Annexe T.5

```
1   //  CommunicationSTM.ino
2   //
3   //  Description : fonctions liées à la communication entre l'ESP et le STM
4   //  Auteur : Perret Mélissa
5   //  Création :  22/09/2024
6   //  Modifications :  --
7
8   //  Version     :   V1.0
9   /*-------------------------------------------------------*/
10
11
12  #include "CommunicationSTM.h"
13  #include "CommunicationServeur.h"  // pour OCTET_DEBUT et OCTET_FIN
14  #include <HTTPClient.h>            // pour HTTPClient
15
16
17  //// Fonction ConfigurerUART: configuration de la transmission UART
18  //// Description: défini les pins utilisées pour l'écriture et la lecture UART
19  //// Entrées: -
20  //// Sorties: -
21  void ConfigurerUART() {
22    uart_config_t uart_config = {
23      .baud_rate = 115200,
24      .data_bits = UART_DATA_8_BITS,
25      .parity = UART_PARITY_DISABLE,
26      .stop_bits = UART_STOP_BITS_1,
27      .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
28    };
29
30    ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));         //
      Configuration UART
31    ESP_ERROR_CHECK(uart_set_pin(uart_num, TXD_PIN, RXD_PIN, -1, -1));  // Indiquer les
      pins utilisées pour l'écriture et la lecture UART
32  }
33
34  //// Fonction ReveillerSTM: logique pour réveiller le STM
35  //// Description: change l'état de la pin utilisée pour réveiller le STM
36  //// Entrées: -
37  //// Sorties: -
38  void ReveillerSTM() {
39    pinMode(PIN_POUR_REVEILLER_STM, OUTPUT);  // Important de mettre la pin en mode
      OUTPUT pour pouvoir modifier sa valeur
40    digitalWrite(PIN_POUR_REVEILLER_STM, HIGH);
41
42    if (MODE_DEBUG) {
43      printf("Reveil du STM en mettant la pin %d à l'état %d\n", PIN_POUR_REVEILLER_STM,
        digitalRead(PIN_POUR_REVEILLER_STM));
44    }
45  }
46
47  //// Fonction EnvoyerSignalFinSTM: logique pour indiquer au STM que l'ESP a fini de
    lui transmettre des trames UART
48  //// Description: met la pin utilisée pour réveiller le STM à l'état bas (la pin à
    l'état haut réveille le STM, et l'état bas lui indique qu'il a reçu toutes les trames
    nécessaires de la part de l'ESP)
49  //// Entrées: -
50  //// Sorties: -
51  void EnvoyerSignalFinSTM() {
52    pinMode(PIN_POUR_REVEILLER_STM, OUTPUT);  // Important de mettre la pin en mode
      OUTPUT pour pouvoir modifier sa valeur
53    digitalWrite(PIN_POUR_REVEILLER_STM, LOW);
54
55    if (MODE_DEBUG) {
56      printf("Envoie signal fin transmission au STM en mettant la pin %d à l'état %d\n",
        PIN_POUR_REVEILLER_STM, digitalRead(PIN_POUR_REVEILLER_STM));
57    }
58  }
59
60  //// Fonction ReceptionSTM: logique pour recevoir les trames UART de la part du STM
61  //// Description: réception des données UART
62  ////              analyse des données pour localiser les trames (caractères de début
    et de fin)
63  ////              analyse des trames reçues pour extraire et traiter les données
```

```
    reçues
64  ////              envoie de requête HTTP PUT pour transmettre les nouvelles données
    au serveur (états alarme seuils et batterie)
65  //// Entrées: Pointeur:HTTPClient http (client http utilisé pour effectuer les
    requêtes auprès du serveur)
66  //// Sorties: -
67  void ReceptionSTM(HTTPClient* http) {
68    int length = 0;
69
70    do {
71      // Réception UART
72      ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length));
73      length = uart_read_bytes(uart_num, receptionTramesUART, length,
        UART_RECEPTION_TIMEOUT_MS);
74
75      if (length > 0) {  // Si on a reçu quelque chose
76
77        // On parcourt le buffer pour trouver toutes les trames (identifiées par
          OCTET_DEBUT)
78        for (int indexDebutTrame = 0; indexDebutTrame < length; indexDebutTrame++) {
79          if (receptionTramesUART[indexDebutTrame] == OCTET_DEBUT) {  // Début de trame
            trouvé
80            int indexFinTrame = indexDebutTrame + TAILLE_TRAME - 1;
81            if (indexFinTrame < length && receptionTramesUART[indexFinTrame] ==
              OCTET_FIN)  // Si on a bien une fin de trame comme attendu
82            {
83              uint8_t index = receptionTramesUART[indexDebutTrame + 1];  // Deuxième
                octet de la trame correspond à l'identifiant de la trame
84              // Troisème octet correspond à un séparateur ':' pour faciltier la
                lecture des trames lors du debuggage
85
86              // Explications pour memcpy :
                https://en.cppreference.com/w/cpp/string/byte/memcpy
87              // Les 8 octets suivants (octets 4 à 11) correspondent à la valeur de la
                trame
88              double valeur;
89              memcpy(&valeur, &receptionTramesUART[indexDebutTrame + 3], sizeof(double
                ));  // Copier les 8 octets de la trame représentant la valeur dans la
                variable valeur
90
91              if (MODE_DEBUG) {
92                printf("Nouvelle valeur provenant du STM. Nom:%s Avant:%lf
                  Maintenant:%lf\n", valeursSTM[index].nom.c_str(), valeursSTM[index].
                  valeur, valeur);
93              }
94
95              valeursSTM[index].valeur = valeur;  // Mise à jour de la valeur
96
97              // Envoie de la requête PUT pour transmettre la nouvelle valeur au serveur
98              String putString = valeursSTM[index].nom + "=" + valeur;
99
100             http->setTimeout(HTTP_TIMEOUT_MS);
101
102             int httpCode;
103             int nombreTentativesHTTP = 0;
104             do {
105
106               if(nombreTentativesHTTP > 0)
107               {
108                 delay(DELAI_NOUVELLE_TENTATIVE_HTTP_MS); // S'il s'agit d'une
                  nouvelle tentative, attendre un peu avant de réessayer
109               }
110
111               nombreTentativesHTTP++;
112
113               httpCode = http->PUT(putString);
114
115               // httpCode est négatif en cas d'erreur
116               if (httpCode > 0) {
117                 // Le header HTTP à été envoyé et une réponse du serveur à été reçue
118
119                 if (MODE_DEBUG) {
120                   printf("[HTTP] Requete PUT... code: %d\n", httpCode);
```

```
121                 }
122
123                 if (httpCode == HTTP_CODE_OK) {
124                   String reponseServeur = http->getString();
125                   if (MODE_DEBUG) {
126                     printf("[HTTP] Requete PUT reponse serveur: %s\n", reponseServeur.
                       c_str());
127                   }
128                 }
129               } else {
130                 printf("[HTTP] Requete PUT... echec, erreur: %s\n", http->
                  errorToString(httpCode).c_str());
131               }
132             } while (httpCode <= 0 && nombreTentativesHTTP <
                NOMBRE_TENTATIVES_REQUETES);
133           }
134         }
135       }
136     }
137   } while (length > 0);  // Continuer tant qu'on reçoit quelque chose
138 }
```