Cette étude porte sur l'afficheur **AM-03127-LCD** pourvu d'une seule ligne d'affichage qui permet de visualiser des textes défilants avec plusieurs possibilités d'affichage paramétrables. L'afficheur est connecté à une **« Raspberry »** sur le port USB qui est alors considéré comme un simple port série.

Tous nos premiers travaux seront effectués directement sur la « Raspberry » en mode console. Il est possible de développer les sources sur un ordinateur hôte avec l'éditeur que nous utilisons habituellement et de les déployer ensuite à l'aide de la commande « scp ».

À la fin de notre étude, nous réaliserons également un service permettant de gérer l'afficheur à distance.

INSTALLATION DES PAQUETS NÉCESSAIRES AU FONCTIONNEMENT DE L'INTERFACE SÉRIE AVEC PYTHON

Afin de pouvoir développer des applications en Python sur le port USB (similaire à une interface série), nous devons intégrer les paquets suivants :

```
$ sudo apt-get install python3          // Installation de python version 3
$ sudo apt-get install python3-serial // Intégration du module associé à l'interface série pour cette version là.
```

Les deux installations peuvent se faire en une seule ligne de commande. Il suffit de désigner les paquets à installer les uns à la suite des autres.

DESCRIPTION DES TRAMES À ENVOYER POUR L'AFFICHEUR AM-03127-LED

Nous communiquons avec l'afficheur avec le port USB. Pour cela vous devez fournir une trame qui sera ensuite interprétée par l'afficheur lui-même pour répondre à l'attente du client. La trame à envoyer est une chaîne de caractères avec un certain nombre de balises. Voici quelques remarques que nous pouvons faire sur la constitution de cette trame :

Requête	Description de la trame
Identification de l'afficheur	*Il est possible de connecter plusieurs afficheurs sur la liaison série, il faut donc pouvoir choisir celui avec lequel nous devons communiquer. Chaque afficheur possède donc un identifiant qui d'origine est à **01**. Nous pouvons également communiquer avec l'ensemble des afficheurs sur une même liaison série en donnant comme identifiant la valeur **00** (tout le monde). L'ossature minimum de la trame est donc la suivante :* ***<IDXX> . . . CS<E>*** ✗ *La première balise **<ID>** (pour **IDentifiant**) comporte le numéro de l'afficheur avec lequel communiquer.* ✗ *La dernière balise **<E>** (pour **End**) termine systématiquement toute trame quelque soit ce que vous désirez faire.* ✗ *La somme de contrôle **CS** évalue le contenu de la trame placé entre les deux balises précédentes. Le contrôle ici est de type **XOR** codé sur un simple octet mais traduit à l'aide de deux caractères consécutifs donnant les valeurs hexadécimales équivalentes (toute la trame est une chaîne de caractères).* *Ainsi, si vous désirez communiquer avec le premier afficheur de la liste :* ***<ID01> . . . CS<E>***
Afficher un message	*L'une des fonctions principales d'un afficheur est bien entendu d'afficher le message de votre choix. Voici la trame minimale que vous devez placer entre les deux balises d'identification et de fin :* ***<Ln><Pn><FX><MX><WX><FY>Votre message*** ✗ *Ligne de l'afficheur **<Lx>** (pour **Ligne**) : l'afficheur que nous utilisons ne possède qu'une seule ligne, donc la valeur numérique à spécifier sera toujours 1 : **<L1>*** ✗ *Page de l'afficheur **<Px>** (pour **Page**) : Il est possible d'afficher jusqu'à 26 pages sur un même afficheur. Ces pages seront lues les unes à la suite des autres. Pour identifier la page sur laquelle nous désirons proposer un nouveau message, vous devez spécifier une des lettres de l'alphabet **A ... Z**.* ✗ *Fonction d'apparition du message **<Fx>** : Il existe plusieurs façons d'afficher votre message : immédiatement, en le faisant défiler vers la droite ou vers la gauche, en le faisant apparaître depuis la partie haute de l'afficheur, etc.* ✗ ***<FA>** : Le message apparaît instantanément sans défilement.* ✗ ***<FB>** : Le message apparaît instantanément puis disparaît suivant les quatre orientations : gauche, haut, droite et bas.* ✗ ***<FC>** : Le message défile vers le haut.* ✗ ***<FD>** : Le message défile vers le bas.* ✗ ***<FE>** : Le message défile vers la gauche.* ✗ ***<FF>** : Le message défile vers la droite.* ✗ ***<FG>** : Le message apparaît au centre et et disparaît vers le haut et le bas.* ✗ ***<FH>** : Le message apparaît depuis le haut et le bas.* ✗ ***<FI>** : Le message défile du bas vers le haut.* ✗ ***<FJ>** : Le message défile du haut vers le bas.* ✗ ***<FP>** : Affichage aléatoire des pixels du message.* ✗ *Méthode d'affichage **<Mx>** : Il est possible d'afficher le message normalement ou avec un clignotement plus ou moins rapide : **<MA>** **: Normal,** **<MB>** **: Clignotant**.* ✗ *Temps d'apparition du message **<Wx>** **(**pour **Wait**) : Vous spécifiez à l'aide d'un chiffre le temps de lecture de la page en cours : **<WA>** **: 0,5 s,** **<WB>** **: 1s,** **<WC>** **: 2s, …,** **<WZ>** **: 25s.*** ✗ *Disparition du message **<Fx>** : Comme pour le faire apparaître, vous devez spécifier sa disparition avec le même type de balise que précédemment.*
Autres balises utiles pour la gestion des messages	*Il est possible de rajouter une balise à votre message qui offre une fonctionnalité supplémentaire. Voici les balises que nous pouvons rajouter :* ***<AX>** : Modifie la police d'affichage, les caractères seront plus ou moins gros, de **<AA>** **à** **<AE>.*** ***<BX>** : Émet un son plus ou moins long sur la page en cours : **<BA>** **: 0,5 s,** **<BB>** **: 1s,** **<BC>** **: 2s, …,** **<BZ>** **: 25s.*** ***<CX>** : Choisit une couleur : **<CB>** **: Rouge,** **<CE>** **: Vert,** **<CH>** **: Orange,** **<CS>** **: Arc-en-ciel.*** ***<KX>** : Permet d'intégrer une date ou une heure préformatée : **<KD>** **: Date,** **<KT>** **: Heure.***
Régler la date et l'heure de l'afficheur	***<SC>AASSMMDDHHmmss** : Avec : **AA** **: l'année,** **SS** **: le jour de la semaine (01 lundi - 07 dimanche),** **MM** **: le mois,** **DD**: **le jour du mois,** **HH** **: l'heure,** **mm** **: la minute,** **ss** **: la seconde.***
Organisation des pages affichées	***<TA>AAMMDDHHmmAAMMDDHHmm...PPP...** : Il est possible de spécifier les pages que vous souhaitez voir afficher, à partir de quel moment et dans quel ordre vous désirez les faire apparaître. Exemple, **<TA>0000000000990000000AB** pour afficher les deux premières pages (**A** et **B**) dans le siècle en cours (**00** à **99**).*

Supprimer une page	*<DLxPn> : Vous pouvez supprimer le contenu actuel d'une page. Attention, cela ne veut pas dire qu'elle ne sera plus affichée. Si vous ne désirez plus la voir du tout, il est nécessaire de passer pas la commande d'organisation des pages, de <DL1PA> à <DL1PZ>.*
Tout supprimer	*<D*> : Si vous désirez retrouver votre afficheur tel qu'il était au départ, vous pouvez tout réinitialiser grâce à cette balise.*
Page par défaut	*<RPn> : Si vous ne désirez avoir qu'une seule page à afficher, vous pouvez utiliser cette commande, de <RPA> à <RPZ>.*
Agir sur la luminosité globale	*<BX> : Enfin, vous pouvez contrôler la luminosité de l'afficheur suivant quatre valeurs possibles de 25% à 100%. <BA> : 100 %, <BB> : 75 %, <BC> : 50 %, <BD> : 25 %.*

À titre d'exemple, voici deux trames qui permettent d'afficher un message de bienvenue défilant de droite à gauche qui dure deux secondes et de couleur verte :

```
$ <ID01><L1><PA><FE><MA><WC><FE><CE>bonjour11<E>
$ <ID01><L1><PA><FE><MA><WC><FE><CE>salut0F<E>
```

Dans le tableau ci-dessous, je vous propose un certain nombre d'exemples qui nous permettent de mieux comprendre la mise en œuvre de ces trames, cette fois-ci, sans la balise d'identification et de fin et sans le checksum.

Trame	Description de la trame
`<L1><PA><FA><MB><WF><FF><CB>Bienvenue`	*Afficher un message quelconque, clignotant, de couleur rouge qui s'affiche immédiatement sans défilement, qui dure cinq secondes et qui défile ensuite vers la droite.*
`<L1><PA><FD><MA><WE><FC><AB>Bienvenue`	*Afficher un message quelconque en gras pendant **4** secondes qui apparaît progressivement de la partie haute de l'écran sans mouvement du message lui-même et qui disparaît progressivement de nouveau vers le haut de l'écran.*
`<L1><PA><FE><MA><WC><FE><CS>Bienvenue`	*Afficher le message **« Bienvenue »** de toutes les couleurs.*
`<TA>00000000009900000000ABC`	*Faire en sorte que les trois premières pages de l'afficheur soient opérationnelles pour que ces pages s'affichent indéfiniment, il faut que l'année de début soit **00** et l'année de fin **99**.*
`<DL1PC>`	*Supprimer la troisième page.*
`<D*>`	*Tout supprimer.*

`<TA>00000000009900000000AB`
`<L1><PA><FE><MA><WC><FE><KD>`
`<L1><PB><FE><MA><WC><FE><KT>`
	Afficher la date d'aujourd'hui sur la première page et l'heure actuelle sur la deuxième page.

`<BC>`	*Agir sur la luminosité de l'écran pour que l'éclairage soit seulement à 25% du maximum.*

PREMIER DÉVELOPPEMENT – AFFICHER UN MESSAGE DE BIENVENUE AVEC UN SCRIPT PYTHON

Maintenant que nous connaissons un peu mieux l'architecture des trames à envoyer, je vous propose de réaliser un script qui permet d'afficher un message de bienvenue à l'aide des deux exemples vus ci-dessus. Pour cela, vous devez connecter votre afficheur sur une prise **« USB »** de la Raspberry (ou de votre ordinateur).

*Sous **Linux**, la première connexion **USB** est souvent appelée **« /dev/ttyUSB0 »**. Cette connexion **USB** est considérée comme une interface série. Vous devez normalement régler votre connexion en choisissant convenablement la vitesse (très souvent, par défaut : 9600 bauds) le bit de parité, etc.*

*Dans notre script, nous devons utiliser la classe **« Serial »** qui représente une interface série quelconque, très simple à mettre en œuvre et qui possède quelques méthodes qui nous permettent de résoudre tout ce que nous avons besoin pour gérer cet afficheur :*

- ***Le constructeur :** avec lequel vous spécifier le nom de votre connexion **USB**. Lorsque vous créez votre objet à l'aide de cette phase de construction, vous êtes automatiquement connecté (si bien sûr votre connexion est valide).*
- ***isOpen():** permet de contrôler si votre connexion s'est bien réalisée correctement.*
- ***write():** envoi un flux d'octets au travers de cette connexion. Comme pour le développement réseau, si vous désirez envoyer un texte, vous devez le transformer en flux d'octets au moyen de la méthode **« encode() »**.*

afficheur.py

```
from serial import Serial
import time

bonjour = '<ID01><L1><PA><FE><MA><WC><FE><CE>bonjour11<E>'
salut = '<ID01><L1><PA><FE><MA><WC><FE><CE>salut0F<E>'

afficheur = Serial('/dev/ttyUSB0')

if afficheur.isOpen():
   print('afficheur connecté')
   afficheur.write(bonjour.encode('utf8'))
   time.sleep(0.05) // il faut laisser le temps de la transmission avant la clôture (50 ms)

else: print('afficheur non connecté')
```

*Comme nous communiquons avec une interface série (communication lente) vous devez laisser le temps du transfert et l'interprétation de la trame par l'afficheur. Un bon compromis est de fixer un temps d'attente de **50ms** après chaque trame envoyée.*

Une fois que vous avez édité ce script sur votre ordinateur hôte, vous devez ensuite le déployer sur votre cible (la Raspberry) à l'aide de la commande « scp », comme ci-dessous :

```
$ scp afficheur.py pi@192.168.1.22:/home/pi/python  // copie du script  dans le bon répertoire de la Raspberry
```

CALCUL AUTOMATIQUE DU CHECKSUM

Je vous propose maintenant de réaliser une fonction qui calcule automatiquement le **« checksum »** d'une trame proposée en paramètre sans tenir compte de l'en-tête et de l'en-queue de la trame.

*Je rappelle que la somme de contrôle **CS** évalue le contenu de la trame placé entre les deux balises **<ID01>** et **<E>**. Le contrôle ici est de type **XOR** codé sur un simple octet mais traduit à l'aide de deux caractères consécutifs donnant les valeurs hexadécimales équivalentes (toute la trame est une chaîne de caractères).*

afficheur.py

```
bonjour = '<L1><PA><FE><MA><WC><FE><CE>bonjour'
salut = '<L1><PA><FE><MA><WC><FE><CE>salut'

def checksum(trame):
  calcul = 0;
  for octet in trame.encode('utf8') : calcul ^= octet
  bas = calcul & 0b00001111
  haut = (calcul & 0b11110000) >> 4
  return "{:X}{:X}".format(haut, bas)

if __name__ == "__main__":
  print("<ID01>"+bonjour+checksum(bonjour)+"<E>")
  print("<ID01>"+salut+checksum(salut)+"<E>")
```

*Pour l'instant, nous n'agissons pas sur l'afficheur, c'est juste pour vérifier que le calcul du **« checksum »** est correct.*

PASSAGE DE LA TRAME EN LIGNE DE COMMANDE

Au lieu de placer les trames directement dans le code, je vous propose maintenant de faire en sorte que la trame qui pilote l'afficheur soit donnée en paramètre à la suite du programme que vous exécutez dans le mode console afin de pouvoir valider les exemples sur lesquels nous avons travaillés précédemment.

*Pour cela, vous pouvez utiliser la liste **« sys.argv »** sachant que chaque élément de cette liste sont des chaînes de caractères correspondant à l'ensemble des mots séparés par un espace dans votre ligne de commande, et sachant que le premier élément de cette liste est le nom du programme lui même.*

afficheur.py

```
import sys

def checksum(trame):
  calcul = 0;
  for octet in trame.encode('utf8') : calcul ^= octet
  bas = calcul & 0b00001111
  haut = (calcul & 0b11110000) >> 4
  return "{:X}{:X}".format(haut, bas)

if __name__ == "__main__":
  if len(sys.argv) > 1:
    trame = sys.argv[1]
    print("<ID01>"+trame+checksum(trame)+"<E>")
  else:
    print('Il faut au moins un paramètre')
```

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
manu@HPE-120fr ~/CloudStation/ProjetsPython/Afficheur $ python3 afficheur.py "<L1><PA><FE><MA><WC><FE><CE>bonjour"
<ID01><L1><PA><FE><MA><WC><FE><CE>bonjour11<E>
manu@HPE-120fr ~/CloudStation/ProjetsPython/Afficheur $ python3 afficheur.py "<L1><PA><FE><MA><WC><FE><CE>salut"
<ID01><L1><PA><FE><MA><WC><FE><CE>salut0F<E>
manu@HPE-120fr ~/CloudStation/ProjetsPython/Afficheur $
```

Ce programme nous permet juste de vérifier que nous prenons bien en compte la trame proposée en argument de la ligne de commande sans piloter l'afficheur pour l'instant.

Le programme suivant permet de faire le même traitement mais cette fois-ci intervient sur l'afficheur. Vous pouvez alors visualiser en temps réel la bonne architecture des trames. Remarquez au passage, que nous avons conçu une nouvelle fonction « envoiTrame() » qui permet de soumettre la totalité de la trame à l'afficheur connecté :

afficheur.py

```
from serial import Serial
import time
import sys
```

```
def checksum(trame):
  calcul = 0;
  for octet in trame.encode('utf8') : calcul ^= octet
  bas = calcul & 0b00001111
  haut = (calcul & 0b11110000) >> 4
  return "{:X}{:X}".format(haut, bas)

def envoiTrame(trame):
  afficheur = Serial('/dev/ttyUSB0')
  if afficheur.isOpen():
    print('Afficheur connecté')
    afficheur.write(trame.encode('utf8'))
    time.sleep(0.05)
  else:
    print('Afficheur non connecté')

if __name__ == "__main__":
  if len(sys.argv) > 1:
    trame = sys.argv[1]
    envoiTrame("<ID01>"+trame+checksum(trame)+"<E>")
  else:
    print('Il faut au moins un paramètre')
```

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
pi@raspdev:~/logiciels $ python3 afficheur.py "<L1><PA><FE><MA><WC><FE><CE>Bienvenue <CH><KD> <CB>(<KT>)"
Afficheur connecté
pi@raspdev:~/logiciels $
```

À titre d'exemple, voici la trame qu'il faut soumettre pour afficher un message défilant de bienvenue en vert, suivi de la date du jour en orange, suivie de l'heure entre parenthèses et en rouge.

RÉGLER LA DATE ET L'HEURE DE L'AFFICHEUR ET LES VISUALISER SUR DEUX PAGES CONSÉCUTIVES

Nous allons nous servir des recherches précédentes pour mettre en œuvre un nouveau script qui sera capable d'envoyer plusieurs trames consécutives (avec un délais de **50ms** entre chaque trame). Il s'agit de récupérer l'heure et la date actuelle de la Raspberry pour régler l'afficheur en conséquence et proposer un affichage cette fois-ci sur deux pages, respectivement avec la date et l'heure, avec des réglages différents pour chacun.

afficheur.py

```
from serial import Serial
from time import *

afficheur = Serial('/dev/ttyUSB0')

def checksum(trame):
  calcul = 0;
  for octet in trame.encode('utf8') : calcul ^= octet
  bas = calcul & 0b00001111
  haut = (calcul & 0b11110000) >> 4
  return "{:X}{:X}".format(haut, bas)

def envoiTrame(trame):
  if afficheur.isOpen():
    afficheur.write(trame.encode('utf8'))
    sleep(0.05)
  else:
    print('Afficheur non connecté')

if __name__ == "__main__":
  trame = "<SC>"+strftime('%y0%w%m%d%H%M%S')
  envoiTrame("<ID01>"+trame+checksum(trame)+"<E>")
  trame = "<TA>0000000000990000000AB"
  envoiTrame("<ID01>"+trame+checksum(trame)+"<E>")
  trame = "<L1><PA><FD><MA><WC><FC><CE>Date <KD>"
  envoiTrame("<ID01>"+trame+checksum(trame)+"<E>")
  trame = "<L1><PB><FC><MB><WE><FJ><CB>Heure <KT>"
  envoiTrame("<ID01>"+trame+checksum(trame)+"<E>")
```

*Nous avons besoin de la fonction **« time.strftime() »** que nous connaissons déjà. J'ai utilisé d'autres marqueurs utiles pour notre projet, avec **«%y »** pour les deux chiffres de l'année, **«%w »** pour le chiffre correspondant au jour de la semaine et **«%m »** pour les deux chiffres du mois, les autres marqueurs étant connus.*

PROPOSER PLUSIEURS MESSAGES PROPOSÉS EN LIGNE DE COMMANDE

Nous allons maintenant mixer les deux projets précédents afin de prévoir la visualisation de plusieurs messages donnés en paramètre de la ligne de commande afin qu'ils apparaissent successivement sur l'afficheur (sur plusieurs pages). Le nombre de messages à afficher est totalement libre (dans la limite des 26 pages possibles). Voici un exemple ci-après :

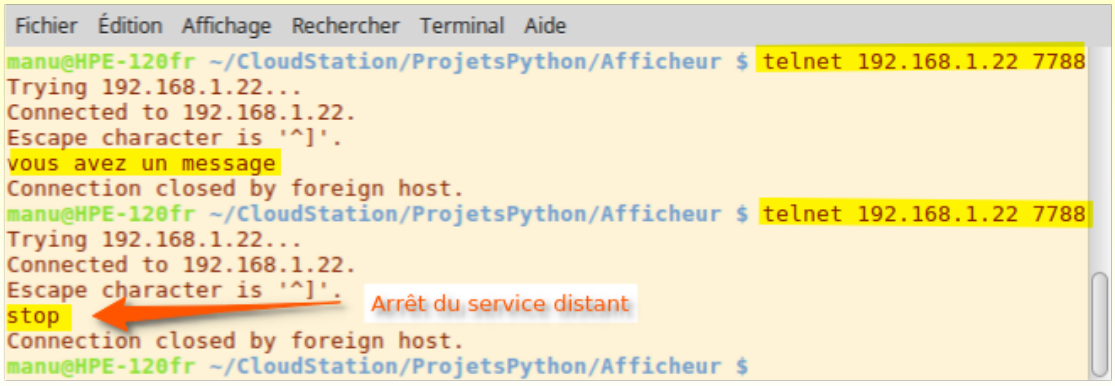
```
$ python3 afficheur.py « bonjour » « salut » « bienvenue » // visualisation de trois messages sur l'afficheur
```

afficheur.py

```
from serial import Serial
from time import *
import sys

afficheur = Serial('/dev/ttyUSB0')
def checksum(trame):
  calcul = 0;
  for octet in trame.encode('utf8') : calcul ^= octet
  bas = calcul & 0b00001111
  haut = (calcul & 0b11110000) >> 4
  return "{:X}{:X}".format(haut, bas)

def envoiTrame(trame):
  if afficheur.isOpen():
    afficheur.write(trame.encode('utf8'))
    sleep(0.07) // il faut laisser le temps de la transmission avant le message suivant  (ici 70ms)
  else:
    print('Afficheur non connecté')

if __name__ == "__main__":
  nombre = len(sys.argv)
  if nombre>1:
    pages = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    trame = '<TA>0000000009900000000'+pages[0:nombre-1]
    envoiTrame("<ID01>"+trame+checksum(trame)+"<E>")
    page = 0
    for message in sys.argv[1:nombre]:
      trame = "<L1><P"+pages[page]+"><FD><MA><WC><FC><CE>"+message
      envoiTrame("<ID01>"+trame+checksum(trame)+"<E>")
      page += 1
  else:
    print('Il faut au moins un paramètre')
```

PROPOSER UN SERVICE D'AFFICHAGE ACCESSIBLE PAR LE RÉSEAU

Pour clôturer ce sujet, je vous propose de réaliser un service sur la Raspberry qui permet d'afficher un message envoyé sur le réseau (communication distante) avec la date du jour et l'heure actuelle (le tout sur trois pages). Il est possible d'arrêter définitivement le service en soumettant le message **« stop »**.



*Pour tester ce service, nous nous servirons d'un simple client **« telnet »**, comme cela vous est montré ci-dessus :*

afficheur.py

```
from serial import Serial
from time import *
from sys import exit
from socket import *

# Création des objets nécessaires au bon fonctionnement de l'application distante
afficheur = Serial('/dev/ttyUSB0')
service = socket()

# Calcul du checksum de la trame fourni en paramètre
def checksum(trame):
  calcul = 0;
  for octet in trame.encode('utf8') : calcul ^= octet
  bas = calcul & 0b00001111
  haut = (calcul & 0b11110000) >> 4
  return "{:X}{:X}".format(haut, bas)
```

```
# Envoi de la trame complète en relation avec la commande souhaitée
def envoiTrame(commande):
  if afficheur.isOpen():
    trame = "<ID01>"+commande+checksum(commande)+"<E>"
    afficheur.write(trame.encode('utf8'))
    sleep(0.07)
  else:
    print('Afficheur non connecté')

# Activation de trois pages avec un message d'invite, une date et une heure
def démarrage():
  envoiTrame("<TA>0000000000990000000ABC")
  envoiTrame("<L1><PA><FE><MA><WE><FE><CE>Introduisez votre message")
  envoiTrame("<L1><PB><FD><MA><WC><FC><CH><KD>")
  envoiTrame("<L1><PC><FC><MA><WC><FJ><CB><KT>")

# Lancement du service avec prise en compte d'un nouveau message éventuel
def lancerService():
  service.bind(('', 7788))
  service.listen(5)
  fin = False
  while not fin:
    client, adresse = service.accept()
    message = client.recv(1024).decode('utf8').strip()
    if message.lower() != 'stop':
      envoiTrame("<L1><PA><FE><MA><WE><FE><CE>"+message)
      client.close()
    else:
      fin = True
      service.close()

# Programme principal
if __name__ == "__main__":
  démarrage()
  lancerService()
```