Bureau

Introduction

TheBureau is a powerful orchestration component that allows multiple agents to operate together within a shared environment. It manages the lifecycle of agents, including their communication, task handling, and external integrations. The Bureau simplifies the process of running agents in a collaborative and coordinated fashion, whether they need to exchange messages internally or interact with external systems.

Let's get started!

Prerequisites

Make sure you have read the following resources before going on with this guide:

- Quick Start Guide for uAgents Framework
- Creating your first agent
- Agents address
- Almanac contract
- Register in Almanac

Imports needed

- uAgents(opens in a new tab)

Bureau Walkthrough

This walkthrough will guide you through the process of setting up and running agents using the Bureau, demonstrating its key functionalities, including agent management, message handling, and external communication via endpoints.

Step 1: Setting Up Agents

To begin, you need to create and define your agents. Each agent will have specific tasks and behaviors that can be scheduled or triggered by messages.

agent-bureau.py from uagents import Agent , Context , Model

class

Message (Model): message :

str

agent_a

Agent (name = "agent_a" , seed = "agent_a recovery phrase") agent_b =

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

Step 2: Defining Agent Behavior

Next, define the behavior of each agent, such as sending and receiving messages. This is done using decorators like on_interval to perform scheduled tasks and on_message to respond to incoming messages.

agent-bureau.py @agent_a . on_interval (period = 3.0) async

def

send_message (ctx : Context): await ctx . send (agent_b.address, Message (message = "Hello from agent_a"))

Handle received messages in agent_a

@agent_a . on_message (model = Message) async def

agent_a_message_handler (ctx : Context ,

sender :

str ,

msg : Message): ctx . logger . info (f "Received message from { sender } : { msg.message } ")

Handle received messages in agent_b

@agent_b . on_message (model = Message) async

def

agent_b_message_handler (ctx : Context ,

sender :

str ,

msg : Message): ctx . logger . info (f "Received message from { sender } : { msg.message } ") await ctx . send (agent_a.address, Message (message = "Reply from agent_b"))

Step 3: Creating the Bureau

Once the agents are set up, you can create a Bureau instance to manage them. The Bureau is responsible for running the agents, handling communication, and orchestrating tasks.

Arguments the Bureau Takes:

The Bureau can be customized through a few important arguments when instantiated:

- agents
- (Optional[List[Agent]]): A list of agents to be managed by the Bureau. If you don't add agents during initialization, you can always add them later using theadd()
- method.
- port
- (Optional[int]): The port number on which the Bureau's ASGI server will run. This is crucial if your agents need to expose REST APIs for external communication. The default port is8000
- , but you can specify any available port.
- endpoint
- (Optional[Union[str, List[str], Dict[str, dict]]]): Configuration for the agent endpoints. You can specify how agents communicate with external systems via REST. This could be a string, a list of strings, or a dictionary defining more complex configurations.
- loop
- (Optional[asyncio.AbstractEventLoop]): The event loop that the Bureau will use to manage asynchronous tasks. By default, the Bureau creates an event loop if one is not provided.
- log_level
- (Optional[Union[int, str]]): The logging level for the Bureau. This allows you to control how much information is logged during the Bureau’s execution, using levels likeINFO
- ,DEBUG
- , orERROR
- .

from uagents import Bureau

bureau

Bureau ()

or

Initialize the Bureau, optionally specifying port, agents, and other configurations

bureau

Bureau (port = 8000 , agents = [agent_a, agent_b])

bureau . add (agent_a) bureau . add (agent_b)

Step 4: Running the Bureau

After adding your agents, it's time to run the Bureau. The Bureau will ensure that the agents communicate seamlessly and handle tasks like message delivery and API management if necessary.

if

name

==

"**main**" : bureau . run () Step 5: Message Handling and Communication

In this setup,agent_a sends a message toagent_b every 3 seconds. Whenagent_b receives the message, it replies back toagent_a . The Bureau handles the coordination and message passing between the agents. You'll see logs indicating the messages sent and received by each agent.

The overall script for this example should look as follows:

Self hosted agent-bureau.py from uagents import Agent , Bureau , Context , Model

class

Message (Model): message :

str

agent_a

Agent (name = "agent_a" , seed = "agent_a recovery phrase") agent_b =

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

@agent_a . on_interval (period = 3.0) async

def

send_message (ctx : Context): await ctx . send (agent_b.address, Message (message = "Hello from agent_a"))

@agent_a . on_message (model = Message) async

def

agent_a_message_handler (ctx : Context ,

sender :

str ,

msg : Message): ctx . logger . info (f "Received message from { sender } : { msg.message } ")

@agent_b . on_message (model = Message) async

def

agent_b_message_handler (ctx : Context ,

sender :

str ,

msg : Message): ctx . logger . info (f "Received message from { sender } : { msg.message } ") await ctx . send (agent_a.address, Message (message = "Reply from agent_b"))

bureau

Bureau () bureau . add (agent_a) bureau . add (agent_b)

if

name

==

"**main**" : bureau . run () We are now ready to run the script:python agent-bureau.py

The output would be:

WARNING: [agent_a]: No endpoints provided. Skipping registration: Agent won't be reachable. WARNING: [agent_b]: No endpoints provided. Skipping registration: Agent won't be reachable. INFO: [agent_b]: Received message from agent1q2n33nmfscfscnz49a9e6nj4054d7r46v7x7522g4zh798tcwgs5q855p6q: Hello from agent_a INFO: [bureau]: Starting server on http://0.0.0.0:8000 (Press CTRL+C to quit) INFO: [agent_a]: Received message from agent1q07ypwau2gv0y005m0ltycx7vpt3hpajsvvnltpu50wpgk87zzkmwc005ga: Reply from agent_b INFO: [agent_b]: Received message from agent1q2n33nmfscfscnz49a9e6nj4054d7r46v7x7522g4zh798tcwgs5q855p6q: Hello from agent_a INFO: [agent_a]: Received message from agent1q07ypwau2gv0y005m0ltycx7vpt3hpajsvvnltpu50wpgk87zzkmwc005ga: Reply from agent_b INFO: [agent_b]: Received message from agent1q2n33nmfscfscnz49a9e6nj4054d7r46v7x7522g4zh798tcwgs5q855p6q: Hello from agent_a INFO: [agent_a]: Received message from agent1q07ypwau2gv0y005m0ltycx7vpt3hpajsvvnltpu50wpgk87zzkmwc005ga: Reply from agent_b Last updated on October 23, 2024

Was this page helpful?

You can also leave detailed feedback[on Github]

[REST endpoints] [Broadcast]

On This Page