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:

