Port Serial – Utilisation de la classe SerialCommandRead

L'utilisation de l'UART pour communiquer avec un périphérique (par ex. avec un autre Arduino ou un PC) est un moyen simple d'échanger des données. Il permet aisément de définir son propre protocole.

Ce document donne un exemple d'utilisation simplifiée de la classe SerialCommandRead disponible dans la librairie cZarduiBase **à partir de la version v4e.**

Il part du principe que vous connaissez déjà :

- L'utilisation de la librairie hardware Serial de l'Arduino
- L'utilisation de chaînes de caractères du type String en langage C++
- L'utilisation d'un câble UART to USB afin de connecter votre Arduino à un PC par un port COM virtuel
- La différence entre l'échange de donnée en mode « binaire » ou « texte »
- La mesure et le décodage d'une trame de donnée à l'aide d'un analyseur logique

Mode "command"

Ce mode part du principe qu'une ***commande*** est :

- ✓ Une chaîne de caractères, donc au format texte
- ✓ Comprise entre deux délimiteurs
  - ***start***, optionnel, une chaîne de un ou plusieurs caractères
  - ***end*** obligatoire, une chaîne de un ou plusieurs caractères
  - ***ATTENTION*** les chaînes ***start*** ou ***end*** ne doivent jamais de retrouver dans la commande

Exemples de commandes : "L1 on\r\n"
"L1 off\r\n"
"@Delay 500\r"

ATTENTION : "@Delay\r 500\r" FAUX
"@Delay\r 500\r\n" OK

La classe *SerialCommandRead* permet de gérer ce mode de manière simplifiée pour votre Arduino, particulièrement pour la réception.

D'autre part le terminal *TermiZord* permet à un PC de communiquer avec un Arduino en prenant en compte le mode *command*. La mise au point votre application Arduino est ainsi facilitée.

Classe SerialCommandRead

Le mode *"command"* utilisé par *SerialCommandRead* (et *TermiZord*) permet de gérer automatiquement :

- ✓ La réception des commandes reçues par l'UART, soit la lecture des caractères jusqu'au *end* y compris
- ✓ La lecture de la commande reçue sans les délimiteurs *start* et *end*
- ✓ La gestion des erreurs
  - Si *start* est défini mais absent
  - Si *start* est défini, contrôle et suppression possible des caractères avant
  - Gestion d'un timeout, si end n'est pas reçu dans le délai défini

Notez que cette classe utilise la librairie Serial de Arduino. Les caractères reçus sont automatiquement mémorisés dans une mémoire tampon (buffer) de 64 octets, cela même si votre application est « bloquée » par d'autres tâches.

Cela permet à *SerialCommandRead* de recevoir une commande (ou plusieurs) même si la précédente n'est pas encore traitée.

Exemple

Voici un exemple simplifié d'utilisation de cette classe. La commande est envoyée par un PC connecté sur le port Serial3 de l'Arduino à 38400 bauds. Elle commence par **@** et se termine par **\r\n** et est reçue dans un délai inférieur à 200ms.

```
// Instanciation de l'objet scr
// Paramètres : - adresse du port série à utiliser
//              - Délimiteurs (de type string, plusieurs car. autorisés)
//                - start, mettre "" si aucun
//                - stop, doit être <> "", sinon prend "\r"
//              - timeout en ms
//                - Optionnel, 100ms par défaut
//              - commandSize (optionnel, pour utilisateur avancé)
// ATTENTION    - Il est absolument INTERDIT de lire le port avec Serial de Arduino !
//             => Utiliser les méthodes de scr.
SerialCommandRead scr(&Serial3, "@", "\r\n", 200);

void setup() {

    // Initialise l'objet scr à la vitesse désirée (et vide les buffers)
    scr.begin(38400);

}

void loop() {
    static int iCpt = 0;
    String strCommand;

    // Déclare la variable pour mémoriser l'état de l'objet scr
    SerialCommandRead::state dataReady;

    // Une commande a-t-elle été reçue ?
    dataReady = scr.available();

    // dataReady vaut SerialCommandRead::state:: ------------------------------------------
    // OK                "start" et "end" ont été reçus
    // CHAR_BEFORE_START   commande OK, mais "start" est précédé par d'autres caractères
    // WAITING           attente de la réception du 1er car.
    // PROCESSING        car. reçus, attente de "end"
    // ERROR_NO_START    "end" est reçu, mais "start" est absent
    // ERROR_TIMEOUT     le délai timeout à été dépassé depuis le dernier caractère
    //                   reçu et "end" est absent

    // Traitement simplifié ---------------------------------------------------------------
    //   - Si un start est utilisé, on ignore les car. avant
    //   - Le timeout est suffisamment grand pour recevoir une commande entière
    // Dans ce cas il suffit de tester dataReady :
    //   Si < WAITING      la commande peut être lue avec read()
    //   Si > PROCESSING   erreur, vider la commande en cours avec restart()

    if (dataReady < SerialCommandRead::state::WAITING) {

        // strCommand contient la commande reçue
        // (sans les car. avant et y compris le "start" ni le "end")
        // L'analyse de la commande suivante est démarrée
        strCommand = scr.read();

        // La commande peut être traitée
        ProcessingCommand(strCommand);
    }

    if (dataReady > SerialCommandRead::state::PROCESSING) {
        // Une erreur est survenue
        // - Pas de start, si start <> ""
        // - Pas de end dans les temps
        // On ignore les car. reçus et recommence une nouvelle analyse pour la commande suivante
        scr.restart();

        // On gère l'erreur, ici en envoyant un message
        // Les délimiteurs start et stop sont automatiquement ajoutés
        scr.print("Command error)");
    }

}
```