IOT Server documentation

Classes:

IOTServerRequestHandler(request, client_address, server):

- A subclass of socketserver.BaseRequestHandler to handles requests for the TCP server. It relies on the IOT server for message parsing and processing.
- **Attributes:**
- Self.Client_address: A tuple (ip_address, port) associated with the request. The ip_address is the IP address of the client that sent the request.
- Self.Request: The request string as received from the TCP socket.
- **Methods:**
- Self.etup(): Called right after a request is received and before the request is handled (processed). Return None.
- Self.Handle(): Called after setup, that is where the request processing usually occurs. Returns None.
- Self.Finish(): Called after handle, usually for cleaning up. Returns None.

ThreadingTCPServer(server_address, server_port, server_handler):

- A subclass of socketserver.ThreadingTCPServer, that defines a TCP server. The server is threaded, which means that it can process multiple requests at the same time (handles requests in threads).
- **Attributes:**
- Self._server_address (private attribute): Server IP address (default value: 192.168.137.1).
- Self._server_port (private attribute): The port number the server should bind and listen to (default value: 5070).
- Self._server_handler (private attribute): The handler used to process TCP requests.
- **Methods:**
- Self.server_forever(): Runs the server forever (blocking function call). Returns None.
- Self.server_close(): terminates the server. Returns None.

IOTServer(tcp_server_address, tcp_server_port, tcp_server_handler, web_server_address, web_server_port):

- A class that manages IOT server, it wraps a threaded TCP server, with a custom TCP Handler, and imports a webserver that serves as an interface to the user. The class handles the creation of the TCP server, starting the webserver, handling TCP requests from the connected nodes and manipulating nodes data on the server.
- **Class attributes:**
- IOTServer._ROOT_DIR: (private attribute) Root directory of the server (required to be consistent with the webserver root directory).
- IOTServer._ADDRESS: (private attribute) IP address of the TCP server (default: 192.168.137.1).
- IOTServer._NODE_PORT: (private attribute) Socket to connect to when sending data to IOT nodes (default: 1234).

- **Attributes:**
- Self._tcp_server_address: (private attribute) IP address of the TCP server (default: 192.168.137.1).
- Self._tcp_server_port: (private attribute) Port number of the TCP server (default: 5070).
- Self._server_handler: (private attribute) TCP server request handler (default: IOTServerRequestHandler).
- Self._web_server_address: (private attribute) IP address of the webserver (default: 192.168.137.1).
- Self._web_server_port: (private attribute) Webserver port number (default: 80).
- self._web_server: web server process handle. When the IOT server is initialized, this attribute is None, its value is set when the IOTServer.start method is called.
- **Methods:**
- Self.start(): Starts the TCP sever, and runs the webserver in a background process. Returns None.
- Self.tear_down(): Stops the TCP server, and kills the web server process. Returns None.
- IOTServer.process_message(iot_message): process the given IOT message object. Returns an iot_error object, that indicates if the message process was successful or not.
- IOTServer.process_request(iot_message): process the given request IOT message object. Returns an iot_error object, indicating if the request processing was successful or not.
- IOTServer.process_response(iot_message): process the given response IOT message. Returns None.
- Send_message(iot_message, dst_address, dst_port): Sends a given IOT message object to the given remote address and port (dst_address, dst_port). Returns an iot_error object, indicates if the message was sent successfully or not.
- IOTServer.get_module_files(module): Gets the paths to files associated with the given module. Returns the paths to the module files as a dictionary.
- IOTServer.check_moule_support(module): Checks if the given module is supported by the server. Returns an iot_error object.
- IOTServer.check_module_registration(module): Checks if the given module is already registered on the server. Returns as iot_error object.

Usage:

- Import the iot_server package in your code.
- Create a IOTServer object, and initialize it as follows:

```
import iot_server

def main():

  my_iot_server = iot_server.IOTServer(
    tcp_server_address="192.168.137.1",
    tcp_server_port=5070,
    tcp_server_handler=iot_server.IOTServerRequestHandler,
    web_server_address="192.168.137.1",
    web_server_port=80
  )

  try:
    my_iot_server.start()

  except Except as e:
    print(e):

  finally:
    my_iot_server.tear_down()


main()
```

Note:

- The IP given to the IOT server must be in the same LAN network as the IOT nodes. For example, a common wireless network from a wireless AP, or a PC's hotspot.
- In this example, I'm running the IOT server on my PC, and using the PC's hotspot as the common wireless LAN for the IOT server and the IOT nodes. The IP assigned to the server is the IP of the PC in the hotspot (from Adapter settings in Windows), and it will listen to connections coming from that IP. The IOT nodes connect to that hotspot as well, so they can "see" the IOT server.
- If you're going to use a wireless AP's network (or a wireless ADSL router), you've to make sure that the PC (or the central device which is running the IOT server) will be assigned a static address, so that IOT nodes can connect to it each time the network is reset.