Procès-Verbal de test : CClientMQTT

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Élément testé : | | Ces tests seront effectués sur les méthodes de la classe CClientMQTT, on travaillera sur la réception des données ainsi que les possibilités de création d’une nouvelle instance de la classe. | | | | | | |
| Objectif du test : | | Ce test vérifiera la bonne réception des données, ainsi que les différents codes d’erreur lors de leur validation. Les différents scénarios seront validés. | | | | | | |
| Nom du testeur : | | Théo Bourgoin | | | Date : | | 21/05/2025 | |
| Moyens mis en œuvre : | | Logiciel : Broker MQTT | | Matériel : Machine virtuelle hôte | | Outil de développement : Node JS & Express. | | |
| Procédure du test : | | | | | | | | |
| Id | Description du vecteur de test | | Résultat attendu | | Résultat obtenu  (voir le document de plan de test sur le github pour avoir l’échantillon exact des données) | | | Validation (O/N) |
| 1.1 | Instanciation de client MQTT par défaut  Ce test correspond au constructeur de la classe, on instancie un nouvel objet du client MQTT avec les valeurs proposées par défaut ainsi qu’un analyseur lambda (classe CAnalyse non incluse). | | topicName (donné par l’utilisateur)  addrBroker : 127.0.0.1 (localhost par défaut)  port : 1883 (port MQTT par défaut) | | « Client MQTT instancié : CClientMQTT {  topicName: 'test/topic',  objCAnalyse: {},  addrBroker: '127.0.0.1',  port: '1883',  client: null  } » | | | O |
| 1.2 | Client MQTT avec données personnalisées  Ce test correspond au constructeur de la classe, on instancie un nouvel objet de ClientMQTT, cette fois-ci avec des données personnalisées. L’analyseur sera toujours lambda. | | Les critères donnés doivent coïncider avec le retour  topicName (donné)  addrBroker : donné dans constructeur  port : donné dans constructeur | | « Client MQTT personnalisé instancié : CClientMQTT {  topicName: 'custom/topic',  objCAnalyse: {},  addrBroker: '192.168.1.100',  port: '1884',  client: null  } » | | | O |
| 2 | Connexion au broker MQTT  Ce test correspond à la méthode de connexion de la classe (\_connect) permettant de se connecter au broker MQTT. | | On s’attend à recevoir le message qui nous valide la connexion au broker ainsi que le topic :  « Connecté au broker MQTT  Abonné au topic : *topic name* » | | « Connecté au broker MQTT  Abonné au topic : test/topic » | | | O |
| 3 | Ce test correspond à la méthode de réception de messages MQTT. On prépare donc il client MQTT puis on lui envoie un message à l’aide du topic choisi et du broker mosquitto (ici en localhost) puis après avoir lancé le programme, copier-coller la ligne de broker mosquitto (précisée en dessous du code) dans un autre terminal. Ainsi on envoie des données dans un format accepté (JSON) et les données sont reçues par le client | | On s’attend à recevoir les données reçues sans erreur, et toujours dans le bon format.  Ex :  « {  Capteur : « … »,  TypeDeDonnee : « … »,  Date : « … »,  Valeur : « … »  } » | | « Message reçu : {  Capteur: 'Roseraie',  TypeDeDonnee: 'Temperature',  Date: '2025/05/19 15',  Valeur: 23.4  } » | | | O |
| 4 | On ne précise pas que les données doivent être reçues en JSON, une erreur de format est vite arrivée. Il faut donc vérifier ce qui nous est retourné lors de cette erreur. On répète donc la même procédure que le test précédent, mais cette fois-ci le message MQTT envoyé n’est pas bon. | | On s’attend à ce que les données soient refusées, avec le message d’erreur nous permettant de savoir ce qui ne va pas. | | « Erreur de traitement du message : Unexpected token m in JSON at position 0 » | | | O |
| Conclusion du test : | | Les tests des méthodes n’ont donné aucune erreur, ils ont tous été validés sans problème. On en conclut que la classe fonctionne sans problème, elle est donc validée. | | | | | | |