
```

```
request_funds ( ctx : Context ): await ctx . send ( bob.address, PaymentRequest ( wallet_address = str ( ctx.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 ( ctx.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, ctx.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?

[How to use agents to verify messages](#) [How to use the agents to simulate a cleaning scenario](#) ☆☆☆