

## Project Omnibus An OpenAI Agent Toolshed

### Project Description

Python sample code, built on the OpenAI Beta Assistant API, to aid the integration of OpenAI LLMs (e.g. ChatGPT4) into applications using MQTT messaging.

Documentation for the API is available at  
<https://platform.openai.com/docs/overview>.

MQTT is a light-weight, easy-to-use, publish-subscribe messaging bus widely used in internet of things (IoT) communications. It is multi-platform (e.g. Windows, Linux) and is supported by all major software languages. The site is <https://mosquitto.org/> and downloads are available at <https://mosquitto.org/download/>. The service must be running and the machine that runs the service must be reliable.

Access to the GPT-4 Assistant API requires an OpenAI account and an OpenAI key from openai.com. Set the OPENAI\_API\_KEY environment variable using the Windows Powershell "setx" command.

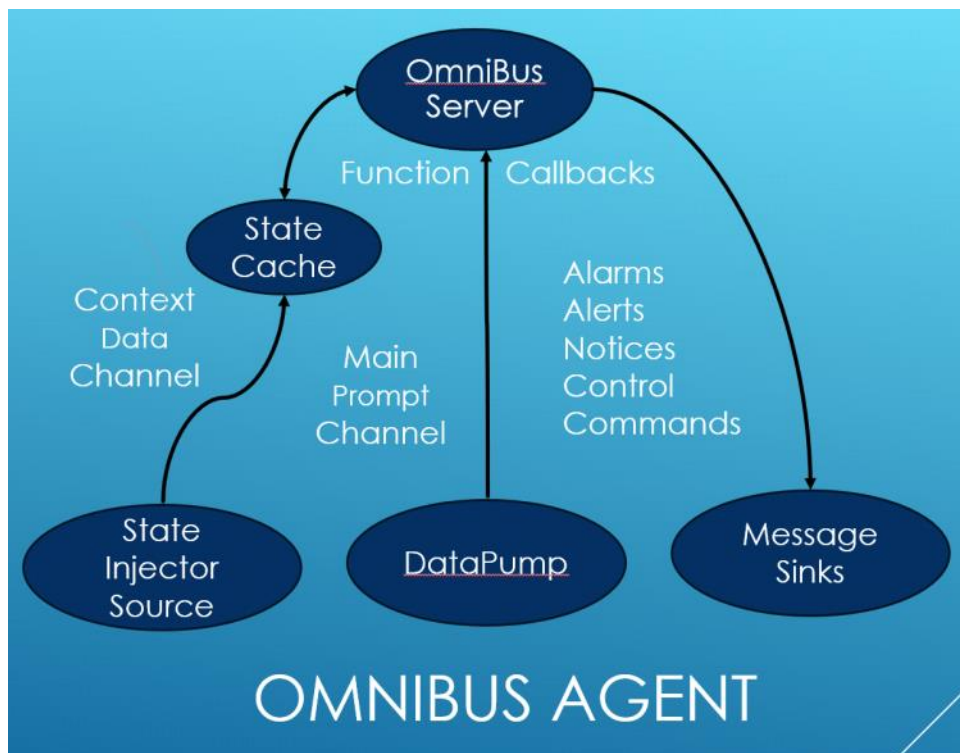
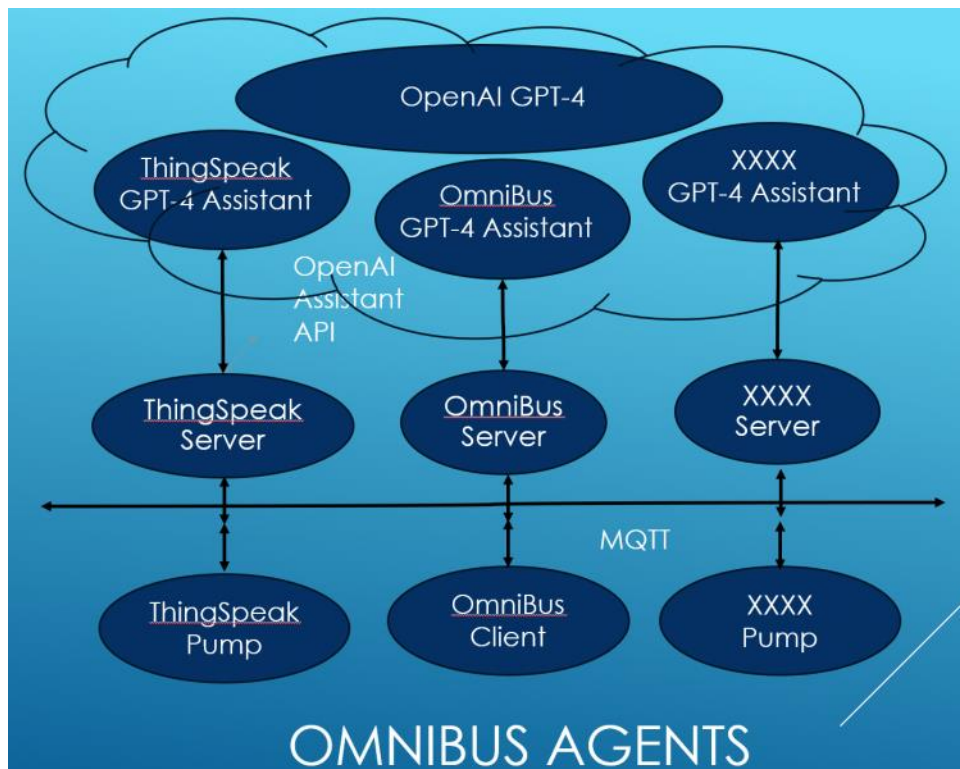
Monitor assistants: <https://platform.openai.com/playground>  
Monitor OpenAI API operations: <https://status.openai.com/>

### Repository

[https://github.com/upperbayhawk/C2\\_AIAgents.git](https://github.com/upperbayhawk/C2_AIAgents.git)

### Overview

<https://upperbay.com/v1/blogs/beachblogs/the-personal-advisory-sidekick>



## Code Summary

The code is 100% Python 3.12 developed on Windows 11 with Microsoft VSCode.

The code consists of a folder with a set of console apps that perform different functions associated with a specific GPT-4 Assistant that resides in the OpenAI cloud.

The GPT-4 assistant is created by `makeagent.py`. Use <https://platform.openai.com/assistants> to verify creation. All GPT-4 Assistant behavior is in this file. Delete the assistant using the OpenAI API site. See <https://platform.openai.com/docs/overview>.

The `runserver.py` is the primary local agent that communicates directly with the cloud assistant. It sends messages to the assistant and waits for responses. Other functions include:

- Prompt messages can be entered on the command line or received in an MQTT message.
- Maintains a local cache of tag/data values that is refreshed from the `runsource.py` agent. The `runsource.py` agent collects whatever external data it desires and sends that data in JSON to the `runserver.py` agent via an MQTT message. The data is then available to the GPT-4 assistant through function call-backs.
- Implements a set of call-back functions that the GPT-4 assistant can call whenever he/she/it desires. These represent external actions that the assistant can perform based on the combined prompt data submitted.

Messages handled include:

- COMMAND
- CONTROL
- NOTICE
- ALERT
- ALARM
- Command-line: `python runserver.py`

Agent libraries include:

- `Openailib.py`
  - Handles Assistant communications using the Assistant API
- `Xfunctionlib.py`
  - Handles function call-backs from the Assistant API
- `Xnetworklib.py`
  - Handles MQTT messaging
- `Xcachelib.py`
  - Handles `runserver.py` tag/value caching
- `Thingsspeaklib.py`
  - Handles data access to the MATLAB ThingSpeak data service,

(<https://thingspeak.com/>)

Other agents are optional. They are:

- runclient.py
  - Sends and receives prompts using MQTT to and from runserver.py
  - Command-line: python runclient.py
- runsink.py
  - Receives MQTT messages from GPT-4 function call-backs in runserver.py
  - Speaks the message received using OpenAI text-to-speech
  - Command-line: python runsink.py
- runsource.py
  - Sends JSON MQTT messages to runserver.py using tag/value. This data is available to GPT-4 using function call-backs and is accessible using the "getSensorValueByName" function.
  - Sends hard-coded sample data
  - Command-line: python runsource.py
- runpump.py
  - Periodically sends fully-dressed prompt messages with both text and data to runserver.py
  - Accesses thingspeak.com and sends public weather data from MATLAB in Boston, MA to runserver.py
  - Command-line: python runpump.py

These console apps have a simple structure that extends the OpenAI Assistant API in a straight forward if not elegant way. No attempt has been made to create production code and it deserves a good-dose of refactoring. The purpose of this code is to help experiment and explore the capabilities of GPT-4 while providing a means to integrate GPT4 with other apps, regardless of where the apps are running.

Each console app is multi-threaded. They create threads to handle keyboard and MQTT message dispatching based on topic. The topic tree is defined in config.py to avoid conflict between agents. Incoming MQTT messages are pushed into topic queues and the topic worker threads get and service the messages received. Other worker threads are spawned as needed. Most of the code is plumbing and can be changed to do whatever. Note that the OpenAI API uses the standard logging package so the agents use logbook to avoid conflict.

The runserver.py agent can only handle one prompt run at a time so interleaved prompts from multiple agents will result in bad behavior.

Readme.txt provides further information about the Python environment.

## Usage

- Git clone the repository to a local folder
  - git clone [https://github.com/upperbayhawk/C2\\_AIAgents.git](https://github.com/upperbayhawk/C2_AIAgents.git)
- Obtain an OpenAI key from your openai.com API account and set the OPENAI\_API\_KEY using the Windows Powershell "setx" command
- Create a new agent by cloning and renaming the "OmniBus" folder
- Rename agent\_name in config.py. Agent version is encoded in name. (e.g. OmniBus-1-0-0) The agents should now run under the new name.
- Modify makeagent.py to create the assistant with the characteristics desired
- Run makeagent.py and confirm creation on OpenAI API site
- Modify runserver.py and prompt data files to synch with assistant and change the MQTT hub IP address from "localhost" if an external hub is used
- Test using keyboard text prompts
- Run more agents as desired. Custom agents in any language interact with the runserver.py using MQTT messages.

=====

## NOTES:

The OpenAI Assistant API and the OpenAI runtime environment are BETA and subject to change, with intermittent downtime and slow response.

The following agents are play toys and can be disregarded:

- GridLoadMan
- HomePowerMan
- ThingsSpeakMan
- WeatherMan

