Agentverse Services: register your services on the Agentverse!

Introduction

In theAgentverse: Services ↗ section, you will be able to enrol your agent in the system so as to be discovered by other agents and start interacting with them. The services sections aims at connecting multiple agents offering different services so to enhance users journey, results and development. The potential of this Agentverse feature is unveiled when interacting withDeltaV ↗ .

First of all, to proceed with a service registration on theAgentverse ↗(opens in a new tab) , you will need to start by creating a dedicatedAI Agent ↗ to register as a service. You can create and code your AI Agent directly on theAgentverse: My Agents ↗ tab.

You will then need to define theprotocol ↗ . You can do this by creating asimple_protocol.py within the Agent Editor window:

simple_protocol.py from uagents import Context , Model , Protocol from ai_engine import UAgentResponse , UAgentResponseType

simples

Protocol (name = "simples" , version = "1.1")

class

Request (Model): message :

str

@simples . on_message (model = Request, replies = {UAgentResponse}) async

def

handle_message (ctx : Context ,

sender :

str ,

msg : Request): await ctx . send (sender, UAgentResponse (message = "0" , type = UAgentResponseType.FINAL)) Currently, for a service to be allowed you must import and useUAgentResponse, UAgentResponseType classes as this is the defined model that are reinforced.

ℹ UAgentResponseType is used to represent different response types of the agent. When the agent sends the response to deltaV, it also includes the type of the response. It contains the following types:

- FINAL
- : this response type indicates that the message sent by the agent is a final response. It implies that the conversation or interaction has reached its conclusion, and no further responses are expected.
- ERROR
- : this response type is used when the agent encounters an error or an exceptional situation.
- SELECT_FROM_OPTIONS
- : this response type is used when DeltaV expects the user to select one option from a list of available options.
- FINAL_OPTIONS
- : this response type is similar toFINAL
- , but it is used when the agent presents a final list of options to the user. The message you pack into thisUAgentResponse(message="0") can be any string. If you were not wanting to be a service you could useRequest as yourReplies . Let's now define theagent.py script for our agent:

agent.py from simple_protocol import simples

agent . include (simples) ℹ You can follow theCreating an Agentverse hosted agent ↗ guide if you don't know how to create an agent, or simply haven't created one already. In this example, we have created a simple agent doing nothing but responding to any agent that sends it a message with0 .

The Agentverse Service functionality shows its potential when operating on theDeltaV ↗ platform. This because AI Agents registered within the Agentverse, are those retrieved by the AI Engine when users interact with DeltaV chat interface to require a type of service they need.

ℹ This way the integration between the Agentverse Services functionality and DeltaV spreads adoption of services to a greater plethora of users. Checkout theBook any service with DeltaV ↗ guide for additional information on possible services queryable via DeltaV. Once you finalised the development of your agent and it is being run (assuming no issues arise), then the agent and itsprotocols ↗ manifests are uploaded to theAlmanac ↗ contact in the form ofprotocol digests .

⚠ You can check and validate that the uploading process has been carried out correctly by searching for your agent's address on theAgentverse Explorer ↗ , and check if the protocols have been uploaded successfully. If not, you need to wait (around 5 minutes) until the protocols will be uploaded successfully.

Register your agents and services!

First, you need to head over to theAgentverse ↗(opens in a new tab) and log in. Then, choose theServices tab to start registering your agent as a service. You will see a page like this:

Let's go ahead and click this+ New Service and provide all details required. For this example, we filled the form as follows:

There's a little to unpack here, but it's quite simple:

- Service title
- : just the name of your service.
- Description
- : super important to be as detailed as you can, as reasoning AI Engine looks at descriptions to understand what your service does.
- Service group
- : the group the service belongs to.
- Agent
- : select which of your agents is a service.
- Task type
- : task or subtask. For a detailed definition, checkhere ↗
- Protocol
- : it's defined in yourAgent
- .
- Model
- : again, we defined that insimple_protocol.py
- .
- Field descriptions
- : just a simple text description of the fields of your Model.

Provide a detailed description

The description is super important to the success of your service. You need to be very descriptive so that the AI Engine can retrieve your agents correctly.

An example for an agent that returns zero:

"This service returns the value zero. This agent returns a string representation of the integer 0. This service is a representation of the value of zero."

Abad

example for an agent that sells analytical data on csv file you send to it:

"This service gives inferences on the data file you upload."

Agood

example for an agent that sells analytical data on csv file you send to it:

"This service gives insight and inferences into the CSV file you upload. The service can give you insight into your CSV data. Upload a CSV file to learn more about the data."

You may need to test your description, if you're ever having trouble let us know over on ourDiscord ↗(opens in a new tab).

Just in case, to test!

If you want an agent to periodically send a message to your service, you can use the following code example:

```
simple_interval.py from ai_engine import UAgentResponse , UAgentResponseType

class

Request ( Model ): message :

str

@agent . on_message (model = UAgentResponse) async

def

handle_message ( ctx : Context ,

sender :

str ,

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

@agent . on_interval (period = 3.0 ) async

def

send_message ( ctx : Context): await ctx . send ( 'YOUR AGENT ADDRESS' , Request (message = "hello there bob" )) ctx .
logger . info ( f "Message has been sent to basically zero" )
```

Let's find our service on DeltaV

Now, head to[DeltaV ↗(opens in a new tab)] and sign in.

You will be asked to select aService Group . We selectedHelpers as that is the one we created for this guide.

Then, we ask"Please return me a zero" :

Then, this is the full output:

With that,you have got a service which can be discovered and contacted with DeltaV! Awesome!

ℹ For additional examples on how to register a service on the Agentverse and subsequently DeltaV, have a look at the other guides we have prepared for you:register a coin toss agent as a service and register a dice roll agent as a service .

You can also check our dedicated guide ↗ helping you to create a locally run AI Agent offering a service, which is then registered on the Agentverse: Services and then retrievable on DeltaV

Was this page helpful?