Agent Asynchronous Loops

Introduction

Agents need to communicate, perform tasks, and respond to events simultaneously and independently within any decentralized system. This guide shows how to create asynchronous agents that operate in parallel, enabling them to handle their own workflows while still interacting with other agents or external processes.

By usingasynchronous loops and attaching agents toexternal event loops , you can build agents that manage tasks simultaneously, send periodic updates, and process incoming messages in real-time. This approach is particularly useful when working with distributed systems, where agents must collaborate or handle multiple simultaneous operations without interruptions.

Supporting documentation

- Creating an agent
- Creating an interval task
- Communicating with other agents
- Agent Handlers

Walk-through

The following scripts show how to define agents, manage their life-cycle and attach them to external asynchronous loops.

Script 1

The first script depicts how toattach an agent to an external event loop and allow it to runsimultaneously with other asynchronous tasks.

First of all, let's create a Python script:

windows echo .

external_loop_attach . py Now, paste the below code into it:

Self hosted external_loop_attach.py import asyncio import contextlib

from uagents import Agent , Bureau , Context

loop

asyncio . get_event_loop ()

agent

Agent (name = "looper" , seed = "" ,)

bureau

Bureau (agents = [agent],)

@agent . on_event ("startup") async

def

startup (ctx : Context): ctx . logger . info (">>> Looper is starting up.")

@agent . on_event ("shutdown") async

def

shutdown (ctx : Context): ctx . logger . info (">>> Looper is shutting down.")

async

def

coro (): while

True : print ("doing hard work...") await asyncio . sleep (1)

if

name

==

"**main**" : print ("Attaching the agent or bureau to the external loop...") loop . create_task (coro ())

> when attaching the agent to the external loop

loop . create_task (agent. run_async ())

> when attaching a bureau to the external loop

loop.create_task(bureau.run_async())

with contextlib . suppress (KeyboardInterrupt): loop . run_forever () This script is for an agent using an external event loop. We first import the required libraries for this script to be run correctly. We then proceed and instantiate an agent calledlooper using aseed . Remember that you need to provide a seed within the field parameter. We then need to create abureau to manage the agents. We can now add thelooper agent into thebureau .

We can proceed and define astartup() function decorated using the.on_event("startup") decorator. The function is triggered when the agent is started and it logs a message indicating the agent has started. Similarly, theshutdown() function is triggered when the agent shuts down, logging an appropriate message.

We go on and define a functioncoro() which simulates a separate, long-running task that will print"Doing hard work..." every second. This task runs independently of the agent and showcases the agents' ability to handle multiple simultaneous tasks. We now need to attach both the agent and the external task (coro) to the same event loop usingloop.create_task() . This allows both the agent and other tasks to execute simultaneously in an asynchronous way.

In the__main__ block, we define a message to be printed indicating that the process of attaching the agent or bureau to the asynchronous event loop has started. Theloop.create_task(coro()) adds a coroutine (coro) to the event loop.coro is the task performing"doing hard work..." asynchronously. This allows the task to run simultaneously with other tasks managed by the event loop without blocking the rest of the program.

The agent is attached to the external event loop by usingloop.create_task() to schedule the agent's asynchronous operation. The methodagent.run_async() is a non-blocking function that runs the agent within the event loop, allowing the agent to perform its tasks and handle events simultaneously.

Finally, in the last line we create a context manager that suppressesKeyboardInterrupt exceptions, which are typically raised when the user pressesCtrl+C to stop the program. This ensures that the program can shut down without printing a traceback or throwing an error when the user stops it manually.

Script 2

The goal of the second script is to create an agent that runs tasks inside an external event loop. The agent can execute certain actions (e.g., print messages or respond to events) while simultaneously performing a separate background task.

Let's start by creating a Python script:

windows echo .

external_loop_run . py Then, let's paste the below code into it:

Self hosted external_loop_run.py import asyncio

from uagents import Agent , Bureau , Context

loop

asyncio . get_event_loop ()

agent

Agent (name = "looper" , seed = "" , loop = loop,)

bureau

Bureau (agents = [agent], loop = loop,)

@agent . on_event ("startup") async

def

startup (ctx : Context): ctx . logger . info (">>> Looper is starting up.")

@agent . on_event ("shutdown") async

def

shutdown (ctx : Context): ctx . logger . info (">>> Looper is shutting down.")

async

def

coro (): while

True : print ("doing hard work...") await asyncio . sleep (1)

if

name

==

"**main**" : print ("Starting the external loop from the agent or bureau...") loop . create_task (coro ())

> when starting the external loop from the agent

agent . run ()

> when starting the external loop from the bureau

bureau.run()

We start by importing the required libraries to correctly run this script. We then create an asynchronous event loop usingasyncio.get_event_loop() . This loop is used to handle all asynchronous operations, such as the agent's actions and background tasks.

We proceed and create an agent calledlooper using theAgent class. The agent takes three parameters:name ,seed , andloop . Remember to provide aseed for your agent otherwise a random address will be generated every time you run the agent. We then create abureau object using theBureau class. Thebureau is created with a single agent,looper .

We can then define our agent functions to handle the agent's lifecycle events:

1. startup()
2. : This function runs when the agent is started. It logs a message to indicate that the agent has been started up.
3. shutdown()
4. : This function runs when the agent is shut down. It logs a message to indicate that the agent has been stopped.

In the next step, we define thecoro() function; As before, this function defines an infinite loop where the agent performs a task ("doing hard work...") every second. This simulates a long-running background task. Theawait asyncio.sleep(1) pauses execution for one second between each iteration, allowing other tasks to run during that time.

Finally, in the__main__ block, we define a message to be printed indicating that the external event loop is being started. Theloop.create_task(coro()) schedules thecoro() coroutine to run in the background, simultaneously with the agent's operations.

Expected Output

We are now ready to run the scripts.

The output should be similar to the following:

- Script 1:
- Attaching the agent or bureau to the external loop...
- Doing hard work...
- Doing hard work...
- Doing hard work...

- Looper is starting up.

- Script 2:
- Starting the external loop from the agent or bureau...
- Doing hard work...
- Doing hard work...
- Doing hard work...

- Looper is starting up.

Last updated on October 17, 2024

Was this page helpful?

You can also leave detailed feedbackon Github

Almanac Contract Agent Envelopes

On This Page