Broadcast

Introduction toctx.broadcast()

Thectx.broadcast() function is used within an agent to send a message to all other agents that support a specific protocol. It is a powerful method when you want to communicate with multiple agents at once without knowing their addresses beforehand. This is especially useful in systems where many agents may be participating in the same protocol, and you want to notify or query all of them simultaneously.

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

Overview

In this guide, we will create a multi-agent broadcast system using theuagents library. Three agents will be initialized: Ethan, Olivia, and Liam. Ethan and Olivia will support a defined communication protocol, while Liam will broadcast a message to all agents supporting this protocol and handle their responses. We'll usectx.broadcast to manage communication between agents and demonstrate the structure of message handling within the context of theuagents framework.

Step 1: Import Classes and Initialize Agents

from uagents import Agent , Bureau , Context , Model , Protocol

ethan

Agent (name = "ethan" , seed = "ethan recovery phrase") olivia =

Agent (name = "olivia" , seed = "olivia recovery phrase") liam =

Agent (name = "liam" , seed = "liam recovery phrase") In this step, we import the core classes needed to define agents, protocols, and message handling. We then initialize three agents: Ethan, Olivia, and Liam, each with unique seed phrases, ensuring fixed addresses for their identities.

Step 2: Define Message Models

class

BroadcastExampleRequest (Model): pass

class

BroadcastExampleResponse (Model): text :

str We define two message models:

- BroadcastExampleRequest
- : This model represents a request message (empty for now).
- BroadcastExampleResponse
- : This model holds a response message containing a text field.

Step 3: Define a Communication Protocol

proto

Protocol (name = "proto" , version = "1.0") A protocol named proto with version 1.0 is defined. This protocol will govern communication between agents that support it.

Step 5: Message Handler for Broadcast Requests

@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.agent.name } ")) This function handles incomingBroadcastExampleRequest messages. When an agent (e.g., Ethan or Olivia) receives a request, it replies with aBroadcastExampleResponse , sending a message containing its own name.

Step 6: Include the Protocol in Agents

ethan . include (proto) olivia . include (proto) We include the protocol in Ethan and Olivia. This allows them to participate in communication governed by theproto protocol.

Step 7: Liam's Behavior - Broadcasting and Handling Responses

@liam . on_interval (period = 5) async

def

say_hello (ctx : Context): status_list =

await ctx . broadcast (proto.digest, message = BroadcastExampleRequest ()) ctx . logger . info (f "Trying to contact { len (status_list) } agents.") Thesay_hello function is executed by Liam every 5 seconds (usingon_interval). Liam sends aBroadcastExampleRequest message to all agents supporting theproto protocol viactx.broadcast . Theproto.digest is used to identify the protocol.

@liam . on_message (model = BroadcastExampleResponse) async

def

handle_response (ctx : Context ,

sender :

str ,

msg : BroadcastExampleResponse): ctx . logger . info (f "Received response from { sender } : { msg.text } ") When Liam receives a BroadcastExampleResponse, the handle_response function logs the sender and the content of the message.

Step 8: Run the Bureau

bureau

Bureau (port = 8000 , endpoint = "http://localhost:8000/submit") bureau . add (ethan) bureau . add (olivia) bureau . add (liam)

if

name

==

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

Explanation of ctx.broadcast

The broadcast() method in the Context class allows an agent to send a message to multiple agents that support a specific protocol. Here's a detailed look at its usage:

status_list = await ctx.broadcast(proto.digest, message=BroadcastExampleRequest())

- proto.digest
- : This is a unique identifier for the protocol that the message will be sent under. Only agents supporting this protocol will receive the broadcast.
- message
- : The message being broadcast, in this case, a BroadcastExampleRequest.
- status_list
- : This is a list of statuses returned by the broadcast, indicating which agents were successfully contacted.

By calling broadcast, Liam attempts to contact all agents supporting the proto protocol. This communication is asynchronous and can target multiple agents concurrently.

The overall script for this example should look as follows:

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

ethan

Agent (name = "ethan" , seed = "ethan recovery phrase") olivia =

Agent (name = "olivia" , seed = "olivia recovery phrase") liam =

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

class

BroadcastExampleRequest (Model): pass

class

BroadcastExampleResponse (Model): text :

str

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.agent.name } "))

ethan . include (proto) olivia . include (proto)

@liam . on_interval (period = 5) async

def

say_hello (ctx : Context): status_list =

await ctx . broadcast (proto.digest, message = BroadcastExampleRequest ()) ctx . logger . info (f "Trying to contact { len (status_list) } agents.")

@liam . 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 (ethan) bureau . add (olivia) bureau . add (liam)

if

name

==

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

The output would be:

INFO: [ethan]: Registration on Almanac API successful INFO: [ethan]: Almanac contract registration is up to date! INFO: [olivia]: Registration on Almanac API successful INFO: [olivia]: Almanac contract registration is up to date! INFO: [liam]: Registration on Almanac API successful INFO: [liam]: Almanac contract registration is up to date! INFO: [bureau]: Starting server on http://0.0.0.0:8000 (Press CTRL+C to quit) INFO: [liam]: Trying to contact 4 agents. INFO: [liam]: Received response from agent1q2hdqe8hxa6g0awspktktgc5furywq5jur5q9whh9hzyffxsm9ka6c2dmhz: Hello from olivia INFO: [liam]: Received response from agent1qff9zl5cehj2z68zef7q68uw76jjslh2r8xda93avayedqajzjwwyce8pt9: Hello from ethan Last updated on October 23, 2024

Was this page helpful?

You can also leave detailed feedbackon Github

Bureau Introducing dialogues

On This Page